

nibble[®] EXPRESS



VOLUME III

NIBBLE EXPRESS VOLUME III

**The Best Articles and Programs
from NIBBLE Magazine 1982**

**Compiled by the Editors of
NIBBLE Magazine**

NIBBLE EXPRESS VOLUME III

**The Best Articles and Programs
from NIBBLE Magazine 1982**

**Compiled by the Editors of
NIBBLE Magazine**

EXPRESS Volume III

Table of Contents

Key Perfect Tables	4
Apple Turtle Graphics by David Krathwhol	11
Applesoft Line Cruncher by Alan and Valerie Floeter	17
TRAC Income System by Chris Exner	23
MAMA by Sandy Mossberg	28
Apple Flash by Todd Livesey	28
Amper Free Space Catalog by Ben Colley	30
M.L.E. Machine Language Editor by Doug Sprinkle	32
M.L.E. Driver by William Doyle	39
Apple Calendar by Glenn Teman	40
Peaceful Coexistence by Robert Anderson	46
Apple Slugger by Robert Devine	47
Reverse Key by Phillip Rowley	49
Amper Find by Joe Brooks	50
Bond Manager by Doug Sprinkle	53
DOS 3.2 by Joel Buckley	59
Amper Speed by William Reynolds	63
Apple Darts by Robert Devine	67
Micro-Calc by Stephen Bohn	69
Variable Cruncher by Alan and Valerie Floeter	77
LIFE by Raul Cartaya	82
Initia by Philippe Francois	85
Apple Checker by Ken McCandless	86
GO — A Greeting Program by Chris Glenn	88
Quick Sort by Cornelis Bongers	90
A.L.E. Applesoft Line Editor by Joe Brooks	95
Othello by Doug Hennig	100
SPRINT Speed Reading by Ken & Michelle McCandless	103
Disk Dump by David Krathwohl	108
Hi-Res Colors by Rob Smythe	109
Oiler by Glen Bredon	109
A.R.C. Apple Record Command System by Chuck Kiss	111
Apple Art Gallery by Ed Young	119
Game I/O Experiment by Gary Little	124
Hi-Res Dump 2X by Robert Delaney	126
Super Numeric Keypad by Alan Cohen	128
Disk Commander by Preston Black, M.D.	131
Lower Case I/O by Ben Colley	136
Compare Applesoft by Chuck Boody	137
Binary Clock by Rip Toren	139
Amp-L-Soft Utilities by Paul Irwin	141
Electronic Message Center by Rudy Guy	154
The Shape by Bill Fortenberry	156
Disk Map by James Church	159
Apple Bowl Football by Preston Black M.D. and Richard Morris	161
Lower Case Letter by James Totten	166
Apple Scroller by Bill Fortenberry	168
Recipe Box by Jan Costenbader	171
Disk Doctor by Ben Colley	177
Quasar II by Brent Iverson	181
DOS Command Entry Points by Mark Chatow	186

EDITOR/PUBLISHER

Mike Harvey

ILLUSTRATOR/DESIGNER

Len Boylan

CONTROLLER

Adele Harvey

ADVERTISING MANAGER

Jerry Merrifield

P.O. Box 639

Lincoln, MA 01773

(617) 259-9710

ADVERTISING ASSISTANT

Mary M. Waddick

MANAGER DEALER SALES

Tammy J. Arico

TECHNICAL EDITOR

David P. Szetela

SHIPPING & LOGISTICS

David M. Sears

ADMIN. ASSISTANT

Kelley M. Magiera

ACCOUNTING

Gladys Hillman

ADMIN. SECRETARY

Johanna Van Doren

Key Perfect 4.0 and Apple Checker 3.0 Tables

Key Perfect and Apple Checker tables are presented in the text of the major articles to help you verify the entry of Nibble programs. Using Key Perfect or Apple Checker after you have keyed in one of our programs should result in tables identical to the ones shown. If they do not match, then you have made a typing error and should closely examine your keyed-in program. Key Perfect 4.0, which points out the portion of your program that contains the typing error, is available from MicroSparc Systems Division — see the ad elsewhere in this issue. If you purchased Key Perfect prior to 6/01/82, then you may receive a free update to version 4.0 by returning your disk to MicroSparc. The tables were created using the updated version.

The Apple Checker tables were generated using Apple Checker 3.0, presented in Volume 3, No. 4. NOTE: All Binary Files must be BLOADED at \$B00 (e.g. BLOAD QUICKSORT, A\$B00) before using Apple Checker 3.0

ED. NOTE

In using Apple Checker and Key Perfect, it is important to be alert to several kinds of Assembly Language Statements which:

1. Reserve memory locations for use by a published program ... but ...
2. Do not physically put data into memory.

Examples of such statements are: **DS** and **DFS**. Here, for example, is how a program might look with a series of DFS (or DS) statements using the Nibble M.L.E. program (Vol. 3 #2 page 21):

436B	60	459		RTS
436C		460	*	
436C		461	* VAR	
436C		462	ENDADR	DFS \$2
436E		463	IMADD	DFS \$2

Notice the **Hex \$60** is stored in memory location **\$436B**. This increases the memory counter to hex **\$436C**.

On **line 462**, there is a **DFS** instruction which reserves a 2-byte memory location and gives it the label **ENDADR**. This increases the memory counter by two positions to **\$436E**, but does not put any data into memory locations **\$436C** and **\$436D**.

The same thing happens on **line 463** where another 2-byte memory location is reserved and labelled **IMADD**. Again, the memory counter is increased by two positions, but **no data is put into memory** at this point.

If no data is physically forced into memory, the Key Perfect or Checker Tables will be generated with whatever happened to be in memory and there will be inconsistent results.

****Important**** Nibble will force zero entries into **DS** and **DFS** reserved memory locations prior to preparing Key Perfect and Checker Tables in order to assure consistent check-sums.

In order to assure compatibility with the Checksums, you should plan to make a note of the occurrences of **DS** and **DFS** statements. Then you can force zero entries into those locations by entering the Monitor (by

typing **CALL-151**) and directly entering zeros into those locations. In the example above, you would:

Type **CALL -151** (to enter the Monitor)

Type **436C:00 00 00 00** (to fill the reserved areas with zeros)

Then you can **BSAVE** the program and then successfully run Key Perfect or Apple Checker against it for a consistent and accurate comparison with the published tables.

Alternatively, with **Key Perfect only**, you can simply ignore the block of memory which covers the **DFS** or **DS** statements (since their contents will not affect the operation of a program), and concentrate on the comparison of those memory locations which contain actual program code.



How to Enter Assembly/ Machine Language Programs in Nibble

To use Machine Language programs in this issue, first enter the Apple Monitor by typing **CALL -151**. At this point an asterisk (*) should appear on your screen.

Now, when you look at an Assembler Listing in the magazine, remember first that the Apple Disassembler does not produce assembly listings such as those shown in the magazine.

Typically, the assembler listings used in Nibble are produced by commercially available Assembler programs. In these listings, the first numbers/characters on the left margin are the memory locations of each machine language statement. An example is: **0300:A9 FF**. In this example, the **0300** is the memory location of the machine language statement.

The second grouping of numbers/characters is the machine code for each statement in memory. In this example:

```
0300-A9 01
0302-8D 00 0A
0305-AD 00 0A
0308-20 A8 FC
030B-20 E4 FB
030E-4C 05 03
```

The **A9 01 8D 00 0A** etc. is the actual machine code.

These two groupings may be separated by either a colon (:) or a dash (-) in the listing. It is very important, in typing in the machine code, to do three things:

1. Separate the Memory Location from the first pair of Machine Code characters with a colon (:), regardless of the symbol used in the assembly listing.
2. Immediately follow the colon with the first pair of Machine Code characters without an intervening space.

3. Then type a space and the second pair of characters, a space and then the third pair of characters and so on.

You can use this method to type in up to **85 consecutive pairs** of machine code instructions without having to type in a new memory location address. Using the example above, you could type in:

```
0300:A9 01 8D 00 0A AD 00 0A <Return>
0308:20 A8 FC 20 E4 FB <Return>
030E:4C 05 03 <Return>
```

When you approach a large number (**LESS THAN 85**) of pairs of machine code characters, you can press **RETURN** to store them in memory. Then begin again with the **memory location address** which follows the last pair of machine code characters you typed in ... followed by a colon ... and continue entering the Hex code until the program is completely entered into memory.

Once the program has been entered, type **CTRL C** to re-enter Basic. Then save the program on disk by typing in a **BSAVE** command which is appropriate to the published program. In the example, the command is:

```
BSAVE BUZZ, A$300, L$11 <Return>
```

You will be given the values which follow the **A\$** and the **L\$** (starting Memory Address and program Length) in each article which contains an assembly/machine language program.

Then you can run it by typing the following command:

```
BRUN BUZZ <Return>
```

To summarize, you can enter the entire machine language program by typing in the first two sets of characters from an assembler listing. They are:

Memory Location

followed by

: (colon)

followed by

Pairs of Machine Language characters

Finally, you can re-enter Basic by typing **CTRL C** and then use the **BSAVE** instruction to save the program. For more information on entering machine language instructions into memory, see the Apple Reference Manual, page 44.

P.S. You might try exercising the example above. It is a small machine language program for sounding your Apple's buzzer. You can run it by saving it to disk and then typing **BRUN BUZZ** as shown above.





The Pain of Entering Machine Language Programs

Let's face it...typing machine language programs from magazines and books is a pain! And if you make an error or leave something out, you may have to go back and retype the whole thing!

The NIBBLE MACHINE LANGUAGE EDITOR (MLE) uses word processing techniques to make the TYPING much easier. Automatic formatting and carriage returns do the job!

If you make a mistake and leave out some code, MLE lets you go back and INSERT the missing bytes. The rest of the program is automatically opened up to make room!

Or if you mistakenly insert extra code, MLE lets you DELETE it, and the rest of the program is moved to fill in the space — automatically.

To EDIT mistyped bytes all you do is move the cursor over the incorrect characters and retype them!

Here's what some new MLE users say!

"MLE is one of those rare utilities that makes you wonder how you ever managed to get along without it."

"I can enter programs I never would have attempted before!"

So, if you want to kill the pain of typing and editing machine language, MLE is for you.

MLE is available for \$29.95 plus \$1.50 shipping/handling (\$2.50 Outside the U.S.) from NIBBLE.

nibble

PO Box 325,
Lincoln, MA 01773
Telephone (617) 259-9710

APPLE CHECKER 3.0 TABLE

PROGRAM NAME:	FILE TYPE:	LENGTH:	CHECKSUM:
AMPERFIND	B	00C2	B3
AMPERSPEED	B	02F8	5D
AMP-L-SOFT	B	06FE	90
APPLESOFT COMPARE	A	087F	DE
APPLESOFT LINE CRUNCHER	B	03E8	C5
APPLESOFT VARIABLE CRUNCHER	B	0380	3F
APPLE ART GALLERY	A	292C	EE
APPLE BOWL FOOTBALL	A	4182	33
APPLE CAL	A	2828	AC
APPLE CHECKER 3.0	B	0188	C1
APPLE DARTS	A	0F4F	F6
APPLE SLUGGER	A	13E3	E7
APPLE SPRINT	A	24B9	05
A.R.C.	A	35B7	25
BOND MANAGER	A	25AF	B5
DB SHAPES A*9400/L185	B	00B9	B6
DISK COMMANDER	B	04AC	18
DISK DOCTOR	A	182F	41
DOS 3+2	B	0351	2E
ELECTRONIC MESSAGE CENTER	A	15F4	C8
GO	A	0F55	B1
HIRES DUMP 2X	B	010E	2B
INCOME TRAC	A	3788	F1
INITIALIZE	A	0BD0	AA
LIFE	A	14E1	60
MERGE FILES	A	163A	10
MICRO-CALC	A	21B2	F6
MICRO-CALC. II	A	1E0A	B0
MLE.B	B	09DC	53
MLE.DRIVER	A	029A	4A
OTHELLO	A	0D8F	2E
QUASAR.BASIC	A	05A5	D0
QUASAR.OBJ	B	04F5	48
QUICKSORT	B	0242	2E
RECIPE BOX	A	273A	84
SCRATCH-PAD	A	1FA1	78
TURTLE GRAPHICS	A	1294	B0

Key Perfect Tables

KEY PERFECT RUN ON AMPERFIND		
CODE	ADDR#	ADDR#
1E5D	4000	404F
2423	4050	409F
0CE4	40A0	40C1
TOTAL PROGRAM CHECK IS : C2		

KEY PERFECT 4.0 RUN ON AMPERSPEED		
CODE	ADDR#	ADDR#
2785	9200	924F
2211	9250	929F
2710	92A0	92EF
2856	92F0	933F
27C5	9340	938F
2667	9390	93DF
2957	93E0	942F
26BA	9430	947F
280C	9480	94CF
11A7	94D0	94F7
TOTAL PROGRAM CHECK IS : 02F8		

continued on next page

Key Perfect Tables (Cont.)

KEY PERFECT 4.0 RUN ON AMP-L-SOFT		
CODE	ADDR#	ADDR#
26EB	4750	479F
2A85	47A0	47EF
22A9	47F0	483F
2AEF	4840	48BF
2858	4890	48DF
2B00	48E0	492F
2B26	4930	497F
2DA4	4980	49CF
2931	49D0	4A1F
2A06	4A20	4A6F
256A	4A70	4ABF
278E	4AC0	4B0F
26DC	4B10	4B5F
2BB4	4B60	4BAF
287B	4BB0	4BFF
2633	4C00	4C4F
26FD	4C50	4C9F
29E5	4CA0	4CEF
254A	4CF0	4D3F
00	4D40	4D8F
00	4D90	4DDF
00	4DE0	4E2F
00	4E30	4E4D
TOTAL PROGRAM CHECK IS : 06FE		

KEY PERFECT 4.0 RUN ON APPLE ART GALLERY		
CODE	LINE#	LINE#
72CC	2	40
8701	50	130
97B5	140	230
6793	240	320
5D9E	400	490
81FC	500	590
876E	600	690
87B6	700	790
8002	800	890
6DA2	900	980
9673	990	1080
911F	1090	1180
7ECB	1190	1280
9099	1290	1380
65EC	1390	1480
52DA	1490	1580
5354	1590	1680
800E	1690	1780
8A5F	1790	1880
49CF	1890	1980
69C2	1990	2080
3FD8	2090	2180
45DA	2190	2280
4522	2290	2380
7672	2390	2480
3F35	2490	2580
4123	2590	2680
4B6D	2690	2780
2C43	2790	2880
3CB0	2890	2980
80C0	2990	3080
2D37	3090	3180
31C0	3190	3280
7964	3290	3380
5DB7	3390	3480
597E	3490	3580
7F97	3590	3680
6C2D	3690	3780
46CE	3790	3880
4E2D	3890	3980
4547	3990	4080
51C2	4090	4180
4710	4190	4280
44FC	4290	4380
2FC7	4390	4480
4F0F	4490	4580
46F0	4590	4680
4EA6	4690	4780
738A	4790	4880
A39B	4890	4980
658A	4990	5080
5822	5090	5180
5107	5190	5280
74F7	5290	5380
2EAE	5390	5480
99D7	5490	5580
3ESC	5590	5630
TOTAL PROGRAM CHECK IS : 2FFF		

KEY PERFECT 4.0 RUN ON APPLESOFT LINE CRUNCHER		
CODE	ADDR#	ADDR#
277D	9200	924F
2D24	9250	929F
2A79	92A0	92EF
257C	92F0	933F
259B	9340	938F
2619	9390	93DF
2904	93E0	942F
2942	9430	947F
2363	9480	94CF
2908	94D0	951F
28B7	9520	956F
26F4	9570	95BF
13A5	95C0	95E7
TOTAL PROGRAM CHECK IS : 03EB		

KEY PERFECT 4.0 RUN ON APPLE CAL		
CODE	LINE#	LINE#
786B	1	30
87F9	40	150
443B	160	550
6FEF	560	1020
6DCC	1030	1120
72B7	1130	1220
4603	1230	1290
510C	1300	1390
674B	1400	1485
5010	1490	1550
6528	1560	1650
71B3	1660	1750
3C8D	1760	1850
3CD9	1860	1950
4954	1960	2030
70E1	2035	2120
1FEA	2130	2220
8A7A	2230	2320
5566	2330	2430
39C1	2440	2540
766F	2545	3020
7387	3030	3110
926E	3120	3210
7C4B	3220	3300
63D5	3310	4040
72CF	4060	4140
9139	4150	4240
72BD	4250	4350
77CE	4360	4440
0160BD	4450	4570
01EB7A	4580	4670
E258	4680	5060
6874	5070	5160
5184	5170	5255
45C5	5260	5350
3E19	5360	6015
5802	6020	6110
4195	6120	7020
51DB	7030	7120
36C0	7130	7240
2E26	7250	10005
8096	10010	11030
581A	11040	11135
5008	11140	11210
656F	11215	11290
46F9	11300	20055
494D	20060	20140
3A6B	20150	20230
4577	20240	20330
4A20	20340	20440
01B0	20450	20450
TOTAL PROGRAM CHECK IS : 2DDA		

KEY PERFECT 4.0 RUN ON APPLE CHECKER		
CODE	ADDR#	ADDR#
25C2	0803	0852
2998	0853	08A2
2A9B	08A3	08F2
2294	08F3	0942
27E9	0943	0992
2960	0993	09E2
264A	09E3	0A32
0F65	0A33	0A59
TOTAL PROGRAM CHECK IS : 0257		

KEY PERFECT 4.0 RUN ON APPLE BOWL FOOTBALL		
CODE	LINE#	LINE#
5898	10	100
015872	110	200
5395	210	300
A4B3	310	400
72C5	410	500
BB32	510	600
542A	610	700
4C77	710	800
9893	810	900
5AFE	910	1000
7928	1010	1100
9749	1110	1200
C37F	1210	1300
63C7	1310	1400
AF23	1410	1500
E79C	1510	1600
85FB	1610	1700
80AD	1710	1800
937D	1810	1900
8874	1910	2000
6DAD	2010	2100
7932	2110	2200
AEC3	2210	2300
A991	2310	2400
49B5	2410	2500
B11E	2510	2600
B6D7	2610	2700
E0B1	2710	2800
BC37	2810	2900
7BE3	2910	3000
4E4B	3010	3100
9FAD	3110	3200
ABBB	3210	3300
7053	3310	3400
D14E	3410	3500
D72E	3510	3600
B2EA	3610	3700
633F	3710	3800
E041	3810	3900
92A7	3910	4000
C38F	4010	4100
9180	4110	4200
E9CB	4210	4300
F165	4310	4400
C196	4410	4500
EC7F	4510	4600
B38E	4610	4700
B9FB	4710	4800
016DDD	4810	4900
014C11	4910	5000
E1ED	5010	5100
B93D	5110	5160
TOTAL PROGRAM CHECK IS : 47BB		

KEY PERFECT 4.0 RUN ON APPLE DARTS		
CODE	LINE#	LINE#
7BD3	1	102
665E	105	245
A735	250	320
725C	350	430
C794	440	1010
7189	1015	1060
962D	1065	1145
E327	1200	10039
01B3C6	10040	15000
016D61	15010	16010
TOTAL PROGRAM CHECK IS : 105A		

KEY PERFECT 4.0 RUN ON APPLE SLUGGER		
CODE	LINE#	LINE#
E85D	100	1090
AB4C	1120	7005
A33E	7010	7300
7764	7310	7347
7009	7348	7386
70AF	7390	7550
E830	7560	8040
DC91	8043	8235
0103CB	8240	8421
DAC2	8422	8442
57DB	8443	8517
7FEC	8518	8625
0104CC	8630	8845
9760	8850	9010
95AB	9040	9210
TOTAL PROGRAM CHECK IS : 1572		

KEY PERFECT 4.0

RUN ON
APPLE SPRINT

CODE	LINE# - LINE#
6827	50 - 100
F15B	150 - 600
6D7A	650 - 1100
8066	1150 - 1500
5856	1550 - 2000
671C	2050 - 2500
449C	2550 - 3000
69CE	3050 - 3500
872B	3550 - 4000
59B1	4050 - 4500
8E0E	4550 - 5000
42AB	5050 - 5500
4B85	5550 - 6050
4AEA	6100 - 6550
5DAE	6600 - 7050
A36C	7100 - 7550
0127AB	7600 - 8050
476A	8100 - 8550
493B	8600 - 9100
6DAA	9125 - 9500
4085	9550 - 10000
575A	10100 - 10550
58B2	10600 - 11050
5C5C	11100 - 11550
4AA9	11650 - 12150
800F	12200 - 12700
8944	12750 - 13000
4FED	13090 - 13500
8866	13550 - 13950
6B25	14000 - 14450
7D0B	14500 - 14950
85E4	15000 - 15450
9650	15500 - 15950
D3DF	16000 - 16450
01E726	16500 - 16950
01481C	17000 - 17450

TOTAL PROGRAM CHECK IS : 2867

KEY PERFECT 4.0

RUN ON
A.L.E. UPDATE 6.24.82

CODE	ADDR# - ADDR#
282B	9010 - 905F
2A5F	9060 - 90AF
275E	90B0 - 90FF
28A9	9100 - 914F
2657	9150 - 919F
2408	91A0 - 91EF
297C	91F0 - 923F
2A54	9240 - 928F
2A95	9290 - 92DF
284A	92E0 - 932F
2343	9330 - 937F
282A	9380 - 93CF
1792	93D0 - 93FE

TOTAL PROGRAM CHECK IS : 03EF

KEY PERFECT 4.0

RUN ON
DISK COMMANDER

CODE	ADDR# - ADDR#
28FC	9200 - 924F
2A96	9250 - 929F
28D4	92A0 - 92EF
27C3	92F0 - 933F
28E9	9340 - 938F
2567	9390 - 93DF
24A5	93E0 - 942F
24D4	9430 - 947F
255E	9480 - 94CF
2BE3	94D0 - 951F
28C6	9520 - 956F
2BB4	9570 - 95BF
05F4	95C0 - 960F
2B5C	9610 - 965F
23FF	9660 - 96AB

TOTAL PROGRAM CHECK IS : 04AC

KEY PERFECT

RUN ON
DOS 3+2

CODE	ADDR# - ADDR#
0FE9	0B00 - 0B4F
13B9	0B50 - 0B9F
12DB	0BA0 - 0BEF
11A3	0BF0 - 093F
13F7	0940 - 098F
1FBF	0990 - 09DF
2410	09E0 - 0A2F
226E	0A30 - 0A7F
1ED7	0AB0 - 0ACF
307A	0AD0 - 0B1F
19BD	0B20 - 0B50

TOTAL PROGRAM CHECK IS : 0351

KEY PERFECT 4.0

RUN ON
A.R.C.

CODE	LINE# - LINE#
6431	10 - 100
A4C9	110 - 200
6C4F	210 - 300
AB73	310 - 400
58CD	410 - 500
CEB3	510 - 600
772F	610 - 700
010DD2	710 - 800
6FD2	810 - 900
69EE	910 - 1000
5930	1010 - 1100
85BD	1110 - 1200
E5BC	1210 - 1300
BF3D	1310 - 1400
B6AA	1410 - 1500
A6F6	1510 - 1600
899D	1610 - 1700
A7BD	1710 - 1800
8FC2	1810 - 1900
9630	1910 - 2000
7927	2010 - 2100
B516	2110 - 2200
C3A5	2210 - 2300
BGC2	2310 - 2400
62AE	2410 - 2500
7A36	2510 - 2600
9191	2610 - 2700
90EE	2710 - 2800
6ED2	2810 - 2900
8D1E	2910 - 3000
9980	3010 - 3100
B0D4	3110 - 3200
AE60	3210 - 3300
A703	3310 - 3400
D725	3410 - 3500
C983	3510 - 3600
936A	3610 - 3700
A364	3710 - 3800
8F56	3810 - 3900
BABF	3910 - 4000
63DB	4010 - 4100
885F	4110 - 4200
FA87	4210 - 4300
598B	4310 - 4400
3E31	4410 - 4500
72F7	4510 - 4600
754D	4610 - 4700
0E83	4710 - 4730

TOTAL PROGRAM CHECK IS : 3B68

KEY PERFECT 4.0

RUN ON
DISK DOCTOR

CODE	LINE# - LINE#
55A7	10 - 30
5212	40 - 550
930D	560 - 1040
9540	1050 - 1140
6A99	1150 - 1240
630F	1250 - 2010
8014	2015 - 2060
63FE	2065 - 2110
4978	2115 - 2405
A418	2410 - 2455
9630	2460 - 2496
776D	2500 - 2544
4BDB	2545 - 2695
915B	2700 - 2745
5073	2750 - 2820
401B	2825 - 2870
4BC9	2875 - 2995
4512	3000 - 3090
8E97	3100 - 3190
609D	3500 - 4030
68C8	4040 - 4130
5F9A	4140 - 5010
655F	5020 - 10000
520D	10010 - 10100
5335	10110 - 10200
5E54	10210 - 10300
7E32	10310 - 10400
AD99	10410 - 10500
716C	10510 - 10980
521D	10990 - 11080

TOTAL PROGRAM CHECK IS : 1B8B

KEY PERFECT 4.0

RUN ON
BOND MANAGER

CODE	LINE# - LINE#
4E69	1 - 120
3D46	130 - 220
3494	1000 - 1090
29ED	1100 - 1190
2C2E	1200 - 1290
38CC	1300 - 2050
322E	2060 - 2150
29DD	2160 - 3040
97BD	3050 - 3140
675A	3150 - 4060
5458	4070 - 5020
5C0C	5030 - 5120
AE6B	5130 - 5220
5E4E	5230 - 5320
6047	5330 - 6000
51A9	6010 - 6100
AB91	6110 - 6200
6453	6210 - 6300
4A57	6310 - 6400
42EA	6410 - 6500
57B2	6510 - 6600
46B9	6610 - 7000
95FA	7090 - 7180
6309	7190 - 7280
46B2	7290 - 7380
49C1	7390 - 7480
4B42	7490 - 7580
3D25	7590 - 7680
3FFE	7690 - 7780
4925	7790 - 7880
4610	7890 - 7980
4B2F	7990 - 10000
4FB5	10010 - 10100
4D00	10110 - 10200
3471	10210 - 10300
423B	10310 - 10400
72EF	10410 - 10500
8847	10510 - 11020
66CC	11030 - 11120
CD65	11130 - 11220
3B61	11230 - 11320
7146	11330 - 11420
4C68	11430 - 12000
40A3	12010 - 13030
9740	13040 - 13130
76E9	13140 - 13230
4FD6	13240 - 14050
7587	14060 - 14150
56B0	14160 - 14250
3E7F	14260 - 14350
4451	14360 - 15040
67B6	15050 - 15140
5518	15150 - 15240
2EE9	15250 - 15340
3611	16000 - 16130
6ADB	16140 - 16230
2E56	16240 - 17050

TOTAL PROGRAM CHECK IS : 2B97

KEY PERFECT 4.0

RUN ON
EMC

CODE	LINE# - LINE#
87C9	10 - 120
CCB6	130 - 220
97BB	230 - 1070
74F4	1080 - 1170
7499	1180 - 2010
9A69	2020 - 2110
8755	2120 - 3080
0104FB	3090 - 4010
7159	4020 - 4110
61F6	4120 - 4210
AFE1	4220 - 4310
60F5	4320 - 5000
8FD6	5010 - 5100
9373	5110 - 6030
4261	6040 - 6130
4D01	6140 - 8040
6600	8050 - 9060
5BF8	9070 - 10030
86DF	10040 - 10130
A66D	10140 - 15050
83B4	15060 - 32080
37F7	32090 - 32120

TOTAL PROGRAM CHECK IS : 1B4E

continued on next page

Key Perfect Tables (Cont.)

KEY PERFECT 4.0 RUN ON GO		
CODE	LINE# - LINE#	
5234	1 -	40
6CA9	50 -	140
979B	150 -	240
F133	250 -	340
CFB4	350 -	440
75DB	450 -	540
8B21	550 -	640
A6E1	650 -	740
5F7D	750 -	840
82ED	850 -	940
93DC	950 -	1040
836F	1050 -	1140
C4ED	1150 -	1240
39D5	1250 -	1260
TOTAL PROGRAM CHECK IS : 10E0		

KEY PERFECT 4.0 RUN ON HIRES DUMP		
CODE	ADDR# - ADDR#	
266D	9200 -	924F
272E	9250 -	929F
2B24	92A0 -	92EF
0B1A	92F0 -	930D
TOTAL PROGRAM CHECK IS : 010E		

KEY PERFECT 4.0 RUN ON INCOME TRAC		
CODE	LINE# - LINE#	
7CEB	10 -	75
47F0	80 -	110
AB55	120 -	200
C9C3	205 -	340
5EA3	500 -	1070
620F	1080 -	2410
42D5	2420 -	2530
65D6	2535 -	2575
AB1C	2580 -	2920
4D2A	3000 -	3090
ASB3	3100 -	3530
8CA3	3540 -	3620
5324	3630 -	3770
6AC6	3780 -	5002
6B2A	5005 -	5110
891E	5130 -	5220
C2E0	5230 -	5320
74A9	5330 -	5460
6AF7	5470 -	7015
5B3D	7020 -	7102
8B9B	7104 -	7240
A91E	7250 -	7305
54DA	7310 -	7372
6F47	7375 -	7420
8D95	7430 -	9030
945F	9040 -	9075
7D2A	9080 -	9130
7AB4	9135 -	9200
6AD3	9205 -	9255
5F7B	9260 -	9320
51F0	9325 -	9370
8F7B	9380 -	9420
708D	9425 -	9455
96B1	9460 -	9502
8347	9505 -	9551
75E4	9552 -	9582
6B0E	9585 -	9620
947B	9625 -	9675
704D	9680 -	9850
6B7E	9855 -	9890
6B9E	9895 -	9940
7B6B	9945 -	10010
76D6	10020 -	11035
A09C	11100 -	11250
8D3C	11260 -	11350
979C	11360 -	12070
AABE	12080 -	12220
2EAD	12240 -	20000
TOTAL PROGRAM CHECK IS : 3266		

KEY PERFECT 4.0 RUN ON LIFE		
CODE	LINE# - LINE#	
633A	0 -	9
6464	15 -	28
5C27	30 -	180
56BF	190 -	310
544B	330 -	420
5CCB	430 -	1000
47C1	1010 -	1090
496A	1100 -	1190
552B	1200 -	1350
3F3E	1360 -	1445
416B	1448 -	1540
4B7B	1550 -	1720
5A34	1730 -	1800
4C9D	1810 -	3030
4AF9	3040 -	3500
849A	3505 -	4010
77D3	4020 -	4070
44FB	4080 -	4115
82D4	4120 -	4520
4DB1	4530 -	4620
8093	4630 -	5070
8342	5080 -	5170
AE10	5180 -	5270
A62F	5280 -	5370
9DEA	5380 -	5470
A3D9	6000 -	6090
577D	6100 -	6160
TOTAL PROGRAM CHECK IS : 1823		

KEY PERFECT 4.0 RUN ON MICRO-CALC		
CODE	LINE# - LINE#	
6DA2	10 -	25
FE3F	26 -	110
75BB	120 -	210
B75E	220 -	310
A41B	320 -	410
A071	420 -	510
A304	520 -	610
C1D4	620 -	710
AEA6	720 -	810
AE8B	820 -	910
BE81	920 -	1010
AFFB	1020 -	1110
C7DF	1120 -	1210
82C9	1220 -	1310
AC37	1320 -	1410
9BA2	1420 -	1510
010DF5	1520 -	1610
AAC1	1620 -	1710
C171	1720 -	1810
7D83	1820 -	1910
BEF5	1920 -	2010
B195	2020 -	2110
7EA7	2120 -	2210
B63E	2220 -	2310
E7B4	2320 -	2410
B3BC	2420 -	2510
B931	2520 -	2610
1A02	2620 -	2640
TOTAL PROGRAM CHECK IS : 24FB		

KEY PERFECT 4.0 RUN ON MICRO-CALC.II		
CODE	LINE# - LINE#	
71B9	1 -	40
86B2	50 -	140
012C44	150 -	240
018C92	250 -	340
D504	350 -	440
83CB	450 -	540
A2B9	550 -	640
91B7	650 -	740
A3A2	750 -	840
7D0C	850 -	940
87FD	950 -	1032
CFB1	1040 -	1130
94B0	1140 -	1230
C7D1	1240 -	1330
C6B3	1340 -	1430
A22B	1435 -	1520
766E	1530 -	1620
B02B	1630 -	1720
CCFE	1730 -	1800
AA2E	1810 -	1900
A365	1910 -	1990
8655	2000 -	2070
TOTAL PROGRAM CHECK IS : 20A3		

KEY PERFECT RUN ON MLE.B		
CODE	ADDR# - ADDR#	
2AAE	4000 -	404F
29FE	4050 -	409F
2A92	40A0 -	40EF
212F	40F0 -	413F
272D	4140 -	41BF
2764	4190 -	41DF
2337	41E0 -	422F
2207	4230 -	427F
27E3	4280 -	42CF
264B	42D0 -	431F
23BB	4320 -	436F
3B95	4370 -	43BF
38BB	43C0 -	440F
3BE6	4410 -	445F
2D3A	4460 -	44AF
2ABF	44B0 -	44FF
22AC	4500 -	454F
1EA9	4550 -	459F
2B34	45A0 -	45EF
25B3	45F0 -	463F
23E1	4640 -	46BF
275E	4690 -	46DF
2670	46E0 -	472F
237D	4730 -	477F
269B	4780 -	47CF
2650	47D0 -	481F
272B	4820 -	486F
2CA1	4870 -	48BF
2514	48C0 -	490F
262B	4910 -	495F
2691	4960 -	49AF
144B	49B0 -	49DB
TOTAL PROGRAM CHECK IS : 09DC		

KEY PERFECT 4.0 RUN ON MLE.DRIVER		
CODE	LINE# - LINE#	
651E	1 -	15
8535	20 -	85
6D5F	90 -	210
TOTAL PROGRAM CHECK IS : 02FF		

KEY PERFECT 4.0 RUN ON OTHELLO		
CODE	LINE# - LINE#	
9B6B	2 -	50
60BB	90 -	500
84D2	510 -	620
5C0B	630 -	900
5D16	910 -	2020
47A5	2025 -	2220
6774	2230 -	2430
57A9	2440 -	2530
5359	2600 -	3040
692F	3050 -	3510
6233	3520 -	3900
4F07	4000 -	38020
4A5C	38030 -	38990
4B06	39000 -	39120
5D56	39130 -	60020
013930	60030 -	62045
27AB	62050 -	62070
TOTAL PROGRAM CHECK IS : 0F5B		

KEY PERFECT 4.0 RUN ON QUASAR.BASIC		
CODE	LINE# - LINE#	
62B0	0 -	2020
55D2	2030 -	4020
7DD0	4030 -	4120
7456	4130 -	4220
6790	4230 -	5060
C7A9	5070 -	5180
1C7E	5190 -	5210
TOTAL PROGRAM CHECK IS : 066E		

KEY PERFECT 4.0
RUN ON
QUASAR.OBJ

CODE	ADDR# - ADDR#
2A06	4000 - 404F
2962	4050 - 409F
260E	40A0 - 40EF
26B1	40F0 - 413F
298A	4140 - 418F
2A4B	4190 - 41DF
2697	41E0 - 422F
29DA	4230 - 427F
2213	4280 - 42CF
24D1	42D0 - 431F
2B3C	4320 - 436F
2920	4370 - 43BF
2A3A	43C0 - 440F
24C7	4410 - 445F
23AD	4460 - 44AF
2263	44B0 - 44F4
TOTAL PROGRAM CHECK IS : 04F5	

KEY PERFECT 4.0
RUN ON
QUICKSORT

CODE	ADDR# - ADDR#
2229	9200 - 924F
256C	9250 - 929F
257D	92A0 - 92EF
2CF7	92F0 - 933F
2A33	9340 - 938F
27C1	9390 - 93DF
236A	93E0 - 942F
0855	9430 - 9441
TOTAL PROGRAM CHECK IS : 0242	

KEY PERFECT 4.0
RUN ON
RECIPE BOX

CODE	LINE# - LINE#
9F0F	10 - 85
82B1	90 - 114
7B85	116 - 134
5594	136 - 160
824E	162 - 200
62CF	205 - 300
6806	305 - 365
8E5C	370 - 480
3544	500 - 590
5B1F	600 - 618
36DD	620 - 638
4CFD	640 - 666
3A35	668 - 686
5770	688 - 810
63AD	815 - 900
3E10	905 - 955
5B5F	960 - 1020
AF40	1030 - 1120
A96D	1130 - 1220
78D2	1230 - 2000
9FD5	2005 - 2050
BD66	2055 - 2300
A1DF	2310 - 2470
6E1E	2480 - 2570
6453	2580 - 2670
817A	2680 - 2860
5F0E	2870 - 3000
82CC	3010 - 3110
7817	3120 - 3210
8B6C	3220 - 3470
6F93	3480 - 3570
64F9	4000 - 4540
94B4	4550 - 4640
5B9F	4650 - 4740
5FFB	4750 - 6040
90B7	6050 - 6140
8574	6150 - 6240
62EA	6250 - 6570
5950	6580 - 7070
5748	7080 - 7170
8B8A	7180 - 9050
D777	9060 - 9150
B324	9160 - 9250
5DA6	9260 - 9320
TOTAL PROGRAM CHECK IS : 2C4A	

KEY PERFECT 4.0
RUN ON
SCRATCH-PAD

CODE	LINE# - LINE#
77D6	10 - 100
91B2	110 - 200
6EE5	210 - 300
62FC	310 - 400
6CA1	410 - 500
95BF	510 - 600
9735	610 - 700
AB46	710 - 800
93AE	810 - 900
B141	910 - 1000
990D	1010 - 1100
9E73	1110 - 1200
C54F	1210 - 1300
9F12	1310 - 1400
9C9F	1410 - 1500
A6E5	1510 - 1600
5A1B	1610 - 1700
7ED3	1710 - 1800
455D	1810 - 1900
4E74	1910 - 2000
7CF6	2010 - 2100
C071	2110 - 2200
A5BD	2210 - 2300
63D4	2310 - 2400
87A5	2410 - 2500
CC0B	2510 - 2600
BB65	2610 - 2700
A3A7	2710 - 2800
7381	2810 - 2890
TOTAL PROGRAM CHECK IS : 2313	

KEY PERFECT 4.0
RUN ON
TURTLE GRAPHICS

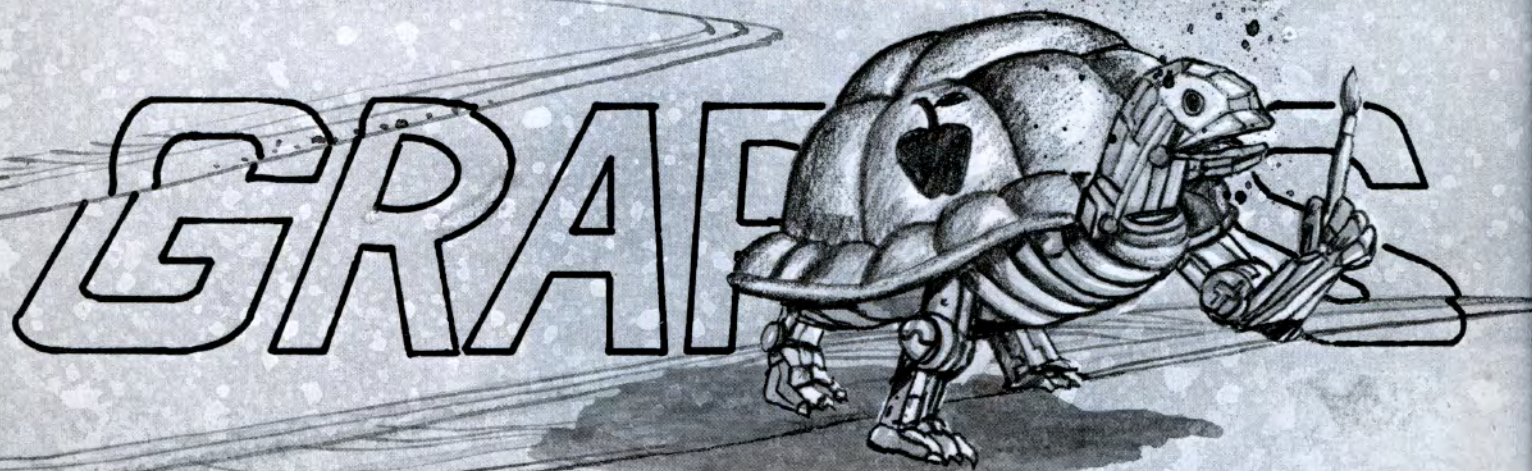
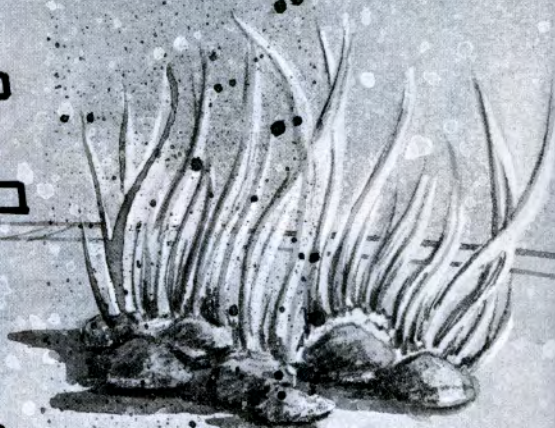
CODE	LINE# - LINE#
85EA	20 - 110
705D	120 - 210
695E	220 - 300
5D30	310 - 450
599C	460 - 560
85C9	570 - 700
4B72	710 - 910
8474	920 - 1010
74AB	1020 - 1180
68B0	1190 - 1340
5344	1350 - 1440
6EBB	1445 - 1520
61CE	1530 - 1699
4624	1700 - 1860
6F7A	1870 - 1960
A84A	1970 - 2070
42B3	2080 - 2170
7027	2180 - 2270
4888	2280 - 2370
73DD	2380 - 2470
339B	2480 - 2570
712A	2580 - 2700
BFF0	2799 - 2875
0FBA	2880 - 2890
TOTAL PROGRAM CHECK IS : 155F	

KEY PERFECT 4.0
RUN ON
THE SHAPE

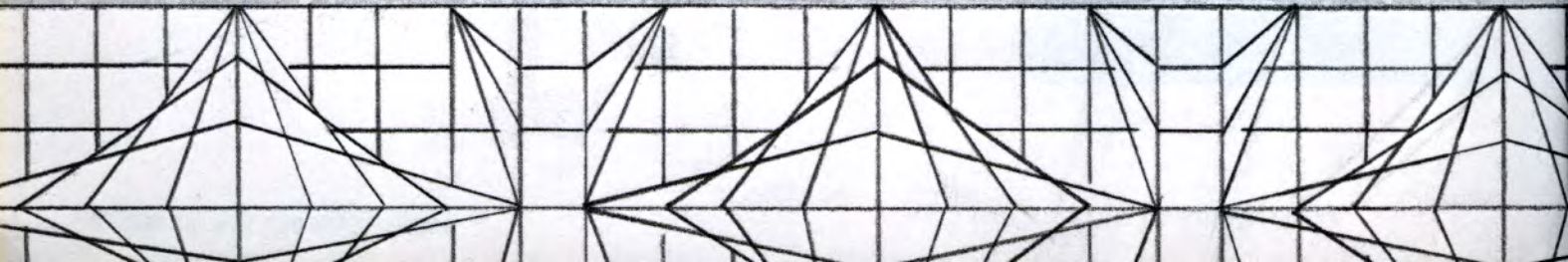
CODE	LINE# - LINE#
74EC	10 - 100
80B8	110 - 200
5DC7	210 - 300
523B	310 - 400
60F8	410 - 500
3F97	510 - 600
39EE	610 - 700
447A	710 - 800
3984	810 - 900
3DA3	910 - 1000
6223	1010 - 1100
4AF1	1110 - 1200
4A38	1210 - 1300
5027	1310 - 1400
686C	1410 - 1500
53DE	1510 - 1600
3F00	1610 - 1700
4792	1710 - 1800
92C5	1810 - 1900
4105	1910 - 2000
5987	2010 - 2100
5E5B	2110 - 2200
6E1F	2210 - 2300
3164	2310 - 2400
90C7	2410 - 2500
329A	2510 - 2600
7A55	2610 - 2700
BCAD	2710 - 2800
62C4	2810 - 2900
7BE9	2910 - 3000
F6BC	3010 - 3100
0172FC	3110 - 8275
0CD6	21317 - 21317
TOTAL PROGRAM CHECK IS : 1CD0	



APPLE TURTLE



Boylan



APPLE TURTLE GRAPHICS

Apple Turtle Graphics

by David A. Krathwohl
19 Tri St.
Ashland, MA 01721

TURTLE GRAPHICS is a hi-res graphics drawing program in which a "turtle" traces a drawing **under control of written instructions**. The instruction language though simple to learn, encompasses several levels of complexity, from simple line segments to curves, from simple repeating functions to recursion. Turtle instructions may be entered from the keyboard in a series as an instruction set. Picture names may be assigned to a specific instruction set and then used to call the picture. Variables may be used, and picture instruction sets may be edited. Picture names and instruction sets may then be saved to disk as a text file for further use and development.

Depending on your orientation, TURTLE GRAPHICS can either be an excellent tool to learn simple programming concepts, a fun way of working with math and learning geometry, or just an enjoyable "picture creator". For the programmer, it can also be an **exploration of computer language creation** on a small scale and the manipulation of the hi-res graphics mode of the Apple. Before examining these possibilities, it would help to better understand what TURTLE GRAPHICS does and how it is used.

THE TURTLE

The "turtle" in TURTLE GRAPHICS is a set of five points on the hi-res graphics screen which indicate a position and a direction. When the program is run, you are first presented with the turtle and a question, "**What do you want the turtle to do?**" In order to respond to this question, **you must use a language with which the turtle is familiar**. The four basic commands in turtle-ese are **FD** for forward, **BK** for back, **RT** for right, and **LT** for left. Each two letter command precedes either a positive or negative quantity. For example, **FD40** moves the turtle forward 40 units. (Each unit corresponds to a point on the hi-res screen.) A command of **RT90** would turn the turtle 90° to the right.

Most simply, then, you could enter a series of these commands, each followed by "return" to move the turtle as you wish. Fortunately there are easier ways. Turtle-ese has several levels of complexity, each of which requires its own symbols and syntax.

COMMAND SERIES

First, when the turtle asks for instructions, a series of commands can be entered on the same line as long as they are separated by periods. For example, **FD20.RT90.FD20** will move the turtle forward 20 units, turn it right 90° and then move it forward another 20 units before again requesting instructions.

Before discussing the next level of complexity, there are **three other commands** which will help your drawing. The first is **CL** for clear. This command clears the screen and sends the turtle back to the center, pointed toward the top. This is the position of the turtle when the program is first run.

TURTLE PEN

The other two commands are **PU** for penup and **PD** for pendown. These commands switch the "ink supply" on or off. When the pen is up, the turtle still moves and turns, but does not draw. PD causes the "ink" to flow again. These commands can be used, just like the four movement commands, in a series separated by periods. For example, **FD30.PU.FD10.PD.FD20** would draw a segment of 30 units, skip over 10, and draw another segment of 20 units.

By now you could probably deduce that to draw a square with a side of 20 units, the instructions would be **FD20.RT90.FD20.RT90.FD20.RT90.FD20**. But that is a lot of typing for just a square. The next level of turtle-ese has a repeat command that lets you tell the turtle to repeatedly DO something a certain number of times. The above square, for instance, becomes **DO4(FD20.RT90)**, which translates to "do four times the instructions move forward 20 units then turn right 90°". Again, the DO command can be used serially with other commands such as **FD30.DO4(FD20.RT90)**. **RT90**.

The one thing you cannot do is to put a DO inside another DO, such as **DO2(DO4(FD20.RT90))**. But what if you want to draw two squares, one right after the other? The solution is in yet another level of turtle-ese.

TURTLE PICTURES

The turtle recognizes any word preceded by a > as a picture name. The first time you instruct it to draw a particular picture, the turtle asks for the specific directions. If you tell the turtle to **>SQUARE**, it will first ask, "How". If you then tell it to **DO4(FD20.RT90)** it will now know how to **>SQUARE**, and these instructions will be retained and re-used as long as the program is running (and has not been modified by a file command that will be explained later). Again, the picture command can be used serially with other commands. For example, **>SQUARE.RT45**, will first draw a square and then turn the turtle 45°. If you want to draw two squares, you can now write **DO2(>SQUARE.RT45)**. (The RT45 turns the turtle so it won't just trace over the same square twice.)

INVISIBLE TURTLE

Before examining the next level of turtle-ese, there are three other commands that will make life easier. The first two of these allow you to determine if the turtle is actually drawn on the screen. The command **NT** for no-turtle, removes the turtle and the command **TU** for turtle, brings it back. The primary advantage in this is the speed with which the turtle can execute your instructions. If the program has to draw and erase the turtle each time it moves, it takes more time to draw your shapes.

COMMAND EDITING

The third command needed is **ED** for edit. The edit command has to be followed by the name of a picture in memory. For example, if you wanted to edit **>SQUARE** to give it a side of 30 instead of 20, you would type **ED >SQUARE**. This puts you into the picture editor mode where you are presented with the name of the picture being edited and the

instruction set for the picture, printed twice. The cursor sits at the beginning of the second instruction set.

For minor changes you can simply move the cursor over the instructions by using the forward arrow (re-type) key until you reach a character you want to change. So, after "forward arrowing" over "**DO4(FD**", you stop with the cursor over the "**2**" and type a "**3**". Then "forward arrow" (using the repeat key, if you wish) to complete the instruction set, and press "return" to return to the graphics mode. Now whenever you tell the turtle to **>SQUARE**, it will draw a square with a side of 30 units.

TURTLE VARIABLES

Now assume that you want to draw a square, but you don't know yet how big you want it. The next level of turtle-ese allows variables to be named. In the **>SQUARE** example, the instructions could be **DO4(FD SIDE.RT90)**. When executing these instructions, the turtle will ask only once for the value of **SIDE**. It will then assign that value any time the variable name **SIDE** is encountered. A total of ten variable names are allowed in any instruction set.

In some pictures, it would be useful to increase or decrease a value each time a variable name is encountered. The turtle-ese for this is **IN** for increment. This command allows the creation of spiral-type shapes and may be used with either positive or negative numbers. (**IN** is most practical when used with only one variable. It will increment all variables at the same rate if others are included.)

CALLING PICTURES

One of the most useful properties of the picture naming command is the ability of one picture to "call" other pictures. For instance, assume you are trying to create a landscape which contains two identical birds. You might first create a picture called **>BIRD**, and then include it twice in a picture called **>LANDSCAPE**. It would also be possible to have several layers of pictures calling others. For instance, **>LANDSCAPE** might call **>BIRD** which in turn might call **>WING**, **>BEAK**, and **>TALON**. This not only saves typing, but also allows your instruction set to effectively exceed the 256 character limitation of Apple string input.

RECURSION

A very special feature of this picture — calling — pictures capability is the possibility of having a picture call itself. This is called recursion, and it allows you to create endless varieties of repetitive images. With the inclusion of the increment command, it allows for spiral-type figures of unlimited structure and design. An example of recursion is **>TRIANGLES**, where the instruction set is: **DO3(FD SIDE.RT120).RT30.>TRIANGLES**. This will draw a series of equilateral triangles, each rotated right 30° from the last one.

(If you actually try this instruction set, you will also need to know how to stop the drawing without stopping the program. Pressing any key during the execution of a drawing will stop it and return you to a request for further instructions.)

continued on next page

Apple Turtle Graphics (Cont.)

HOME

A particularly useful command when you have pictures calling other pictures is **HO for home**. This moves the turtle to the center of the screen, pointed toward the top, without clearing. It makes a very convenient reference point for starting your separate picture instructions. To effectively use it, your last three instructions for a picture might be **PU.HO.PD**. In this way, the location and orientation of the succeeding picture are not related to the end position of the last picture, and changes can be more easily made in either one.

TURTLE PICTURE FILING

After you have spent considerable time in creating your pictures, it would be a shame to lose your creations when you turn off your Apple. Fortunately the turtle understands the command **FI for file**. Using this command gives you a menu of five filing options: **1) Save pictures, 2) Retrieve previously saved pictures, 3) Delete a file, 4) Catalog and index files, or 5) Return to the graphics mode.**

The first option allows you to create a file name and store the pictures you have created on a disk. Saving pictures to disk does not remove them from the current memory space of the program. If you return to graphics after saving a picture file, you will still be able to call the "saved" picture names.

The second option allows you to reload previously saved pictures into the current memory space of the program. By retrieving a picture file, you replace the current pictures with those from the disk. If you are uncertain of the current contents of memory, it is a good idea to save first and ask questions later.

The third option allows you to clear the disk of unwanted files. This option lets you delete inadvertently created duplicate files as well as files which are no longer useful.

The fourth option has two parts. The first part is the cataloging of the disk. Since picture files are saved as text files, the file names preceded by a "T" are your picture files. If you want to find out which pictures are in a particular file, the indexing option lets you step through each one, showing both name and instructions, by simply pressing any key.

The fifth option brings you back to the graphics mode. The screen is cleared, and the turtle is in the "home" position.

Finally, after the day's work is saved to disk and you are in the graphics mode, the command **QU for quit**, returns you to Applesoft in the text mode with a clear screen.

WHY TURTLE GRAPHICS?

The idea of turtle graphics is not a new one. It was developed at MIT and has been extensively discussed in a book called **Mindstorms** by Seymour Papert (Basic Books, N.Y., 1980). Unfortunately, it was not previously available in BASIC for the Apple. I first discovered it in a computer workshop for teachers and became so enthusiastic that I decided to write a program in BASIC for the Apple.

The idea appealed to me for several reasons. First, it is obviously fun, and with the option of saving pictures, it could produce fairly complex drawings.

Second, **turtle graphics creates an easily learned minilanguage that has many of the**

characteristics of high level programming languages. It has a set of commands and symbols, it has a specific syntactic form (and accompanying error messages), and it allows variables, incrementing (IN), loops (DO) and the creation of subroutines (a picture calling a picture). In addition, it involves editing and disk use. It therefore seems an excellent **starting "language"** in which people without computer backgrounds can obtain a conceptual foundation for programming.

Finally, turtle graphics is an ideally opened "game" for the exploration of mathematical concepts, particularly plane geometry. Since the turtle's language is units of length and degrees, discoveries (such as the 180° sum of the internal angles of a triangle) can easily be made through experience rather than merely the abstract medium of a geometry book. Creative play could thus lead to an intuitive grasp of geometry.

By using a picture which first creates a grid, the concept of area could also be explored. And since commands can be given in both negative and positive quantities, a picture which creates a number line could be used to explore the addition and subtraction of negative (as well as positive) numbers.

PROGRAMMING TECHNIQUES

As a computer program, **TURTLE GRAPHICS** consists of a very short main segment (lines 40-280) which controls the calls to 16 subroutines or segments (see table of program segments). After the program has called the set-up subroutine (starting at line 2800), it asks for instructions (IN\$ in line 45). To make life easier, the first two characters of the instructions are given their own variable names (I\$ in line 80) and then checked against a series of conditional statements. If a recognizable command is found, the appropriate series of subroutines is called. Once the command has been carried out, a parsing routine is executed (starting at line 300) which searches for the first period, removes the command preceding it (which has just been executed), and sends the remainder of the instructions back through the main segment for execution.

For example, the instructions **FD40.RT90.NT** are first interpreted in line 100 when the **FD** is "found". This line calls the subroutine which determines the quantity of the command (starting at line 400). In this case the value of the instruction is 40. (It doesn't matter at this point whether this is degrees or linear units.) The next subroutine called is "move turtle" (starting at line 500). This subroutine first calls the "erase turtle" subroutine (starting at line 700), then calculates the new position of the turtle (line 520) and plots a line to it (if the pen is down), then calls the "draw turtle" subroutine (starting at line 800). Finally, line 100 branches to line 300 for parsing. There the **FD40** is eliminated, **RT90.NT** becomes the new instruction set (IN\$) and the main program segment is run again. All three of these program segments are called from the line (100) which "found" the **FD**.

The other commands are treated similarly. The **PD,PU,NT**, and **TU** commands merely **change flag values** and are then parsed out of the action. **DO, IN, HO, ED, CL**, and **FI** (as well as the **>symbol**) all **call subroutines**. However, since **ED, CL**, and **FI** branch to the request for instructions (line 40) instead of to the parsing segment, they are always effec-

tively the final command in a series. If a quantity in a move, turn or increment command evaluates to zero, a variable is assumed, and the subroutine which sets variable values (starting at line 1500) is called.

STRUCTURE AND SPEED

Since a major consideration for this program was its speed and efficiency, the main program segment is as short as possible. The arrangement of the subroutines was determined by their frequency of use. More frequently used subroutines, such as movement and turtle drawing, are near the beginning of the program, while those less frequently used, such as editing and filing, are nearer the end. The subroutine which sets up the screen, dimensions the variables and pokes a machine language subroutine into memory is at the end since it is used only once.

If you look at the code, you will see that most lines are numbered in even tens and the subroutines in even hundreds. This is not only for neatness's sake, but to make it easier to remember the subroutines by their starting numbers. Remarks are contained in the preceding "99" line so no speed is lost by calling a remark line.

SPECIAL FEATURES

Perhaps the most important influence on the design of this program was the orientation to the non-computer oriented user. Since one of its intended uses is in the classroom, the program needs to be essentially **user-proof**. The first consequence of this is the use of English instead of computer jargon in asking questions and responding to the user. For example, instead of "Input instructions?", the program asks, "What do you want the turtle to do?"

The second result of this orientation is the careful attention to boundary checking and error trapping. If the turtle has been told to move past its boundary, the user is simply asked to try a different move, and the turtle is replaced on the screen at the last legal move (line 530).

Error trapping is done in several ways. First are the double checks built into the program. If the instructions go through the main program without branching, the user is not using correct turtle-ese. The message, "I don't understand," followed by the incorrect instructions is displayed before the user is returned to the request for further instructions (line 280). When a picture is created with incorrect instructions, the user is automatically sent to the edit mode to correct the mistake (line 270).

When **numeric responses** are called for in the file option menu, they are accepted as strings and first converted to numbers, then checked for range errors (lines 2030-2040).

Also, in the **filing routines**, the user is asked to confirm his response before deleting or retrieving picture files, since both of these operations have irreversible effects (lines 2190-2230 and 2330-2340). If an attempt is made to save pictures when there are none in memory, the program checks the picture counter (PI) before opening a file (line 2060). This avoids the creation of empty files which would later give "out of data" errors when indexed or retrieved.

If all else fails, and the user does manage to create an error, two separate error traps are set (ONERR GOTO's in lines 50 and 2000). If

the error occurs in the graphics part of the program, the user is told that the turtle doesn't understand and is then returned to a request for instructions.

If the error occurs in the filing section, the error trap first checks the type of error (with a PEEK(222), line 2640). This is necessary because in order to read a file, the Apple first has to open it. If the user tries to retrieve or index a non-existent file, the file is created in the act of searching, and an "out of data" (code 5 error) results when it is read. If the PEEK(222) shows that this has happened, the bogus file is deleted before a message asking the user to check the file name is displayed. (Since trying to access a text file created by a different program might result in a code 5 error, thereby deleting the file, it is suggested that only turtle text files be used on the program disk.) With any other error in the file section, the same message is displayed without deleting the file named, and the user is returned to the file menu.

MINI-LANGUAGE CREATION

Another feature of TURTLE GRAPHICS is its method of creating a mini-language. It does this by employing a specific set of symbols, syntactic rules, and a method of parsing statements.

For the sake of speed and simplicity, the commands consist of two letters (such as FD, FI, and QU) and the symbols are the period, the greater than sign, and parentheses.

The syntax is the order in which instructions must be given. The commands must precede the quantity, a period must follow each command in a series, picture names must be preceded by a ">", and instructions to be repeated must be contained within parentheses.

Since the syntax of commands requires that the two letter designation precede any other information (such as the angle or a variable name) no time-consuming search is needed to locate the start of this information. In order to use a series of commands, however, a search must be made to find the separate commands. Parsing is simply a method of dissecting the input to determine its meaning.

The BASIC MID\$ command is used to determine where one command stops and another begins. Depending on the program segment, a FOR/NEXT loop uses MID\$(IN\$,J,1) to search for either a period, a greater than sign, or parentheses. When the symbol is found, the program branches to a line which reassigns the instruction variables by using a RIGHT\$ or LEFT\$ command and the value (J) of the loop variable. In this way, the instruction set is read and executed one command at a time.

ONE COMMAND AT A TIME

One advantage to the "one command at a time" parsing method is the ability of the picture to call other pictures or themselves. Each picture name (PI\$(n)) calls a separate string variable containing the picture instructions (FI\$(n)). When a picture name is encountered in an instruction set, the picture instructions (plus a period) are concatenated onto the instruction set (line 1170). This means that picture instructions which contain the name of the picture that called them will create an endless string of instructions through concatenation.

TURTLE PROBLEMS AND SOLUTIONS

One of the most difficult challenges in the development of this program was the creation and manipulation of the turtle itself. The turtle started out as a shape table using the DRAW, XDRAW, and ROT commands for control. However, there were several problems. Foremost was the fact that the ROT command cannot be incremented in single degrees. To solve this, the turtle was drawn using trigonometric functions to determine placement (lines 900-950). In both this and the trigonometry employed for movement, the Apple had to be adapted to accept degrees. Since the Apple only accepts trigonometric functions expressed as radians, a conversion factor was used (Degrees X $\pi/2$ = Radians).

But this still left the more annoying problem of the disappearing drawing. Each time a part of the turtle coincided with a part of the drawing, the next move of the turtle erased that part. A partial solution was a turtle made of dots instead of lines. The whole solution came in the form of a borrowed ampersand subroutine that does for hi-res graphics what the SCRNB command does for low-res. (The source of this subroutine is "Apple II Peeks Pokes & Calls" published by A.P. Computer Products, Holtsville, N.Y.)

The subroutine is poked into memory (lines 2830-2880) from the initial set up subroutine (starting at line 2800) and is called into use during the turtle's creation (lines 960-1010). A new variable is created for each point of the turtle and evaluated by the ampersand routine. The syntax is &C=X,Y. If the value of C returned is one, the point X,Y is lit. Its color is disregarded. If the value is zero, the point is not lit.

For the turtle, this means checking five points. If the point is lit, the Y coordinate of that point is changed so that it is plotted in the text portion (bottom four lines) of the screen. Since the erase routine uses this new Y value to erase a "non-existent" point, the drawing is unaffected.

OTHER TRICKS

The program has other minor features which are worth mentioning. First is the LOMEM command in line 20. The Apple stores variables beginning at LOMEM (which is usually the end of the program) and builds upwards. Since this program occupies the memory space right up to hi-res graphics page one, any variables stored there would be seen as white dots moving across the screen and disrupting your drawing. The command, LOMEM 16400, sets the beginning of the variable space above the hi-res screen.

Another feature is the use of the right arrow (re-type) key for editing. By printing the picture instructions twice, the second time VTAB-bing to a specific line, the location of the line is known (lines 1950-1960). To get the cursor to fall on the first character of this line, a VTAB to the same line precedes the input statement for the corrected instruction set. (Notice the HTAB 2 before printing the instructions to account for the "?" printed by the input statement.)

A third feature is the periodic clearing of old string variable space. Since the Apple stores the old values of strings when they are reassigned, memory slowly fills up with old strings as new instructions are given. This is particularly true if several picture files are

indexed, as each new file reassigns values to the same temporary indexing variables. To clear out this space, the program keeps track of the number of times instructions are requested. When this number reaches 25, the statement FC = FRE(1) is executed (line 40). This clears memory of strings which are no longer being used. You will notice this happening when there is a short pause before the turtle requests new instructions.

PROGRAM MODIFICATIONS

There are many possibilities for customizing TURTLE GRAPHICS. One would be to include a color command which would let you change the colors of the lines being plotted. Another would be to modify the IN command so that only a specified variable would be incremented.

For math exploration, the ability to save and retrieve the entire hi-res screen would be useful. This would allow for a much quicker creation of background graphics such as a grid or numberline.

The difficulty with modifying TURTLE GRAPHICS is that it presently occupies almost all of the available space below hi-res graphics page one. Any serious modification will require either shortening other parts of the program, bridging the hi-res screen, or relocating the entire program above it.

A simple "modification" which would be helpful is to create a "turnkey" system. Such a system would be most helpful for non-computer oriented users. Using an entire disk exclusively for TURTLE GRAPHICS with the following HELLO program will do the trick nicely.

```
10 HOME
20 VTAB 10: HTAB 16: PRINT "LOADING":
  PRINT:INVERSE : HTAB 12: PRINT "
  ": HTAB 12: PRINT
  "TURTLE GRAPHICS": HTAB 12: PRINT
  ": NORMAL: PRINT: HTAB 13:
  PRINT "PLEASE WAIT..."
30 DS = " ": REM CONTROL D BETWEEN " "
40 PRINT DS;"RUN TURTLE GRAPHICS"
50 END
(The spaces in line 20 are 15 characters long.)
```

Now all you have to do is load the disk, turn on the machine, and be ready to talk turtle-ese.

TURTLE GRAPHICS SYNTAX AND USE

1. Pressing any key during the execution of a drawing will stop the execution and return the user to a request for instructions.
2. The commands CL, FI, and QU should not be followed by other commands as the extra commands will be ignored.
3. Spaces between a command and a variable name are not significant. (They are treated as a part of the variable name.) However, both spaces and control characters are significant if included in picture names.
4. Two slash marks (//) are used internally as a flag for the DO function. An error message with these symbols at the end indicates an error in the DO instructions.
5. The DO function may be repeated serially, but may not be used inside another DO

continued on next page

Apple Turtle Graphics (Cont.)

command. However, a picture name which uses the DO command in its instruction set may be used inside another DO command.

6. A quantity of zero in the DO command will result in endless repetition as the DO counter increments toward the Apple's negative integer limit.
7. If the first character (other than a space) after a FD, BK, RT, LT, or IN command is non-numeric or zero, the program assumes a variable has been named. When a variable is encountered, the program will ask for its value. It will retain and use this value for the remainder of the instruction set.

800	Draw turtle
900	Determine turtle location (hi-res screen test)
1100	Check for picture
1300	Repeat function (DO)
1500	Variable determination
1600	Home
1700	Clear
1800	Picture editor
2000	File options and filing
2700	Time delay for message display
2800	Set up variables and screen

X2	Positions of turtle dots
X3	
X4	
X5	
X6	
Y2	
Y3	Hi-res screen test for turtle dots
Y4	
Y5	
Y6	
BT	
CT	
DT	Turn — amount in degrees Quantity of move or angle Conversion factor — radians to degrees
ET	
FT	
TU	
QU	
CV	
NI	Conversion factor for turtle location
T	Turtle/Noturtle flag
DR	Penup/Pendown flag
B	Backward movement flag
LE	Left turn flag
Picture Variables	
PI	Picture counter
BP	Bad picture instructions flag
PI\$(n)	Picture names
FI\$(n)	Picture instructions
File System Variables	
NU	Temporary picture counter
PX\$(n)	Temporary picture name
FX\$(n)	Temporary picture instructions
OP\$	File option response
OP	Value of OP\$
NA\$	Name of file
K\$	Miscellaneous keyboard responses
D\$	Control D for DOS commands

Editor Variable

ED\$	Picture to be edited
------	----------------------

DO Function Variables

D(C)	DO counter
C	DO level counter
SA\$(n)	DO loop command
SB\$(n)	Remainder of IN\$ for DO function

Increment Function Variables

IG	Increment flag
IC	Increment counter
IV	Increment value

Variable-Name Variables

VC	Variable counter
VA\$(n)	Variable value

Miscellaneous Variables

FC	Frees memory of old string variable values
H,I,J,L,L1	Loop variables

TURTLE GRAPHICS COMMANDS

COMMAND	EXPLANATION	EXAMPLE
FDn	Move forward n units	FD40
BKn	Move backward n units	BK30
RTn	Turn right n degrees	RT45
LTn	Turn left n degrees	LT90
PU	Pen-up (move without drawing)	PU
PD	Pen-down	PD
NT	No turtle (speeds execution)	NT
TU	Turtle	TU
DO n(xxx)	Repeat (do instructions xxx, n number of times)	DO4(FD20.RT90)
INn	Increment all variables by n (used with repetition)	IN2
.	Period is used to separate instructions	FD20.RT45.BK60
>xxx	Greater than symbol is used to indicate a picture name	>HEXAGON
any name	Variable name (may be used with FD, BK, RT, LT, and IN)	FDSIDE or RT ANGLE
HO	Home (sends turtle to the center of the screen, headed toward the top of the screen)	HO
ED >xxx	Edit picture xxx	ED>SQUARE
CL	Clear (clears screen and sends turtle home)	CL
FI	File (presents filing options)	FI
QU	Quit (ends program, clears screen and returns user to Applesoft in text mode)	QU

TURTLE GRAPHICS PROGRAM SEGMENTS

TURTLE GRAPHICS VARIABLE LISTING

STARTING

LINE #	FUNCTION
10	Set up, ask instructions, check instructions
300	Parse instructions
400	Determine quantity (QU)
500	Move turtle
600	Turn turtle and change heading
700	Erase turtle

Command Variables

IN\$	Instructions
I\$	First two letters of IN\$
KY	Keyboard strobe for interrupt

Turtle Variables

X	Positions for start and end of movement
Y	
X1	
Y1	
WI	Dimensions of turtle
TA	



1 Turtle Graphics
by David Krathwohl
Copyright © 1982
1 by Micro SPARC Inc.

```

20 LOMEM: 16400: TEXT : HOME : GOSUB 2800: HGR : GOSUB
800
40 HOME :FC = FC + 1: IF FC = 25 THEN FC = FRE (1):FC =
0
45 VTAB 22: INPUT "WHAT DO YOU WANT THE TURTLE TO DO?"
:IN$
50 BP = 0:VC = 0:IV = 0:IC = 0:C = 0: ONERR GOTO 285
60 IF IN$ = "" THEN 40
70 IF RIGHT$(IN$,1) = "." THEN IN$ = LEFT$(IN$, (LEN
(IN$) - 1))
80 I$ = LEFT$(IN$,2)
90 KY = PEEK (- 16384): IF KY > 127 THEN POKE - 1636
8,0: GOTO 40
100 IF I$ = "FD" THEN GOSUB 400: GOSUB 500: GOTO 300
110 IF I$ = "BK" THEN B = 1: GOSUB 400: GOSUB 500: GOTO
300
120 IF I$ = "RT" THEN GOSUB 400: GOSUB 600: GOTO 300
130 IF I$ = "LT" THEN LE = 1: GOSUB 400: GOSUB 600: GOTO
300
140 IF LEFT$(IN$,1) = ">" THEN 1100
150 IF I$ = "DO" THEN 1300
160 IF I$ = "/" THEN D(C) = D(C) - 1: GOTO 1470
170 IF I$ = "PU" THEN DR = 0: GOTO 300
180 IF I$ = "PD" THEN DR = 1: GOTO 300
190 IF I$ = "NT" THEN T = 1: GOSUB 700: GOTO 300
200 IF I$ = "TU" THEN T = 0: GOSUB 800: GOTO 300
210 IF I$ = "HD" THEN GOSUB 1600: GOTO 300
220 IF I$ = "CL" THEN GOSUB 1700: GOTO 40
230 IF I$ = "FI" THEN 2000
240 IF I$ = "ED" THEN 1800
250 IF I$ = "QU" THEN TEXT : HOME : END
260 IF I$ = "IN" THEN IG = 1: GOSUB 400: GOTO 300
270 IF BP = 1 THEN ED$ = PI$(PI): TEXT : HOME : GOTO 19
00
280 HOME : VTAB 22: PRINT "I DON'T UNDERSTAND ";IN$: GOSUB
2700: GOTO 40
285 CALL 807: GOTO 280
299 REM * PARSE INSTRUCTIONS *
300 FOR J = 1 TO (LEN (IN$) - 1)
310 IF MID$(IN$,J,1) = "." THEN IN$ = RIGHT$(IN$, (LEN
(IN$) - J)): GOTO 60
320 NEXT J
330 GOTO 40
399 REM * DETERMINE QUANTITY *
400 FOR J = 1 TO LEN (IN$)
410 IF MID$(IN$,J,1) = "." THEN QU = VAL (MID$(IN$
,3,J - 3)): GOTO 440
420 NEXT J
430 QU = VAL (MID$(IN$,3))
440 IF QU = 0 THEN GOSUB 1500: GOTO 460
450 IF IG = 1 THEN IC = IC + 1:IV = QU:IG = 0
460 RETURN
499 REM * MOVE TURTLE *
500 GOSUB 700
510 IF B = 1 THEN TU = 180 + TU
520 X1 = X + SIN (TU * CV) * QU + .00005:Y1 = Y - COS
(TU * CV) * QU + .00005
530 IF X1 < 9 OR Y1 < 9 OR X1 > 270 OR Y1 > 139 THEN 58
0
540 IF DR = 0 THEN 555
550 HPLLOT X,Y TO X1,Y1
555 IF B = 1 THEN TU = TU - 180
560 B = 0:X = X1:Y = Y1
570 GOSUB 800: RETURN
580 HOME : VTAB 22: PRINT "THE TURTLE WOULD BE OUT OF B
OUNDS. TRY A DIFFERENT MOVE.
": GOSUB 2700:X1 = X:Y1 = Y: IF B = 1 THEN B = 0:TU
= TU - 180
590 POP : GOSUB 800: GOTO 40
599 REM * TURN TURTLE *
600 GOSUB 700
610 IF LE = 1 THEN QU = 360 - QU:LE = 0
620 IF ABS (QU) > 360 THEN QU = INT ((QU / 360 - INT
(QU / 360)) * 360.05) * SGN (QU)
630 TU = TU + QU: GOSUB 800: RETURN
699 REM * ERASE TURTLE *
700 IF T > 1 THEN T = T + 1: IF T > 2 THEN RETURN
710 HCOLOR= 0
720 HPLLOT X2,Y2: HPLLOT X3,Y3: HPLLOT X4,Y4: HPLLOT X5,Y5:
HPLLOT X6,Y6
730 HCOLOR= 3
740 RETURN
799 REM * DRAW TURTLE *
800 IF T > 1 THEN RETURN
810 GOSUB 900: RETURN
899 REM * TURTLE LOCATION *
900 WI = 6:TA = 7

```

```

910 X2 = X1 - (COS (TU * CV)) * WI:Y2 = Y1 - (SIN (TU *
CV)) * WI
920 X3 = X1 + (SIN (TU * CV)) * TA:Y3 = Y1 - (COS (TU *
CV)) * TA
930 X4 = X1 + (COS (TU * CV)) * WI:Y4 = Y1 + (SIN (TU *
CV)) * WI
940 X5 = X2 + ((X3 - X2) / 2):Y5 = Y2 + ((Y3 - Y2) / 2)
950 X6 = X3 - ((X3 - X4) / 2):Y6 = Y3 - ((Y3 - Y4) / 2)
960 & BT = X2,Y2: & CT = X3,Y3: & DT = X4,Y4: & ET = X5
,Y5: & FT = X6,Y6
970 IF BT = 1 THEN Y2 = 170
980 IF CT = 1 THEN Y3 = 170
990 IF DT = 1 THEN Y4 = 170
1000 IF ET = 1 THEN Y5 = 170
1010 IF FT = 1 THEN Y6 = 170
1020 HPLLOT X2,Y2: HPLLOT X3,Y3: HPLLOT X4,Y4: HPLLOT X5,Y5
: HPLLOT X6,Y6: RETURN
1099 REM * CHECK FOR PICTURE *
1100 PI = PI + 1
1120 FOR J = 1 TO (LEN (IN$) - 1)
1130 IF MID$(IN$,J,1) = "." THEN PI$(PI) = LEFT$(IN
$,J - 1):IN$ = RIGHT$(IN$, (LEN (IN$) - J)): GOTO
1160
1140 NEXT J
1150 PI$(PI) = IN$:IN$ = ""
1160 FOR I = 0 TO (PI - 1)
1170 IF PI$(I) = PI$(PI) THEN IN$ = FI$(I) + "." + IN$:
PI = PI - 1: GOTO 60
1180 NEXT I
1190 HOME : VTAB 22: PRINT "I DON'T KNOW HOW TO ";PI$(P
I): PRINT : PRINT "PLEASE TELL ME HOW"
1200 GOSUB 2700
1210 HOME : VTAB 22: PRINT "TO ";PI$(PI): INPUT " ";FI
$(PI)
1220 IN$ = FI$(PI) + "." + IN$:BP = 1: GOTO 60
1299 REM * REPEAT FUNC. (DO) *
1300 C = C + 1
1310 FOR J = 1 TO LEN (IN$)
1320 IF MID$(IN$,J,1) = "(" THEN 1350
1330 NEXT J
1340 C = C - 1: GOTO 280
1350 FOR L = LEN (IN$) TO 1 STEP - 1
1360 IF MID$(IN$,L,1) = ")" THEN 1390
1370 NEXT L
1380 C = C - 1: GOTO 280
1390 FOR L1 = J TO L
1400 IF MID$(IN$,L1,1) = ")" THEN 1420
1410 NEXT L1
1420 IF L1 < L THEN L = L1
1430 SA$(C) = MID$(IN$,J + 1,L - J - 1)
1440 IF L < LEN (IN$) THEN SB$(C) = MID$(IN$,L + 2):
GOTO 1450
1445 SB$(C) = ""
1450 D(C) = VAL (MID$(IN$,3,J - 3))
1460 IN$ = SA$(C) + "://": GOTO 60
1470 IF D(C) = 0 AND SB$(C) = "" THEN C = 0: GOTO 40
1480 IF D(C) = 0 THEN IN$ = SB$(C):SB$(C) = "":C = C -
1: GOTO 60
1490 IN$ = SA$(C) + "://": GOTO 60
1499 REM * VARIABLE FUNCTION *
1500 VC = VC + 1
1510 VA$(VC) = MID$(IN$,3,J - 3)
1520 FOR L = 0 TO VC - 1
1530 IF VA$(L) = VA$(VC) THEN QU = VA$(L) + IV * IC:VC =
VC - 1: GOTO 460
1540 NEXT L
1550 HOME : VTAB 22: PRINT "WHAT IS THE VALUE OF ";VA$(
VC): INPUT VA$(VC)
1560 QU = VA$(VC): RETURN
1599 REM * HOME *
1600 GOSUB 700:X = X1:Y = Y1:X1 = 139:Y1 = 79
1610 IF DR = 0 THEN 1630
1620 HPLLOT X,Y TO X1,Y1
1630 TU = 0:X = X1:Y = Y1: GOSUB 800: RETURN
1699 REM * CLEAR (CL) *
1700 HGR :X = 139:Y = 79:X1 = X:Y1 = Y
1710 TU = 0:T = 0:DR = 1: GOSUB 800: RETURN
1799 REM * PICTURE EDITOR *
1800 TEXT : HOME
1810 FOR J = 1 TO LEN (IN$)
1820 IF MID$(IN$,J,1) = ")" THEN 1850
1830 NEXT J
1840 GOTO 280
1850 ED$ = MID$(IN$,J)
1860 FOR I = 1 TO PI
1870 IF PI$(I) = ED$ THEN HOME : GOTO 1900
1880 NEXT I
1890 IN$ = ED$: GOTO 1100
1900 INVERSE : PRINT : HTAB 10: PRINT "* PICTURE EDITO
R *": NORMAL
1910 PRINT : PRINT : INVERSE : PRINT "EDITING:": NORMAL
: PRINT ED$

```

continued on next page

Apple Turtle Graphics (Cont.)

```

1920 PRINT : INVERSE : PRINT "THE INSTRUCTIONS FOR THIS
      PICTURE ARE:" : NORMAL
1930 PRINT : PRINT FI$(I)
1940 INVERSE : VTAB 15: PRINT "WHAT SHOULD THEY BE?": NORMAL

1950 VTAB 17: HTAB 2: PRINT FI$(I)
1960 VTAB 17: INPUT FI$(I)
1970 IN$ = "CL": GOTO 50
1999 REM * FILING SYSTEM *
2000 TEXT : HOME : D$ = CHR$(4): ONERR GOTO 2640
2010 VTAB 5: PRINT "OPTIONS AVAILABLE:

      ) SAVE PICTURES TO A FILE          (2) RETRIEVE
      PICTURES FROM A FILE          (3) DELETE A FILE"
2020 PRINT "      (4) CATALOG AND INDEX FILES
      (5) RETURN TO GRAPHICS MODE
2030 VTAB 22: PRINT SPC(40): VTAB 22: INPUT "TYPE THE
      NUMBER OF YOUR CHOICE. "; OP$: OP = VAL (OP$)
2040 IF OP < 1 OR OP > 5 THEN 2030
2050 ON OP GOTO 2060,2190,2320,2380,2630
2060 IF PI = 0 THEN HOME : VTAB 12: PRINT "THERE ARE N
      0 PICTURES TO SAVE": GOSUB 2700: GOTO 2000
2070 GOSUB 2620
2080 HOME : VTAB 12: HTAB 15: FLASH : PRINT "SAVING": NORMAL

2090 PRINT D$;"OPEN ";NA$
2100 PRINT D$;"DELETE ";NA$
2110 PRINT D$;"OPEN ";NA$
2120 PRINT D$;"WRITE ";NA$
2130 PRINT PI
2140 FOR I = 1 TO PI
2150 PRINT PI$(I): PRINT FI$(I)
2160 NEXT I
2170 PRINT D$;"CLOSE ";NA$
2180 GOTO 2000
2190 HOME : VTAB 5: INVERSE : PRINT "WARNING:": NORMAL

2200 PRINT "RETRIEVING A FILE WILL DESTROY ANY PICT
      URES YOU CURRENTLY HAVE IN MEMORY."
2210 PRINT : PRINT "IF YOU WISH TO CONTINUE, PRESS 'C'
      PRESS ANY OTHER KEY TO RETURN TO FILE ACCESS
      OPTIONS."
2220 INPUT K$
2230 IF K$ < > "C" THEN 2000
2240 GOSUB 2620
2250 HOME : FLASH : VTAB 12: HTAB 14: PRINT "RETRIEVING
      ": NORMAL
2260 GOSUB 2500
2270 PI = NU
2280 FOR I = 1 TO PI
2290 PI$(I) = PX$(I):FI$(I) = FX$(I)
2300 NEXT I
2310 GOTO 2000
2320 GOSUB 2620
2330 PRINT : PRINT : INPUT "ARE YOU SURE? ";K$
2340 IF LEFT$(K$,1) < > "Y" THEN 2000
2350 HOME : FLASH : VTAB 12: HTAB 14: PRINT " DELETING
      ": NORMAL
2360 PRINT D$;"DELETE ";NA$
2370 GOTO 2000
2380 HOME : PRINT "YOUR FILE ENTRIES ARE PRECEDED BY A
      "; CHR$(34);"T"; CHR$(34)
2390 PRINT D$;"CATALOG"
2400 PRINT : INVERSE : INPUT "DO YOU WANT TO SEE WHICH
      DRAWINGS ARE IN A CERTAIN FILE? ";K$: NORMAL
2410 IF LEFT$(K$,1) < > "Y" THEN 2000
2420 GOSUB 2620
2430 HOME : VTAB 12: HTAB 14: FLASH : PRINT " SEARCHING
      ": NORMAL
2440 GOSUB 2500
2450 HOME : FOR L = 1 TO NU
2460 PRINT PX$(L);": ";FX$(L)
2470 GOSUB 2600: PRINT
2480 NEXT L
2490 GOSUB 2590: GOTO 2000
2500 PRINT D$;"OPEN ";NA$
2510 PRINT D$;"READ ";NA$
2520 INPUT NU
2530 FOR I = 1 TO NU
2540 INPUT PX$(I): INPUT FX$(I)
2550 NEXT I
2560 PRINT D$;"CLOSE ";NA$
2570 RETURN
2580 GOSUB 2590: GOTO 2000
2590 VTAB 24: HTAB 5: INVERSE : PRINT "PRESS ANY KEY TO
      CONTINUE": NORMAL
2600 GET K$: IF K$ = "" THEN 2600
2610 RETURN

```



```

2620 HOME : VTAB 12: INPUT "FILE NAME? ";NA$: RETURN
2630 IN$ = "CL": GOTO 60
2640 CALL 807: IF PEEK (222) = 5 THEN PRINT D$;"DELET
      E ";NA$
2650 HOME : VTAB 12: PRINT "CHECK YOUR FILE NAME AND TR
      Y AGAIN.": GOSUB 2700: GOTO 2000
2699 REM * TIME DELAY *
2700 FOR H = 1 TO 1000: NEXT H: RETURN
2799 REM * SET UP VARIABLES *
2800 TU = 0:X = 139:Y = 79:CV = 3.14159265 / 180:DR = 1:
      HCOLOR= 3
2810 DIM PI$(100): DIM FI$(100):PI = 0: DIM PX$(100): DIM
      FX$(100)
2820 POKE - 16301,0
2830 FOR J = 768 TO 816: READ K: POKE J,K: NEXT J
2840 DATA 32,227,223,133,133,132,134,169,208,32
2850 DATA 192,222,165,18,72,165,17,72,32,185
2860 DATA 246,32,17,244,165,48,49,38,240,2
2870 DATA 169,1,168,32,1,227,76,91,218
2875 DATA 104,168,104,166,223,154,72,152,72,96: REM E
      RROR TRAP SUBROUTINE FROM P.82 OF APPLESOFT MANUAL.
      STARTS AT 807
2880 POKE 1013,76: POKE 1014,0: POKE 1015,3
2890 X1 = X:Y1 = Y: RETURN

```

1



APPLESOFT LINE CRUNCHER

Applesoft Line Cruncher

by Alan and Valerie Floeter
The Software Experience
4333 N. 71 St.
Milwaukee, WI 53216

In the past few years we have published several utilities to automatically shorten Applesoft programs. These include methods such as renumbering programs to lower numbered lines, removing REM statements, and reducing variable names to single characters.

There is yet another method available to Apple users. **Applesoft allows more than one statement on a line.** By separating them with a colon (:), several statements can be placed on a single line. This is very commonly used in programs. An example of this is as follows

100 X=3 : INPUT Y : REM EXAMPLE OF MULTI-STATEMENTS

This line contains three statements that could have been written separately and still execute the same.

But there obviously would be a difference internally in how they are stored. A line in Applesoft is always at least six bytes long. **Bytes 1 and 2** store the address of the next line in memory. **Bytes 3 and 4** hold the line number in hexadecimal. **Bytes 5 and beyond** hold any Applesoft tokens in the line, and the **last byte in a line is a 0.**

If two lines are combined into one, the savings in memory is four bytes. If multiple lines are combined, it is easy to see that the savings in memory will really add up.

For this reason the Applesoft Line Cruncher was written. It searches through an Applesoft program, automatically joining lines when possible.

CRUNCH CRITERIA

Several criteria need to be met before two lines are joined. Generally, when you join two lines, separated by a colon, the execution will be the same. However, this is not the case with the **REM**, **IF-THEN**, and **ONERR GOTO** commands. Any executable code following a **REM** command is treated as part of a remark and therefore not executed. So a line could not be added to a line already containing a **REM**.

The **IF-THEN** statement in Applesoft is very different from that in an Integer program. In Integer, when the **IF** part of a statement is false, only the **THEN** portion is skipped. However, in Applesoft, when the **IF** portion is false, the rest of the line is skipped, and execution continues with the next line number. You can see that combining a line after an **IF-THEN** would also cause an error in the execution.

It is also interesting to note that any code following an **ONERR GOTO** command is not executed when on the same line. Therefore, it is treated the same as the **REM** statement.

Besides checking for **REM**'s and **IF-THEN**'s, the line number to be dropped must be checked to see if it is referenced in another statement. Line numbers are used in **GOTO**'s,

```

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
; APPLESOFT LINE CRUNCHER
;
; BY AL AND VALERIE FLOETER
; THE SOFTWARE EXPERIENCE
; COPYRIGHT (C) 1982
; BY MICRO-SPARC INC.
; ALL RIGHTS RESERVED
; THIS ROUTINE COMPRESSES LINES INTO
; MULTI- STATEMENT LINES FOR AN APPLESOFT
; PROGRAM.
;
; THIS IS DONE BY CHECKING IF LINE NUMBERS
; ARE USED, AND COMBINING THOSE UNUSED
; WHENEVER IT CAN.
;
; IT CHECKS THAT AN IF-THEN STATEMENT OR REM
; DOESN'T INTERFERE WITH PROPER EXECUTION.
;
; IT ALSO MAKES SURE THAT A LINE IN MEMORY
; DOES NOT TAKE MORE THAN MAXL BYTES.
; (SET TO 239 FOR APPLESOFT MAX)
;
; TO RUN, LOAD IN THE APPLESOFT ROUTINE
; AND CALL 24576, OR BRUN LINECOMP.
;
CR      EQU  $FDBE
DOUT    EQU  $FDED
IND      EQU  $FC
PIND     EQU  $FE
YSAV     EQU  $FB
BEGN     EQU  $67
PEND     EQU  $AF
LOM      EQU  $69
;
;      DSC  $19
YPOS     DFS 1
SER       DFS 2
;
;
;      ORG  $9200
;
;      JMP  START
MAXL     DFC 239,239/256
;
START    LDA  $AA59      ;FOR BRUN FROM DOS
          PHA
          LDA  $6A59
          PHA
          LDA  BEGN
          STA  IND
          LDA  BEGN+1
          STA  IND+1
          JSR  CR
NEXTLN   LDY  #1
          LDA  (IND),Y
          BEQ  QUITIT
          LDY  #4
          JSR  MAINLP
          LDY  #0
          LDA  (IND),Y
          PHA
          INY
          LDA  (IND),Y
          STA  IND+1      ;NEW HI ORDER
          PLA
          STA  IND        ;LO ORDER
          JMP  NEXTLN
QUITIT   PLA
          STA  $6A59
          PLA
          STA  $AA59
          RTS
;
; MAIN LOOP
;
MAINLP   LDA  IND+1
          STA  INDSAV
          LDA  (IND),Y
          BEQ  ENDMAN
          CMP  #178
          BEQ  NOCOMB
          CMP  #196
          BEQ  NOCOMB
          CMP  #165
          BEQ  NOCOMB
          INY
          BNE  MAIN
          INC  IND+1
          BNE  MAIN
;GET TOKEN
;END OF LINE
;REM?
;YES-DONT COMBINE
;THEN?
;YES-DONT COMBINE
;ON ERR?
;YES
;NEXT CHARACTER
          
```

continued on next page

listed on next page

Applesoft Line Cruncher (Cont.)

GOSUB's, THEN's, DEL's, LIST's, and RUN's. Obviously, if a line number is used, then it can't be combined.

Lastly, the total length of a line can't be too long, or the crunched program will not be able to be edited. The Program Line Editor (PLE) sold through Call-A.P.P.L.E. can only edit lines containing less than 239 characters, (as distinguished). For instance, PRINT is 5 characters, but only a one byte token. So obviously, if you use PLE or a program similar to it, you would want to restrict the line length to something acceptable to those utilities. So, before joining lines, the length must also be checked. Applesoft cannot run with line lengths greater than 256.

With these four considerations, the Applesoft Line Cruncher will check each line for joining. The Applesoft program is compressed and transformed into as few lines as possible.

SYSTEM REQUIREMENTS

The Line Cruncher is written in assembly language and can be run "as is" on 48K systems, with either disk or cassette. Re-ORG the program to \$5200 for a 32K system.

ENTERING CRUNCHER

To enter the program, type in either the hex codes or instructions from the source listing. NOTE: At the symbol **MAXL** a pseudo-op is used that will not be accepted by the mini-assembler and possibly other assemblers as well. **DFC** means to store the hex values following it directly into memory. You must use the monitor to enter those directly into memory, unless you have an assembler with similar pseudo-ops. But also be aware that these are data, not executable instructions, so that if you try to disassemble that part of the program, you may find "???"s and other assorted nonsense. Do not let this bother you as long as the hex codes match those given in the listing.

To save the program

BSAVE LINECRCH,A\$9200,L\$3E7 (for disk)
or
9200.95FFW (for cassette).

To run the program,

1) Load in Applesoft program to be compressed.

2) Load in LINECRCH

3) CALL 37376

LINECRCH can also be BRUN'd.

IN OPERATION

During execution the program lists out any lines that are being merged. This is shown by the line number that will result from the merge, an arrow, and the number of the line that is being merged into the first line. If a hardcopy listing of these changes is desired, activate the printer before BRUN-ing the program. When the program is finished executing, the Applesoft prompt will appear on the screen. You can then continue working with your compressed program.

We have allowed the user to select the length of the line. At the symbol MAXL there

89	9258	BC E8 95	ENDMAN	STY TEMP	;SAVE END OF LINE
90	925B	A5 FD		LDA IND+1	
91	925D	48		PHA	
92	925E	AE E7 95		LDX INDSAV	
93	9261	86 FD		STX IND+1	
94	9263	A0 00		LDY #0	
95	9265	B1 FC		LDA (IND),Y	;GET ADDRESS OF NEXT LINE
96	9267	85 FE		STA PIND	
97	9269	C8		INY	
98	926A	B1 FC		LDA (IND),Y	;HI ORDER
99	926C	F0 56		BEQ NOCMB2	;END OF PROGRAM
100	926E	85 FF		STA PIND+1	
101	9270	B1 FE		LDA (PIND),Y	
102	9272	F0 50		BEQ NOCMB2	;NONE THERE
103	9274	C8		INY	
104	9275	B1 FE		LDA (PIND),Y	;GET LINE NUMBER
105	9277	8D E9 95		STA TLO	
106	927A	C8		INY	
107	927B	B1 FE		LDA (PIND),Y	
108	927D	8D EA 95		STA THI	
109	9280	20 5D 93		JSR SEAR	;SEARCH FOR THIS LINE USED
110	9283	F0 3F		BEQ NOCMB2	;USED SO DONT COMBINE
111	9285	20 D4 94		JSR GETSIZ	
112	9288	AD ED 95		LDA TOT+1	
113	928B	CD 04 92		CMP MAXL+1	;CHECK HI ORDER
114	928E	90 0C		BCC DOCOMB	;DO COMBINE
115	9290	D0 32		BNE NOCMB2	;TOO LARGE
116	9292	AD EC 95		LDA TOT	
117	9295	CD 03 92		CMP MAXL	;CHECK LO ORDER
118	9298	90 02		BCC DOCOMB	;DO IT
119	929A	D0 2B		BNE NOCMB2	;DONT
120	929C	A0 00	DOCOMB	LDY #0	
121	929E	B1 FE		LDA (PIND),Y	
122	92A0	91 FC		STA (IND),Y	
123	92A2	C8		INY	
124	92A3	B1 FE		LDA (PIND),Y	
125	92A5	91 FC		STA (IND),Y	;FOR COMBINING
126	92A7	20 30 95		JSR PRINT	
127	92AA	AC E8 95		LDY TEMP	;GET END OF LINE
128	92AD	68		PLA	
129	92AE	85 FD		STA IND+1	
130	92B0	A9 3A		LDA #"-#80	;TO SEPARATE NEW LINE
131	92B2	91 FC		STA (IND),Y	;IN PLACE OF 0
132	92B4	C8		INY	
133	92B5	84 19		STY YPOS	
134	92B7	20 CB 92		JSR PACK	;PACK IT DOWN
135	92BA	A0 04		LDY #4	
136	92BC	AD E7 95		LDA INDSAV	
137	92BF	85 FD		STA IND+1	
138	92C1	4C 41 92		JMP MAIN	;TRY AGAIN
139	92C4	68	NOCMB2	PLA	
140	92C5	AD E7 95	NOCOMB	LDA INDSAV	
141	92C8	85 FD		STA IND+1	
142	92CA	60		RTS	
143					
144					
145	92CB	A9 04		LDA #4	;POSITION FOR END OF LINE
146	92CD	85 FB	PACK	STA YSAV	;SAVE IT
147	92CF	A5 FC		LDA IND	
148	92D1	48		PHA	
149	92D2	85 FE		STA PIND	
150	92D4	A5 FD		LDA IND+1	
151	92D6	48		PHA	
152	92D7	AD E7 95		LDA INDSAV	
153	92DA	85 FF		STA PIND+1	
154	92DC	A0 01		LDY #1	
155	92DE	B1 FE	KEEP	LDA (PIND),Y	;CHANGE ADDRESS POINTERS
156	92E0	F0 19		BEQ ZMOR	;END
157	92E2	48		PHA	;SAVE HI ORDER
158	92E3	88		DEY	
159	92E4	B1 FE		LDA (PIND),Y	;LO ORDER
160	92E6	48		PHA	;SAVE IT TOO
161	92E7	38		SEC	
162	92E8	E5 FB		SBC YSAV	;BACK IT OFF
163	92EA	91 FE		STA (PIND),Y	;NEW LO ORDER
164	92EC	C8		INY	
165	92ED	B1 FE		LDA (PIND),Y	;HI,ORDER
166	92EF	E9 00		SBC #0	;IF OVERFLOW
167	92F1	91 FE		STA (PIND),Y	;NEW HI ORDER
168	92F3	68		PLA	
169	92F4	85 FE		STA PIND	
170	92F6	68		PLA	
171	92F7	85 FF		STA PIND+1	;NEXT LINE
172	92F9	B0 E3		BCS KEEP	;CARRY SET HERE
173	92FB	18	ZMOR	CLC	
174	92FC	A5 19		LDA YPOS	;GET WHERE TO PACK TO
175	92FE	65 FC		ADC IND	
176	9300	85 FC		STA IND	
177	9302	48		PHA	;SAVE IT FOR PIND
178	9303	AD E7 95		LDA INDSAV	
179	9306	69 00		ADC #0	
180	9308	85 FD		STA IND+1	
181	930A	68		PLA	
182	930B	65 FB		ADC YSAV	;TO OLD NEXT LINE

Applesoft Line Cruncher (Cont.)

are two locations to set. This is currently set up for a 239 character maximum for those who use PLE. To set this from Applesoft the following instructions could be used.

POKE 37379,200

This sets the line length to 200 characters. It is interesting to note that this program could create such long lines, that they couldn't be edited by copying over them or by entering them from the keyboard. So if you're developing a program which others will enter manually into their machines, make sure that you have MAXL set to 239 or less. However, if you give them a copy on tape or diskette, by having lines longer than 239, you make it hard to modify without a lot of typing.

SAVES TIME

Another indirect benefit of using the Line Compressor is a reduction in execution time. If your Applesoft program has quite a few GOTO's, etc., the savings could be substantial. However, this isn't really a very predictable occurrence. In one test we ran the speed improved by only 1%, so this shouldn't be the main reason for using the Line Compressor.

When it comes to shortening an Applesoft program, using multi-statements on a line is a good method. The Applesoft Line Cruncher is a nice utility to add to your collection. It sure beats doing the editing by hand.

LINE CRUNCHER DEMO

LIST

```
10 X = 1
20 Y = 2
30 Z = 3
40 A = 1
50 B = 2
70 IF A = X THEN B = B + 2 * 3
80 REM THIS LINE WILL HAVE NOTHING APPENDED TO IT.
90 ON A GOTO 100,200
100 PRINT "LINE 100": END
200 PRINT "LINE 200": END
400 PRINT "THIS WILL BE A VERY VERY LONG LINE IN ORDER TO ILLUSTRATE THE LINE LENGTH COUNTING CHARACTERISTICS OF APPLESOFT LINE CRUNCHER. SINCE IT VIRTUALLY FILLS THE LINE, IT WILL HAVE NO VARIABLES APPENDED TO IT BY LINE CRUNCHER"
410 XX = 1
420 YY = 2
450 XX = XX + A + B + X + Y + Z
500 Z = 10:Q = 1:P = 2:T = 3:Q = Q + P + T: PRINT "END"
```

CALL 37376

```
10 <-- 20
10 <-- 30
10 <-- 40
10 <-- 50
10 <-- 70
410 <-- 420
410 <-- 450
410 <-- 500
```

```
183 930D 85 FE STA PIND
184 930F A5 FD LDA IND+1
185 9311 69 00 ADC #0
186 9313 85 FF STA PIND+1
187 9315 A0 00 LDY #0
188 9317 B1 FE OK LDA (PIND),Y ; MOVE DOWN EVERYBODY!!
189 9319 91 FC STA (IND),Y
190 931B E6 FC INC IND
191 931D D0 02 BNE NOHI
192 931F E6 FD INC IND+1
193 9321 E6 FE NOHI INC PIND
194 9323 D0 02 BNE NOHI2
195 9325 E6 FF INC PIND+1 ; NEXT LOCATION
196 9327 A5 FF NOHI2 LDA PIND+1
197 9329 C5 B0 CMP PEND+1 ; END OF APPLESOFT?
198 932B 90 EA BCC OK ; NO
199 932D A5 FE LDA PIND
200 932F C5 AF CMP PEND ; LO ORDER END?
201 9331 90 E4 BCC OK ; NO
202 9333 A5 AF LDA PEND
203 9335 E5 FB SBC YSAV ; CHANGE END
204 9337 85 AF STA PEND ; NEW END
205 9339 85 69 STA LOM ; LOMEM TOO
206 933B 85 68 STA LOM+2
207 933D 85 6D STA LOM+4
208 933F A5 B0 LDA PEND+1
209 9341 E9 00 SBC #0 ; HI ORDER
210 9343 85 B0 STA PEND+1
211 9345 85 6A STA LOM+1
212 9347 85 6C STA LOM+3
213 9349 85 6E STA LOM+5
214 934B A0 00 LDY #0
215 934D 98 TYA ; Y=0
216 934E 91 FC STA (IND),Y ; END OF PROGRAM HAS 0'S
217 9350 C8 INY
218 9351 91 FC STA (IND),Y
219 9353 C8 INY
220 9354 91 FC STA (IND),Y
221 9356 68 PLA
222 9357 85 FD STA IND+1
223 9359 68 PLA
224 935A 85 FC STA IND
225 935C 60 RTS
226 ;
227 ;
228 ; SEARCH FOR LINE USED
229 ;
230 935D A5 67 SEAR LDA BEGN ; START OF PROGRAM
231 935F 85 1A STA SER
232 9361 A5 68 LDA BEGN+1
233 9363 85 1B STA SER+1
234 9365 A0 04 NOTB LDY #4
235 9367 A5 1B LDA SER+1
236 9369 48 PHA
237 936A B1 1A NOTY LDA (SER),Y
238 936C F0 25 BEQ LINE
239 936E C9 C5 CMP #197
240 9370 B0 1A BCS NO ; NOT VALID TOKENS
241 9372 C9 85 CMP #133
242 9374 90 16 BCC NO ; NONE HERE EITHER
243 9376 F0 32 BEQ FOUT ; GOT ONE
244 9378 C9 AB CMP #171 ; LOOK FOR TOKENS
245 937A F0 2E BEQ FOUT
246 937C C9 AC CMP #172
247 937E F0 2A BEQ FOUT
248 9380 C9 C4 CMP #196
249 9382 F0 26 BEQ FOUT
250 9384 C9 B0 CMP #176
251 9386 F0 22 BEQ FOUT
252 9388 C9 BC CMP #188
253 938A F0 1E BEQ FOUT
254 938C C8 NO INY ; NONE FOUND
255 938D D0 DB BNE NOTY
256 938F E6 1B INC SER+1
257 9391 D0 D7 BNE NOTY
258 9393 A0 00 LINE LDY #0
259 9395 68 PLA
260 9396 85 1B STA SER+1
261 9398 B1 1A LDA (SER),Y ; GET NEXT LINE
262 939A 48 PHA
263 939B C8 INY
264 939C B1 1A LDA (SER),Y ; GET HI ORDER
265 939E 85 1B STA SER+1
266 93A0 68 PLA
267 93A1 85 1A STA SER
268 93A3 B1 1A LDA (SER),Y ; END?
269 93A5 D0 BE BNE NOTB ; NO
270 93A7 A9 01 LDA #1 ; NOT EQUAL
271 93A9 60 RTS ; YES
272 ;
273 93AA 20 CF 93 FOUT JSR BASE ; CONVERT TO BASE 10
274 93AD AD EB 95 LDA FLAG ; ANY FOUND?
275 93B0 F0 14 BEQ NOT3 ; NO
276 93B2 AD EC 95 LDA TOT
277 93B5 CD E9 95 CMP TLO ; SAME?
```

continued on next page

continued on next page

Apple Soft Line Cruncher (Cont.)

LIST

10 X = 1; Y = 2; Z = 3; A = 1; B = 2;
IF A = X THEN B = B + 2 * 3

80 REM THIS LINE WILL HAVE NOTHING APPENDED TO IT.

90 ON A GOTO 100,200

100 PRINT "LINE 100"; END

200 PRINT "LINE 200"; END

400 PRINT "THIS WILL BE A VERY VERY LONG LINE IN ORDER TO ILLUSTRATE THE LINE LENGTH COUNTING CHARACTERISTICS OF APPLESOFT LINE CRUNCHER. SINCE IT VIRTUALLY FILLS THE LINE, IT WILL HAVE NO VARIABLES APPENDED TO IT BY LINE CRUNCHER"

410 XX = 1; YY = 2; XX = XX + A + B + X + Y + Z; Z = 10; Q = 1; P = 2; T = 3; Q = Q + P + T; PRINT "END"

1



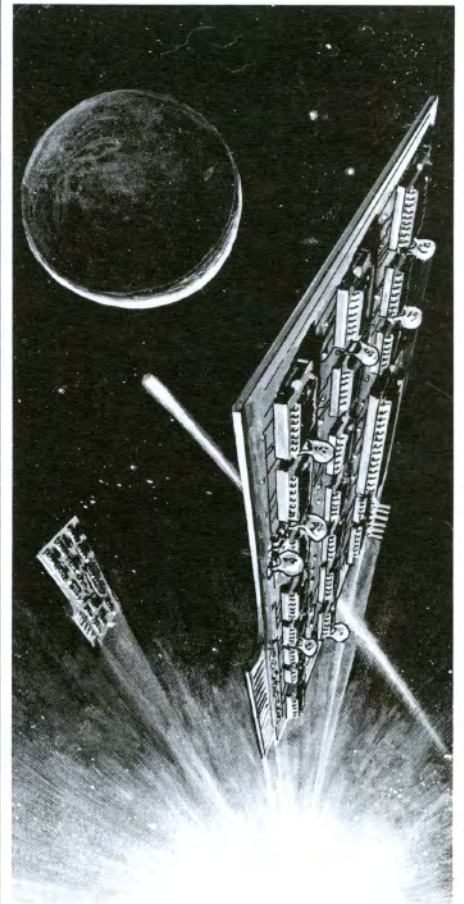
"Oh, Yes — it is an Apple, but I call it of prune juice because it's always up and running"

278	93B8	D0 0C	BNE NOT3	
279	93BA	AD ED 95	LDA TOT+1	
280	93BD	CD EA 95	CMP THI	; STILL SAME?
281	93C0	D0 04	BNE NOT3	; NO
282	93C2	68	PLA	
283	93C3	A9 00	LDA #0	
284	93C5	60	RTS	; FLAGS EQUAL
285	93C6	B1 1A	NOT3 LDA (SER),Y	
286	93C8	C9 2C	CMP #"-#80	; MORE NUMBERS?
287	93CA	F0 DE	BEQ F0UD	; YES
288	93CC	4C 6A 93	JMP NOTY	; NO
289				
290				; CONVERT TO BASE 10
291				
292	93CF	A9 00	BASE LDA #0	
293	93D1	8D EC 95	STA TOT	
294	93D4	8D ED 95	STA TOT+1	
295	93D7	8D EB 95	STA FLAG	
296	93DA	C8	INX	
297	93DB	D0 02	BNE ,+4	
298	93DD	E6 1B	INC SER+1	
299	93DF	B1 1A	LDA (SER),Y	
300	93E1	C9 20	CMP #"-#80	; SPACE?
301	93E3	F0 F5	BEQ NOS	; YES-IGNORE IT
302	93E5	B1 1A	STN LDA (SER),Y	
303	93E7	38	SEC	
304	93E8	E9 30	SBC #"-#80	
305	93EA	90 40	BCC DRTS	; THATS IT
306	93EC	C9 0A	CMP #8A	; OK?
307	93EE	B0 3C	BCS DRTS	; NO
308	93F0	EE EB 95	INC FLAG	
309	93F3	48	PHA	
310	93F4	0E EC 95	ASL TOT	
311	93F7	2E ED 95	ROL TOT+1	; #2
312	93FA	AE ED 95	LDX TOT+1	
313	93FD	AD EC 95	LDA TOT	
314	9400	0E EC 95	ASL TOT	
315	9403	2E ED 95	ROL TOT+1	; #4
316	9406	0E EC 95	ASL TOT	
317	9409	2E ED 95	ROL TOT+1	; #8
318	940C	6D EC 95	ADC TOT	
319	940F	8D EC 95	STA TOT	
320	9412	8A	TXA	
321	9413	6D ED 95	ADC TOT+1	; ADD #2
322	9416	8D ED 95	STA TOT+1	
323	9419	68	PLA	; VALUE
324	941A	6D EC 95	ADC TOT	
325	941D	8D EC 95	STA TOT	
326	9420	90 03	BCC NEXD	
327	9422	EE ED 95	INC TOT+1	
328	9425	C8	INX	
329	9426	D0 BD	BNE STN	
330	9428	E6 1B	INC SER+1	
331	942A	D0 B9	BNE STN	
332	942C	60	DRTS RTS	
333				
334				; A - TOKEN ON INPUT TO COUNT
335				; A - # OF CHARS ON OUTPUT
336				
337	942D	38	CNTCHR SEC	
338	942E	E9 80	SBC #128	; 128 IS LOWEST TOKEN
339	9430	90 09	BCC NOGD	
340	9432	C9 68	CMP #ENDTAB	
341	9434	B0 05	BCS NOGD	
342	9436	AA	TAX	
343	9437	BD 3E 94	LDA CHRTAB,X	
344	943A	60	RTB	
345	943B	A9 01	NOGD LDA #1	
346	943D	60	RTS	
347				
348	943E	05 05	CHRTAB DFC 5,5	; 128-129
349	9440	06 06 07	DFC 6,6,7,5,5,6,4,6,5,5	
		05 05 06 04 06 05		
		05		
350	944A	06 06 06	DFC 6,6,6,6,6,5,9,7,6,7	
		06 06 05 09 07 06		
		07		
351	9454	06 06 06	DFC 6,6,6,8,8,7,9,8,9,7; 150-159	
		08 08 07 09 08 09		
		07		
352	945E	08 05 06	DFC 8,5,6,8,8,7,8,8,7,8	
		08 08 07 08 08 07		
		08		
353	9468	05 06 05	DFC 5,6,5,4,9,3,8,8,5,6	
		04 09 03 08 08 05		
		06		
354	9472	04 06 06	DFC 4,6,6,6,5,6,7,5,5,7	
		06 05 06 07 05 05		
		07		
355	947C	05 05 06	DFC 5,5,6,4,4,6,6,4,5,6	
		04 04 06 06 04 05		
		06		
356	9486	03 03 03	DFC 3,3,3,3,3,5,4,3,3,3	
		03 03 05 04 03 03		
		03		
357	9490	05 05 05	DFC 5,5,5,5,5,7,5,5,5,5	
		05 05 07 05 05 05		
		05		
358	949A	05 05 05	DFC 5,5,5,5,5,5,6,5,6,5	
		05 05 05 06 05 06		
		05		


```

359 94A4 05 06 07 DFC 5,6,7,8,6
      08 06
360 ENDTAB EQU .-CHRTAB
361
362
363 ; X,Y - LINE NUMBER ON INPUT
364 ; A - # OF CHARS ON OUTPUT
365
366 94A9 A9 05 LNCHR LDA #5 ;5 CHARS IF >=2710
367 94AB E0 27 CPX #10000/256
368 94AD 90 06 BCC M1 ;<2700
369 94AF D0 22 BNE DN ;>=2800
370 94B1 C0 10 CPY #10000
371 94B3 B0 1E BCS DN ;>=2710
372 94B5 A9 04 M1 LDA #4 ;4 CHARS IF >=3EB
373 94B7 E0 03 CPX #1000/256
374 94B9 90 06 BCC M2 ;<300
375 94BB D0 16 BNE DN ;>=400
376 94BD C0 EB CPY #1000
377 94BF B0 12 BCS DN ;>=3EB
378 94C1 A9 03 M2 LDA #3 ;3 CHARS IF >= 100
379 94C3 E0 00 CPX #0
380 94C5 D0 0C BNE DN ;>=100
381 94C7 C0 64 CPY #100
382 94C9 B0 08 BCS DN
383 94CB A9 02 LDA #2
384 94CD C0 0A CPY #10
385 94CF B0 02 BCS DN
386 94D1 A9 01 LDA #1
387 94D3 60 DN RTS
388
389
390 ; GET LENGTH OF PRESENT LINE AND NEXT LINE
391 ; IF THEY WERE COMBINED
392 ; LENGTH IN TOT AND TOT+1
393
394 94D4 A0 03 GETSIZ LDY #3
395 94D6 A5 FD LDA IND+1
396 94D8 48 PHA
397 94D9 A9 00 LDA #0
398 94DB 8D EC 95 STA TOT
399 94DE 8D ED 95 STA TOT+1
400 94E1 B1 FC LDA (IND),Y
401 94E3 AA TAX
402 94E4 88 DEY
403 94E5 B1 FC LDA (IND),Y
404 94E7 A8 TAY
405 94E8 20 A9 94 JSR LNCHR
406 94EB 8D EC 95 STA TOT
407 94EE A0 04 LDY #4
408 94F0 B1 FC GETSZ LDA (IND),Y
409 94F2 48 PHA
410 94F3 20 2D 94 JSR CNTCHR
411 94F6 18 CLC
412 94F7 6D EC 95 ADC TOT
413 94FA 8D EC 95 STA TOT
414 94FD 90 03 BCC SZ
415 94FF EE ED 95 INC TOT+1
416 9502 C8 SZ INY
417 9503 D0 02 BNE SZ5
418 9505 E6 FD INC IND+1
419 9507 68 SZ5 PLA
420 9508 D0 E6 BNE GETSZ
421 950A 68 PLA
422 950B 85 FD STA IND+1
423 950D A0 04 LDY #4
424 950F A5 FF LDA PIND+1
425 9511 48 PHA
426 9512 B1 FE SZ2 LDA (PIND),Y
427 9514 48 PHA
428 9515 20 2D 94 JSR CNTCHR
429 9518 18 CLC
430 9519 6D EC 95 ADC TOT
431 951C 8D EC 95 STA TOT
432 951F 90 03 BCC SZ3
433 9521 EE ED 95 INC TOT+1
434 9524 C8 SZ3 INY
435 9525 D0 02 BNE SZ4
436 9527 E6 FF INC PIND+1
437 9529 68 SZ4 PLA
438 952A D0 E6 BNE SZ2
439 952C 68 PLA
440 952D 85 FF STA PIND+1
441 952F 60 RTS
442
443 ;PRINT LINES MERGED
444
445 9530 A0 02 PRINT LDY #2
446 9532 B1 FC LDA (IND),Y
447 9534 8D E9 95 STA TLO
448 9537 C8 INY
449 9538 B1 FC LDA (IND),Y
450 953A 8D EA 95 STA THI
451 953D 20 71 95 JSR CONV ;PRINT TO NUMBER
452 9540 A2 00 LDX #0
453 9542 BD E1 95 M3 LDA ARROW,X
454 9545 F0 06 BEQ PDONE
455 9547 20 ED FD JSR DOUT ;PRINT ARROW
456 954A EB INX
457 954B D0 F5 BNE M3

```



Apple Soft Line Cruncher (Cont.)

```

458 954D A0 02 PDONE LDY #2
459 954F B1 FE LDA (PIND),Y
460 9551 BD E9 95 STA TLO
461 9554 CB INY
462 9555 B1 FE LDA (PIND),Y
463 9557 BD EA 95 STA THI
464 955A 20 71 95 JSR CONV ;PRINT FROM NUMBER
465 955D 20 BE FD JSR CR
466 9560 AD 00 C0 LDA #C000 ;KEY HIT?
467 9563 10 0B BPL NOKEY
468 9565 BD 10 C0 STA #C010
469 9568 AD 00 C0 K LDA #C000 ;WAIT FOR ANOTHER
470 956B 10 FB BPL K
471 956D BD 10 C0 STA #C010
472 9570 60 NOKEY RTS
473
474
475 ;
476 ;CONV HI LO TO ASCII IN ANSW
477
478 9571 A2 04 CONV LDX #4
479 9573 A9 30 LDA #0-#80
480 9575 9D EE 95 STA ANSW,X ;FOR ONLY 0
481 9578 CA DEX
482 9579 A9 20 LDA # -#80 ;SPACE
483 957B 9D EE 95 C2 STA ANSW,X
484 957E CA DEX
485 957F 10 FA BPL C2
486 9581 EB INX
487 9582 A0 00 LDY #0
488 9584 B9 D6 95 GONB LDA NUMS,Y
489 9587 F0 3D BEQ BDON
490 9589 38 GOON SEC
491 958A AD E9 95 LDA TLO
492 9590 F9 D6 95 SBC NUMS,Y ;OK?
493 9591 4B PHA
494 9592 CB INY
495 9595 AD EA 95 LDA THI
496 9598 F9 D6 95 SBC NUMS,Y ;NO
497 959A 90 27 BCC NEXA
498 959D BD EA 95 STA THI
499 959E 6B PLA
500 95A1 BD E9 95 STA TLO
501 95A2 8B DEY
502 95A5 BD EE 95 LDA ANSW,X
503 95A7 C9 20 CMP # -#80 ;SPACE?
504 95A9 D0 13 BNE INCT ;NO
505 95AB A9 30 LDA #0-#80
506 95AD 9D EE 95 STA ANSW,X
507 95AF 8A TXA
508 95B0 4B PHA ;SAVE X
509 95B2 A9 30 LDA #0-#80
510 95B3 EB INX
511 95B5 9D EE 95 STA ANSW,X ;O NOW
512 95B8 E0 04 CPX #4
513 95BA 90 F8 BCC LP2
514 95BB 6B PLA
515 95BC AA TAX ;X BACK
516 95BF FE EE 95 INCT INC ANSW,X ;COUNT IT
517 95C1 D0 C8 BNE GOON
518 95C2 CB INY
519 95C3 6B PLA
520 95C4 EB INX
521 95C6 D0 BE BNE GONB
522 95C8 A2 00 BDON LDX #0
523 95CB BD EE 95 HANBW LDA ANSW,X
524 95CD 09 80 ORA #80
525 95D0 20 ED FD JSR DOUT
526 95D1 EB INX
527 95D3 E0 05 CPX #5
528 95D5 90 F3 BCC HANBW
529 60 RTS
530
531
532 ;
533 ;10000 1000 100 10 1
534 95D6 10 27 EB NUMS DFC #10,#27,#EB,#3,#64,0,#A,0,1,0,0
535 03 64 00 0A 00 01
536 00 00
537
538 ;
539 ;ARROW ABC " <-- "
540 95E1 A0 BC AD ARROW
541 AD A0
542 00 DFC 0
543
544 ;
545 ;D8C .
546 INDBAV DFB 1
547 TEMP DFB 1
548 TLO DFB 1
549 THI DFB 1
550 FLAG DFB 1
551 TOT DFB 2
552 ANSW DFB 5
553
554

```



TRAC-I THE INCOME SYSTEM

by Chris Exner
178 Wakefield St.
Reading, MA 01867

INTRODUCTION

TRAC-I is an easy-to-use, adaptable system for keeping track of your income. Ideal for personal accounting, INCOME-I has four different reports for examining the various sources of your revenue. (If you are using the TRAC-Plus accounting system for collecting expense data, INCOME TRAC will tap that information and produce a profit and loss statement!)

Working with TRAC-I is simple — TRAC-I is user-friendly with descriptive menus and many screen prompts. You name the categories of your income and then, no more than once a month, enter in your paycheck stubs and interest memos (or whatever type of cash receipts you have). To see a spreadsheet of your income year-to-date or an income statement with calculated-month-averages and trend figures, all you need to do is make a simple selection from a menu.

GETTING STARTED

The first screen format which appears after you type RUN is:

** INCOME TRAC **

THIS SYSTEM ALLOWS YOU TO ENTER INCOME DATA. YOU MAY THEN PRODUCE AN INCOME STATEMENT COMPARING YOUR INCOME TO YOUR EXPENSES. THE EXPENSE DATA IS TAKEN FROM TRAC FILES 'BALANCES', 'CARD #' AND 'CHECK #'.

ENTER THE REPORT OUTPUT DEVICE YOU WISH TO USE.

- 1 = SCREEN ONLY
- 2 = PRINTER

ENTER YOUR SELECTION: 1

In this sample run of TRAC-I, we want all reports to be displayed on our monitor screen; therefore we select option one. The alternative option two would send all reports to the parallel or serial printer attached to our Apple.

TRAC-I next asks for the date which will be used in the page headings of our reports:

ENTER REPORT DATE (MO,DY,YR): 3,31,81

THE MAIN MENU

After the date has been entered, the system displays its main menu:

* MAIN MENU *

- 1 = ENTER ACCOUNT NAMES
- 2 = ADD INCOME RECORDS
- 3 = SORT INCOME RECORDS
- 4 = REPORT/LIST OPTIONS
- 5 = END

ENTER YOUR SELECTION:

Looking at the menu, we see the major functions of the TRAC-I system:

1 = ENTER ACCOUNT NAMES — You select your own income categories. Each category you identify is assigned an account number and a short account name. Examples of income categories are Salary, Interest and Dividends from stocks.

2 = ADD INCOME RECORDS — Each revenue payment which you receive is entered as an income record in the **INCOME #** file. This data becomes the basis of your **Income Statement** and **Income Profile** reports.

3 = SORT INCOME RECORDS — The income records may be sorted by account number, date or dollar amount. This sort makes Income Record listings more readable, and it is useful for analyzing your sources of income.

4 = REPORT/LIST OPTIONS — This selection from the main menu presents you with a report menu. There are four different reports available in TRAC-I.

5 = END — Exit the program.

ENTERING ACCOUNT NAMES

As this is your first time using TRAC-I, you must set up the account names. In the main menu, type "1", hit the return key and you are ready:

** ENTER ACCOUNT NAMES **

VALID EXPENSE ACCOUNT #'S ARE 1-24

VALID INCOME ACCOUNT #'S ARE 30-49

TYPE END FOR MAIN MENU

ACCOUNT NO: 45

ACCOUNT NAME: DIV- APPLE STOCK

The data entry screen for updating account names is easy to use. Just enter a valid account number and a name. If the number entered has already been assigned an account name, the screen will display the current account name on file and ask:

ANY CHANGE?(Y/N)

After you have entered all account names, type **END** in the account number field and you will be returned to the main menu:

ACCOUNT NO: END

Entering the account names is a one-time chore. Future runs of INCOME TRAC will have the account names in the ACCOUNTS file available for their use.

ADDING INCOME RECORDS

Once you have set up your account names, a selection of "2" from the main menu will display the following screen:

* ADD INCOME RECORD *

- | | |
|----------------------|---------------------|
| 30. ABC-SALARY | 40. COM-BOOK SALES |
| 31. DEF-SALARY | 41. COM-MAGAZINES |
| 32. | 42. |
| 33. | 43. |
| 34. | 44. |
| 35. INTR-1ST NATL | 45. DIV-APPLE STOCK |
| 36. C.D.-US TREASURY | 46. |
| 37. | 47. DIV-IBM STOCK |
| 38. | 48. |
| 39. | 49. |

INCOME ACCT #'S ARE 30-49

TYPE ACCOUNT #, 'REDO', 'EDIT', OR 'END'

REC #: 1

ACCOUNT #:

MONTH/DAY:

REFERENCE

AMOUNT:

The accounts on the upper portion of the screen are those which you set up through the "ENTER ACCOUNT NAMES" screen. In the lower portion of the "ADD INCOME RECORDS" screen, you will notice that the **entries for account # and other fields do not all appear at one time. They appear as prompts only when the Apple requests data for each one of them.** After a full income record is entered, the lower screen is erased, the rec # is increased by one, and the sequence is repeated.

If you enter the word **REDO** in the account # field, the rec # is decreased by one and you will be allowed to re-enter the last income record.

The word **EDIT** in the account # field allows you to enter the record # of any income record which you would like to change.

When you have finished entering data, type **END** in the account # field. INCOME TRAC will write your income records to the text file **INCOME #** and return you to the main menu.

Selection "3" from the main menu gives the sort option screen:

* ENTER SORT PARAMETERS *

WHICH FIELD DO YOU WANT TO SORT ON?

- 1 = ACCOUNT NUMBER
- 2 = MONTH
- 3 = DOLLAR AMOUNT

ENTER YOUR SELECTION: 1

You must enter the field upon which you would like your file of income records sorted. Here we chose the account number field. After the income records are sorted INCOME TRAC will ask two questions.

The first question is:

SAVE SORTED LIST ON THE DISK?

ENTER 'Y' OR 'N':

The income data has already been saved to disk when you typed 'END' in the account # field of the **"* ADD INCOME RECORDS *"** screen. Saving the sorted file means that future listings of the income records will be in the chosen sort order — by account number, by month or by dollar amount.

continued on next page

TRAC "I" — The Income System (Cont.)

Question two is:

PRINT SORTED LIST ON PRINTER?
ENTER 'Y' OR 'N':

Answering 'Y' will allow you to obtain a listing of your income records without going through the report/list option menu.

THE REPORTS

The report/list option menu is reached by entering "4" from the main menu:

* REPORT OPTIONS *

- 1 = LIST ACCOUNTS
 - 2 = LIST INCOME RECORDS
 - 3 = INCOME STATEMENT
 - 4 = YTD INCOME PROFILE
- ENTER YOUR SELECTION:**

Just select the report you wish to see.

The "Income Statement" report gives current month and year-to-date figures, and therefore needs to know which month to use as the current month. After you select the "Income Statement" report you will be asked:

%AVG COLUMN IS (CURR MO-AVG MO)/AVG MO
YOUR FISCAL YEAR BEGINS WITH JAN
ENTER THE CURRENT MONTH #: 3

Enter a month number (here we used March), and the "Income Statement" report will be printed.

After any report is printed, the monitor screen returns to the main menu. At that time, you may end your INCOME TRAC session by selecting "5" from the main menu and thereby exit the program.

THE ACCOUNTS

Account names are stored in a text file called, appropriately enough, ACCOUNTS. The **List Account Names report** is a simple printout of the account numbers and names from the file ACCOUNTS. Having an account name file allows you to enter only the simple two-digit account code for your income items and yet have the more readable account names printed on subsequent reports.

ACCOUNTS is an APPLE DOS random-access file. Each data record of the file has an **account name** as **data** and an **account number** as the **random-access key**. For example, if account number 30 is the account category "ABC-SALARY", then "ABC-SALARY" is the contents of record number 30 of the text file ACCOUNTS.

Valid account numbers for revenue accounts are in the range of thirty to forty-nine. Expense account numbers must be from one to twenty-four. Expense accounts are used if you record expense information through the TRAC expense-gathering system. (ED NOTE: The ACCOUNTS file must be generated by the TRAC-B program (NIBBLE, VOL. 2, NO. 8).

SAMPLE

* LIST ACCOUNTS * 3/31/81

ACCT #	ACCT NAME
1	CAR EXPENSE
2	BOOKS/MAGAZINES
4	CLOTHES
7	ELECTRICITY
9	FOOD
10	GIFTS
18	MORTGAGE
22	TAXES-FEDERAL
30	ABC-SALARY
31	DEF-SALARY
32	INTR-1ST NAT'L
36	CD-US TREASURY
40	COM-BOOK SALES
41	COM-MAGAZINES
45	DIV-APPLE STOCK
47	DIV-IBM STOCK

INCOME RECORDS

You may use the TRAC-I system to keep track of your dollars as you earn them. For this, TRAC-I needs to be fed the details of your cash inflow, i.e. your paychecks, dividends, garage sale receipts and gifts from rich uncles. This detail information is then recorded in the text file INCOME # and from there is summarized into your financial statements.

If you glance at the sample ***Income Records*** listing below, you will see that the fields of the income records are: a **record number**, a **reference field** for jotting short notes, the **account number** of the category to which the item belongs, a **date** and the revenue **dollar amount**. The record # is assigned automatically by INCOME TRAC and corresponds to that item's record number in the text file INCOME #. The latter four fields contain information that you must tell INCOME TRAC about your revenue items.

Most people will find they do not need to enter income data more than once a month. It is a good idea to jot cash memos on scraps of paper and throw them into a "revenue basket" until you are ready for an INCOME TRAC-I session. A glance at your checkbook and saving book deposits will tickle your memory for any forgotten entries. Then when you are curious as to how your financial picture looks — sit down, run TRAC-I, enter, sort and list the revenue items and run the financial reports.

A word on the "sort" step — it's optional. You may leave your income records unsorted, which is to say in the order in which they were entered. **Sorting rearranges the income records into ascending order by account number, by date or by dollar amount.** You just might find that listing your income records in any one of these orders will make the ***Income Records*** listing a little more readable.

* INCOME RECORDS * 3/31/81

#	* REFERENCE *	A#	MM/DD	AMT
1	20 SHARES	45	1/22	10
2	PAY DECEMBER	31	1/4	500
3	PAY DECEMBER	30	1/3	1000
4	PAY - JANUARY	30	2/3	1000
5	HOW TO BE GOOD	40	2/15	200
6	PAY JANUARY	31	2/15	490
7	14.333 PERCENT	36	2/28	600
8	PAY FEBRUARY	31	3/15	490
9	PAY FEBRUARY	30	3/3	1050
10	GLAMOUR MAG.	41	3/26	60
11	HOW TO BE GOOD	40	3/3	125
12	100 SHARES	47	3/20	25

* TOTALS * 5550

THE INCOME STATEMENT

In the classic accounting definition, an income statement is a report of revenue and expenses. If revenues exceed expenses, then the difference between the two is called **net profit**. If expenses exceed revenues, the difference is called **net loss**. The purpose of an income statement is to report changes in one's assets. For that period of time which the income statement covers, one's assets are either increased by a net profit or else decreased by a net loss.

The Income Statement report of TRAC-I can be used in one of two ways. If the (income-gathering) TRAC-I is used in conjunction with the (expense-gathering) TRAC-B system, then the Income Statement report has available to it both revenue and expense data and can produce profit/loss figures. If TRAC-I is used alone, the "Income Statement" report will contain only revenue data, and while a profit/loss total cannot be calculated, the statement functions very well as an **income analysis report**.

There are four columns of figures in the Income Statement report. Column one shows the current month totals. Column **AVG** is the averages of the months from the beginning of the fiscal year through the current month. Column **%AVG** is the deviation of the current month amounts from the average month amount, expressed as percentages of the month averages. An example of the %AVG calculation: \$125 in the current month column is a 25% %AVG increase over a month average of \$100. The **YTD** column is the cumulative totals from the beginning of the fiscal year through the current month.

THE FISCAL YEAR

What is a fiscal year? For most people, the fiscal year is the calendar year, from January to December. For others, their natural business cycle is such that the beginning of their accounting year is better started on a month other than January. For example, if you are a teacher and view your business year as the school year, your natural fiscal year might begin in September. It is easy to change the fiscal year set-up of INCOME TRAC. (See the Operating Notes of this article for the "how-to's.") In fact, if you begin using your INCOME TRAC system in the middle of the calendar year, it is a good idea to temporarily change the first month of your fiscal year to the month in which you begin using the system. This change will give the **AVG** and **%AVG** columns of the Income Statement report more meaning. When January rolls around again, you can switch the fiscal year back to the calendar year.

If you are using the **TRAC-Plus** system to collect your expense information, then you have both the revenue and the expense side of the Income Statement report. (TRAC-Plus was presented in Nibble Vol. 2 #8). If you have both types of data you will also get profit and loss figures. When you are reviewing these figures, the **matching principle** of accounting should be kept in mind. This principle assumes that revenues and expenses are apportioned among the months they affect. In other words, even though you paid your annual insurance premium as a lump sum in May, it should, ideally, be reported in twelve increments from May through next April. The same holds true for revenue items such as certifi-

TRAC "I" — The Income System (Cont.)

cate interest payments — a portion of that revenue should be reported in each of the months during which you hold the certificate. It is with the matching principle that the truest profit and loss numbers are created. Businesses employ hordes of accountants to keep track of those multi-month entries, but for a personal accounting system such entries are a lot of trouble. My suggestion for entering revenue and expense items is to "put 'em in as you get 'em" — but when you look at your monthly Income Statement report, stand back and keep the year-to-date view. This month's "net loss" could reflect the fact that your taxes are due, and not that you are becoming insolvent.

** INCOME STATEMENT ** 3/31/81				
	MAR	AVG	%AVG	YTD
## INCOME ##				
30 ABC-SA	1050	1017	3	3050
31 DEF-SA	490	493	-1	1480
32 INTR-1	0	0	0	0
36 CD-US	0	200	-100	600
40 COM-BO	125	108	16	325
41 COM-MA	60	20	200	60
45 DIV-AP	0	3	-100	10
47 DIV-IB	25	8	213	25
* GROSS * =====				
* INCOME *	1750	1849	-6	5550
## EXPENSES ##				
1 CAR EX	30	52	-42	155
2 BOOKS/	0	0	0	0
4 CLOTHE	100	90	11	270
7 ELECTR	0	17	-100	50
9 FOOD	210	197	7	590
10 GIFTS	0	17	-100	50
18 MORTGA	820	820	0	2460
22 TAXES-	0	467	-100	1400
* TOTAL * =====				
* EXPENSES *	1160	1660	-31	4975
* NET * =====				
* INCOME *	590	189	212	575

THE YEAR-TO-DATE INCOME PROFILE

The "Income Profile" report is a "spread" of monthly income from January to December. As the entire year is shown on one page, you can see in a glance all your income categories and note which categories contribute the most to your total revenue. "Income Profile" is a straightforward report useful in spotting cash flow problems and recognizing income trends.

YEAR-TO-DATE INCOME PROFILE 3/31/81				
INCOME DESCR'	JAN	FEB	MAR	DEC
=====				
ABC-SALARY	1000	1000	1050	
DEF-SALARY	500	490	490	
INTR-1ST NAT'L				
CD-US TREASURY		600		
COM-BOOK SALES		200	125	
COM-MAGAZINES			60	
DIV-APPLE STOCK	10			
DIV-IBM STOCK			25	
=====				
** TOTALS **	1510	2290	1750	0

OPERATING NOTES

1) To have your fiscal year start on a month other than January change line 9070 from "BMO=1" to "BMO=<your month #>"

2) Printers and the control characters they require can vary. The "Year-To-Date Income Profile" report is 130 characters wide and therefore requires condensed

printing if you are using 8 1/2 inch wide paper. Change lines 11030 and 11340 for your printer's condensed printing control characters.

LIST

```

10 REM *****
15 REM ** TRAC "I" **
20 REM ** THE INCOME SYSTEM **
25 REM ** BY CHRIS EXNER **
30 REM ** COPYRIGHT (C) 1982 **
35 REM ** BY MICRO-SPARC INC **
40 REM ** LINCOLN, MA. 01773 **
45 REM *****
72 GOTO 100
75 REM REQUIRES APPLESOFT AND 1 DISK. PARALLEL OR SERIAL
  L PRINTER IS OPTIONAL
80 REM ' ' PRINTING
81 PRINT SPC( PK - SY)V%
82 SY = SY + (PK - SY) + LEN (V%)
83 RETURN
85 REM END OF LINE PRINTING
86 PRINT SPC( PK - SY)V%
87 PK = 0:SY = 0
88 RETURN
100 TEXT : HOME : HTAB 11: PRINT "** INCOME TRAC **
110 PRINT : PRINT " THIS SYSTEM ALLOWS YOU TO ENTER"
120 PRINT "INCOME DATA. YOU MAY THEN PRODUCE AN "
130 PRINT "INCOME STATEMENT COMPARING YOUR INCOME"
140 PRINT "TO YOUR EXPENSES. THE EXPENSE DATA IS "
150 PRINT "TAKEN FROM TRAC FILES 'BALANCES',"
160 PRINT "'CARD #' AND 'CHECK #'."
175 REM * ACCEPT OUTPUT DEVICE
180 VTAB 12: PRINT "ENTER THE REPORT OUTPUT DEVICE": PRINT
  "YOU WISH TO USE."
190 HTAB 10: PRINT "1 = SCREEN ONLY"
195 HTAB 10: PRINT "2 = PRINTER"
200 VTAB 17: INVERSE : INPUT "ENTER YOUR SELECTION: ";PRT
  YPE: NORMAL : GOSUB 2300
205 IF PRTYPE < > 1 AND PRTYPE < > 2 THEN GOSUB 2100: GOTO
  200
210 IF PRTYPE = 1 THEN 220
215 HOME : VTAB 10: PRINT "ENTER PRINTER TYPE:": PRINT : PRINT
  "1= IDS & CENTRONICS COMPATIBLE": PRINT "2= CENTRONIC
  S 737": PRINT "3= EPSON MX-80": PRINT : INPUT "ENTER
  SELECTION: ";PB%: IF VAL (PB%) < 0 OR VAL (PB%) > 3
  THEN 215
220 REM * ACCEPT REPORT DATE
230 VTAB 20: INVERSE : INPUT "ENTER REPORT DATE (MO, DAY, Y
  R):";RMO,RDY,RYR: NORMAL
240 IF (RMO < 1 OR RMO > 12) OR RDY > 31 THEN GOSUB 2100
  : GOTO 230
300 D% = CHR% (4): REM CTRL-D
310 DIM ACCT%(49),BAL(49,12)
330 DIM MNAME%(12),W%(200,6)
340 DIM WK(300,5),YTD(49),AVG(49),PRCT(49)
500 ONERR GOTO 5060
1000 REM *** MAIN MENU ***
1010 HOME : HTAB 13: PRINT "* MAIN MENU *"
1020 VTAB 4: PRINT "1 = ENTER ACCOUNT NAMES"
1030 PRINT "2 = ADD INCOME RECORDS"
1040 PRINT "3 = SORT INCOME RECORDS "
1050 PRINT "4 = REPORT/LIST OPTIONS"
1055 PRINT "5 = END"
1060 VTAB 11: INVERSE : INPUT "ENTER YOUR SELECTION: ";OM
  : NORMAL
1070 IF OM = 1 THEN GOTO 3000
1080 IF OM = 2 THEN GOTO 5000
1090 IF OM = 3 THEN GOTO 7000
1100 IF OM = 4 THEN GOTO 9000
1110 IF OM = 5 THEN END
1120 GOSUB 2100: GOTO 1060
2000 REM ** GENERAL PURPOSE ROUTINES
2100 VTAB 23: PRINT "INVALID ENTRY... PLEASE REENTER": RETURN
2300 VTAB 23: CALL - 868: RETURN
2400 Q = P:P = ABS (P)
2410 C = (P > = 10) + (P > = 100) + (P > = 1000) + (P >
  = 10000)
2420 IF Q < 0 THEN C = C + 1
2430 RETURN
2500 REM *ADD DETAIL FILE
2501 F% = "INCOME #"
2505 ONERR GOTO 2560
2510 FLASH : PRINT "READING ";F%:" FILE": NORMAL
2515 PRINT D% "OPEN ";F%:";L40"
2520 PRINT D% "READ ";F%:";R0"
2525 INPUT R
2530 FOR X = 1 TO R
2535 PRINT D% "READ ";F%:";R";X
2540 INPUT A,CNO%,MO%,DY%,AMT
2542 MO = VAL (MO%)
2545 BAL(A,MO) = BAL(A,MO) + AMT
2550 NEXT X
2555 GOTO 2570
2560 REM * DETAIL FILE ERR RTN
2565 ER = PEEK (222): POKE 216,0: IF ER < > 5 THEN GOTO
  2900
2570 PRINT D% "CLOSE ";F%

```

continued on next page

TRAC "I" — The Income System (Cont.)

```

2575 IF SW = 1 THEN SW = 0: GOTO 10040: REM RETURN TO PR
      OFILE
2580 IF F# = "INCOME #" THEN F# = "CHECK #": GOTO 2505
2590 IF F# = "CHECK #" THEN F# = "CARD #": GOTO 2505
2595 GOTO 9380: REM RETURN TO INCOME STATEMENT
2800 VTAB 24: INVERSE : INPUT "HIT RETURN FOR NEXT PAGE";
      X$: NORMAL : HOME : PGCT = 0
2810 RETURN
2850 POKE 36,8: PRINT "** INCOME STATEMENT ** ";RMD;"/";
      RDV;"/";RYR
2860 PRINT : POKE 36,12: PRINT MNAME$(CMO);"      AVG  %A
      VG  YTD": PRINT : PGCT = 4: RETURN
2900 PRINT "CHECK APPLESOFT MANUAL PG.136 FOR ERR."
2910 PRINT "THE ERROR IS #"; PEEK (222);"LOCATED IN LINE#
      "; PEEK (218) + PEEK (219) * 256
2920 END
3000 REM ***** LOAD ACCOUNT TABLE ROUTINE
3010 ONERR GOTO 3900
3020 FLASH : PRINT "LOAD ACCOUNTS " : NORMAL : X = 1
3030 PRINT D$"OPEN ACCOUNTS,L30"
3040 PRINT D$"READ ACCOUNTS,R0"
3050 INPUT R
3060 FOR X = 1 TO 49
3070 PRINT D$"READ ACCOUNTS,R";X
3080 INPUT ACCT$(X)
3090 NEXT X
3100 PRINT D$"CLOSE ACCOUNTS"
3110 IF SW = 1 THEN SW = 0: GOTO 5005: REM ADD INCOME IN
      COME RCDS
3120 IF SW = 2 THEN SW = 0: GOTO 9110: REM LIST ACCTS
3130 IF SW = 3 THEN SW = 0: GOTO 9385: REM INCOME ST.
3140 IF SW = 4 THEN SW = 0: GOTO 10050: REM PROFILE RPT
3500 REM * ENTER ACCOUNT NAME
3510 HOME : HTAB 8: PRINT "** ENTER ACCOUNT NAMES **"
3520 VTAB 4: INVERSE : PRINT "VALID EXPENSE ACCOUNT #'S A
      RE 1-24 " : PRINT "VALID INCOME ACCOUNT #'S ARE 30-49"
3525 PRINT "TYPE 'END' FOR MAIN MENU": NORMAL
3530 VTAB 9: CALL - 868: INPUT "ACCOUNT NO.":A$: GOSUB 2
      300
3540 IF A$ = "END" THEN GOTO 3700
3550 A = INT ( VAL (A$))
3560 IF A < 1 OR (A > 24 AND A < 30) OR A > 49 THEN GOSUB
      2100: GOTO 3530
3565 IF ACCT$(A) = "" THEN GOTO 3610
3570 VTAB 11: CALL - 868: PRINT "ACCOUNT NAME: ";ACCT$(A
      )
3580 VTAB 23: INVERSE : INPUT "ANY CHANGE? (Y/N) ";X$: NORMAL
3590 IF X$ = "N" THEN GOTO 3530
3600 IF X$ < > "Y" THEN GOSUB 2100: GOTO 3580
3610 VTAB 11: CALL - 868: INPUT "ACCOUNT NAME: ";ACCT$(A
      )
3620 IF LEN (ACCT$(A)) < 1 OR LEN (ACCT$(A)) > 16 THEN
      VTAB 23: PRINT "NAME LENGTH MUST BE 1-16": GOTO 3610
3630 GOTO 3530
3700 REM * WRITE OUT ACCOUNTS
3710 ONERR GOTO 2900
3715 FLASH : PRINT "WRITING ACCOUNTS FILE": NORMAL
3720 PRINT D$"OPEN ACCOUNTS,L30"
3730 PRINT D$"WRITE ACCOUNTS,R0"
3740 PRINT R
3750 FOR X = 1 TO 49
3760 PRINT D$"WRITE ACCOUNTS,R";X
3770 PRINT ACCT$(X): NEXT X
3780 PRINT D$"CLOSE ACCOUNTS"
3790 GOTO 1000: REM RETURN TO MAIN MENU
3900 REM * ACCOUNTS ERROR RTN
3910 ER = PEEK (222): POKE 216,0: IF ER < > 5 THEN 2900
3920 PRINT D$"CLOSE ACCOUNTS"
3930 FOR Y = X TO 49:ACCT$(Y) = "": NEXT Y
3940 R = 49
3950 GOTO 3110
5000 REM ***** ENTER INCOME RECORDS
5002 SW = 1: GOTO 3000: REM LOAD ACCT NAMES
5005 ONERR GOTO 5060
5030 PRINT D$"OPEN INCOME #,L40"
5040 PRINT D$"READ INCOME #,R0"
5050 INPUT R: GOTO 5080
5060 ER = PEEK (222): POKE 216,0: IF ER < > 5 THEN GOTO
      2900
5070 R = 0
5080 PRINT D$"CLOSE INCOME #"
5095 I = R
5100 HOME : HTAB 9: PRINT "* ADD INCOME RECORD *"
5110 PRINT : FOR X = 30 TO 39: PRINT X;". ";ACCT$(X): HTAB
      20: PRINT X + 10;". ";ACCT$(X + 10): NEXT X
5130 PRINT : INVERSE : PRINT "INCOME ACCT #'S ARE 30-49":
      PRINT "TYPE ACCOUNT #, 'REDO', 'EDIT', OR 'END': NORMAL
5140 I = I + 1
5150 VTAB 18: CALL - 868: PRINT "REC #";I
5160 VTAB 19: CALL - 868: INVERSE : INPUT "ACCOUNT # ";W
      $(I,1)
5170 GOSUB 2300
5180 IF W$(I,1) = "END" THEN NORMAL : I = I - 1: GOTO 540
      0
5190 IF W$(I,1) = "REDO" THEN I = I - 1: GOTO 5150
5200 IF W$(I,1) = "EDIT" THEN IS = I: INPUT "RECORD # TO
      EDIT? ";I:W$(IS,1) = "": GOTO 5150

```

```

5210 W = VAL (W$(I,1)): IF W < 30 OR W > 50 THEN GOSUB 2
      100: GOTO 5160
5220 W$(I,2) = "0": REM FOR CONSISTANCY W/ OTHER TRAC DAT
      A FILES, THE CARD/CHECK FIELD IS PLACE-HELD.
5230 VTAB 20: CALL - 868: INPUT "MONTH,DAY: ";W$(I,3),W$(
      I,4)
5240 GOSUB 2300
5250 WM = VAL (W$(I,3)):WD = VAL (W$(I,4)): IF WM < 1 OR
      WM > 12 OR WD > 31 THEN GOSUB 2100: GOTO 5230
5260 VTAB 21: CALL - 868: PRINT "REFERENCE: "; NORMAL :
      PRINT "-----": HTAB 12: INVERSE : INPUT ""
      ;W$(I,6)
5270 GOSUB 2300
5280 IF W$(I,6) = "" THEN W$(I,6) = " "
5290 IF LEN (W$(I,6)) < 1 OR LEN (W$(I,6)) > 14 THEN VTAB
      24: PRINT "MAX 14 CHAR ALLOWED... PLEASE REENTER": GOTO
      5260
5300 VTAB 22: CALL - 868: INPUT "AMOUNT: ";W$(I,5)
5310 GOSUB 2300
5320 IF ASC (W$(I,5)) < 48 OR ASC (W$(I,5)) > 57 THEN VTAB
      24: PRINT "NOT A NUMBER...PLEASE REENTER": GOTO 5300
5330 VTAB 17: CALL - 958: PRINT "LAST ACCT=";W$(I,1);"
      MONTH=";W$(I,3);" AMT=";W$(I,5)
5340 IF IS > 0 THEN X = I: GOSUB 5400: I = IS: IS = 0: GOTO
      5150
5380 GOTO 5140
5400 REM * WRITE INCOME RECORDS
5410 PRINT D$"OPEN INCOME #,L40"
5420 IF IS > 0 THEN 5460: REM DIRECT EDIT
5430 PRINT D$"WRITE INCOME #,R0"
5440 PRINT I
5450 FOR X = R + 1 TO I
5460 PRINT D$"WRITE INCOME #,R";X
5470 PRINT W$(X,1): PRINT W$(X,2): PRINT W$(X,3): PRINT W
      $(X,4): PRINT W$(X,5): PRINT W$(X,6)
5880 IF IS > 0 THEN 5900
5890 NEXT X
5900 PRINT D$"CLOSE INCOME #"
5910 IF IS > 0 THEN RETURN
5915 IF SW = 2 THEN SW = 0: GOTO 7440: REM RETURN TO SOR
      T RTN
5920 GOTO 1000: REM RETURN TO MAIN MENU
7000 REM ***** SORT INCOME RECORDS
7010 REM * READ INCOME FILE RTN
7015 R = 0: ONERR GOTO 7104
7020 FLASH : PRINT "READING INCOME #": NORMAL
7030 PRINT D$"OPEN INCOME #,L40"
7040 PRINT D$"READ INCOME #,R0"
7050 INPUT R
7060 IF R = 0 THEN GOTO 7106
7070 FOR X = 1 TO R
7080 PRINT D$"READ INCOME #,R";X
7090 INPUT W$(X,1),W$(X,2),W$(X,3),W$(X,4),W$(X,5),W$(X,6
      )
7100 NEXT X
7102 GOTO 7110
7104 ER = PEEK (222): POKE 216,0: IF ER < > 5 OR R < >
      0 THEN GOTO 2900
7106 INVERSE : PRINT "NO RECORDS STORED IN INCOME# FILE":
      NORMAL : FOR X = 1 TO 1500: NEXT X: GOTO 1000
7110 PRINT D$"CLOSE INCOME #"
7120 IF SW = 1 THEN SW = 0: GOTO 9820
7200 REM * ENTER SORT PARAMETERS
7205 HOME : HTAB 7: PRINT "* ENTER SORT PARAMETERS *"
7210 PRINT : PRINT "WHICH FIELD DO YOU WANT TO SORT ON?"
7220 HTAB 10: PRINT "1 = ACCOUNT NUMBER"
7230 HTAB 10: PRINT "3 = MONTH"
7240 HTAB 10: PRINT "5 = DOLLAR AMOUNT"
7250 VTAB 7: INVERSE : INPUT "ENTER YOUR SELECTION: ";SR:
      NORMAL
7260 IF SR = 1 THEN A$ = "BY ACCOUNT NUMBER"
7270 IF SR = 3 THEN A$ = "BY MONTH"
7275 IF SR = 5 THEN A$ = "BY DOLLAR AMOUNT"
7280 IF SR < > 1 AND SR < > 3 AND SR < > 5 THEN GOSUB
      2100: GOTO 7250
7285 FLASH : PRINT "INITIALIZING SORT VARIABLES": NORMAL
7290 H = 0: K = 0: J = 0: V = 0: CM = 0: F = 0
7295 FOR X = R + 1 TO 300: FOR N = 1 TO 5:WK(X,N) = 0: NEXT
      N,X
7300 FOR X = 1 TO R: FOR N = 1 TO 5:WK(X,N) = VAL (W$(X,
      N)): NEXT N,X: REM CONVERT STRINGS TO DIGITS
7305 FLASH : PRINT "SORT IN PROGRESS": NORMAL
7310 M = R: REM ** SORT ROUTINE
7312 M = INT (M / 2): IF M = 0 THEN 7395
7320 J = 1: K = R - M
7325 H = J
7330 V = H + M: CM = CM + 1
7340 IF WK(H,SR) < WK(V,SR) THEN 7387
7350 FOR F = 1 TO 5
7355 TK = WK(H,F):WK(H,F) = WK(V,F):WK(V,F) = TK
7370 NEXT F
7372 T$ = W$(H,6):W$(H,6) = W$(V,6):W$(V,6) = T$
7375 H = H - M
7380 IF H < 1 THEN 7387
7385 GOTO 7330
7387 J = J + 1
7389 IF J > K THEN 7312
7391 GOTO 7325
7395 FLASH : PRINT "SORT COMPLETED": NORMAL
7400 FOR X = 1 TO R: FOR N = 1 TO 5:W$(X,N) = STR$(WK(X
      ,N)): NEXT N,X: I = R

```


TRAC-I the Income System (Cont.)

```

7410 I = R:R = 0:IS = 0: VTAB 10: PRINT "SAVE SORTED LIST
ON THE DISK? "; INVERSE : INPUT "ENTER 'Y' OR 'N'":
X$: NORMAL : GOSUB 2300
7420 IF X$ = "Y" THEN SW = 2: GOTO 5400: REM WRITE INCOM
E RCDS
7430 IF X$ < > "Y" AND X$ < > "N" THEN GOSUB 2100: GOTO
7410
7440 R = 1: VTAB 13: PRINT "PRINT SORTED LIST ON PRINTER?
"; INVERSE : INPUT "ENTER 'Y' OR 'N'": X$: NORMAL : GOSUB
2300
7450 IF X$ = "Y" THEN GOTO 9820
7460 IF X$ < > "Y" AND X$ < > "N" THEN GOSUB 2100: GOTO
7440
7470 GOTO 1000: REM MAIN MENU
9000 REM ***** REPORT OPTIONS
9010 HOME : HTAB 10: PRINT "## REPORT OPTIONS ##"
9020 PRINT : PRINT "1 = LIST ACCOUNTS"
9025 PRINT "2 = LIST INCOME RECORDS"
9030 PRINT "3 = INCOME STATEMENT"
9040 PRINT "4 = YTD INCOME PROFILE"
9050 VTAB 8: INVERSE : INPUT "ENTER SELECTION: "; RP: NORMAL
: GOSUB 2300
9055 IF RP = 1 THEN GOTO 9100
9060 IF RP = 2 THEN GOTO 9800
9065 IF RP < > 3 AND RP < > 4 THEN GOSUB 2100: GOTO 90
50
9067 RESTORE : FOR X = 1 TO 12: READ MNAME(X): NEXT X
9068 IF RP = 4 THEN GOTO 10000: REM PROFILE RPT
9069 VTAB 9: PRINT "XAVG COLUMN IS (CURRMO - AVG MO) / AVG M
O"
9070 BMO = 1: REM CHANGE THIS LINE TO FIT YOUR FISCAL YEA
R SET-UP
9075 VTAB 11: PRINT "YOUR FISCAL YEAR BEGINS WITH "MNAME
$(BMO)
9080 VTAB 14: INVERSE : INPUT "ENTER THE CURRENT MONTH:
"; CMO: GOSUB 2300: VTAB 16
9085 CMO = INT (CMO): IF CMO < 1 OR CMO > 12 THEN GOSUB
2100: GOTO 9080
9090 GOTO 9300: REM INCOME STATE.
9100 REM ** LIST ACCOUNTS
9105 SW = 2: GOTO 3000: REM LOAD ACCT NAMES
9110 IF PRTYPE = 2 THEN GOTO 9200: REM LIST ACCOUNTS ON
PRINTER
9115 REM * LIST ACCTS ON SCREEN
9120 A = 0
9125 HOME : HTAB 11: PRINT "## LIST ACCOUNTS ##"
9130 PRINT : PRINT "ACCT# ACCT NAME"
9135 FOR X = 0 TO 39: PRINT "=": NEXT X: PRINT
9140 PGCT = 0
9145 PGCT = PGCT + 1
9150 IF PGCT > 17 THEN VTAB 24: INVERSE : INPUT "HIT RET
URN FOR NEXT PAGE "; X$: NORMAL : GOTO 9125
9155 A = A + 1
9160 IF A > 49 THEN GOTO 9175
9165 IF ACCT$(A) = "" THEN GOTO 9155: REM SKIP UNUSED A
CCTS
9170 HTAB 3: PRINT A: HTAB 9: PRINT ACCT$(A): GOTO 9145
9175 VTAB 24: INVERSE : INPUT "HIT RETURN FOR MAIN MENU":
X$: NORMAL : GOTO 1000
9200 REM ** LIST ACCTS ON PRINTER
9205 PRINT D$;"PR#1"
9210 PRINT CHR$(9);"40N"
9215 PRINT CHR$(2); CHR$(30)
9220 PRINT TAB(11); "## LIST ACCOUNTS ## "RMO: "/"RDY
: "/"RYR
9230 PRINT : PRINT "ACCT# ACCT NAME"
9235 FOR X = 1 TO 39: PRINT "=": NEXT X: PRINT
9240 FOR A = 1 TO 49
9245 IF ACCT$(A) = "" THEN GOTO 9260: REM SKIP UNUSED A
CCTS
9250 POKE 36,3: PRINT A:
9255 POKE 36,9: PRINT ACCT$(A)
9260 NEXT A
9265 PRINT CHR$(9);"40N"
9270 PRINT D$;"PR#0"
9275 TEXT : GOTO 1000: REM RETURN TO MAIN MENU
9300 REM * INCOME STATEMENT
9305 REM * GATHER ACTUAL EXPENSES RTN
9307 FOR X = 30 TO 49: FOR N = 1 TO 12: BAL(X,N) = 0: NEXT
N,X
9310 REM * READ YTD BALANCES
9315 ONERR GOTO 9355
9320 FLASH : PRINT "READING YEAR-TO-DATE FILES": NORMAL
9325 PRINT D$;"OPEN BALANCES"
9330 PRINT D$;"READ BALANCES"
9335 FOR X = 1 TO 24: FOR N = 1 TO 12
9340 INPUT BAL(X,N)
9345 NEXT N,X
9350 GOTO 9365
9355 REM YTD ERROR RTN
9360 ER = PEEK (222): POKE 216,0: IF ER < > 5 THEN 2900
9365 PRINT D$;"CLOSE BALANCES"
9370 GOTO 2500: REM ADD DETAIL FILES
9380 SW = 3: GOTO 3000: REM LOAD ACCT NAMES
9385 REM *CALC YTD AMTS
9390 FOR A = 1 TO 49
9392 YTD(A) = 0
9395 IF BMO < = CMO THEN FOR M = BMO TO CMO: YTD(A) = Y
T
D(A) + BAL(A,M): NEXT M
9400 IF BMO > CMO THEN FOR M = BMO TO 12: YTD(A) = YTD(A)
+ BAL(A,M): NEXT M: FOR M = 1 TO CMO: YTD(A) = YTD(A)
+ BAL(A,M): NEXT M

```

```

9405 NEXT A
9410 IF BMO < = CMO THEN MCT = (CMO - BMO) + 1
9415 IF BMO > CMO THEN MCT = (13 - BMO) + CMO
9420 FOR A = 1 TO 49: REM CALC AVG MONTHS & PERCENTAGES
AVG(A) = INT ((YTD(A) / MCT) + .5)
9430 PRCT(A) = 0: IF AVG(A) < > 0 THEN PRCT(A) = (BAL(A,C
MO) - AVG(A)) / AVG(A)
9435 PRCT(A) = INT (PRCT(A) * 100 + .5)
9440 NEXT A
9442 REM * PRINT INCOME STATEMENT
9445 IF PRTYPE = 1 THEN HOME : GOTO 9455: REM SKIP TURN
ING PRINTER ON
9447 PRINT D$;"PR#1"
9450 PRINT CHR$(9);"40N"
9452 PRINT CHR$(2); CHR$(30)
9455 GOSUB 2850: REM PAGE HEADING
9460 POKE 36,0: PRINT "## INCOME ##": PRINT " "; PGCT = PG
CT + 2
9465 TC = 0: TY = 0: TA = 0: REM INIT GROSS INCOME LINES
9470 FOR A = 30 TO 49: GOSUB 9600: NEXT A
9475 IC = TC: IY = TY: IA = TA: REM SAVE GROSS TOTALS
9480 TD = 0: IF IA < > 0 THEN TD = INT ((IC - IA) / IA *
100)
9485 PRINT "##GROSS##";
9490 HTAB 10: FOR X = 11 TO 39: PRINT "=": NEXT X: PRINT
9495 PK = 0: V$ = "INCOME#": GOSUB 80: GOSUB 9665: PGCT = P
GCT + 2
9500 IF PRTYPE = 1 AND PGCT > 16 THEN GOSUB 2800: GOSUB
2850
9502 PRINT " "; PRINT "## EXPENSES ##": PRINT : PGCT = PG
CT + 3
9505 TA = 0: TC = 0: TY = 0
9510 FOR A = 1 TO 24: GOSUB 9600: NEXT A
9515 EC = TC: EY = TY: EA = TA: REM SAVE TOTAL EXPENSES
9520 TD = 0: IF EA < > 0 THEN TD = INT ((EC - EA) / EA *
100)
9525 PRINT "##TOTAL##";
9530 HTAB 10: FOR X = 11 TO 39: PRINT "=": NEXT X: PRINT
" "
9540 PK = 0: V$ = "EXPENSE#": GOSUB 80: GOSUB 9665: PGCT =
PGCT + 2
9545 IF PRTYPE = 1 AND PGCT > 18 THEN GOSUB 2800: GOSUB
2850
9550 TC = IC - EC: TC = INT (TC * 100 + .5) / 100
9551 TA = IA - EA: TA = INT (TA * 100 + .5) / 100
9552 TY = IY - EY: TY = INT (TY * 100 + .5) / 100
9555 TD = 0: IF TA < > 0 THEN TD = ((TC - TA) / TA) * 100
9557 TD = INT (TD + .5)
9560 PRINT " "; PRINT "##NET##";
9565 HTAB 10: FOR X = 11 TO 39: PRINT "=": NEXT X: PRINT
9570 PK = 0: V$ = "INCOME#": GOSUB 80: GOSUB 9665
9575 HTAB 10: FOR X = 11 TO 39: PRINT "=": NEXT X: PRINT
" "
9577 IF PRTYPE = 2 THEN GOTO 9590
9580 VTAB 24: INVERSE : PRINT "ANY KEY FOR MENU, 'ESC' FO
R 1ST PAGE": GET X$: NORMAL
9582 IF ASC (X$) = 27 THEN GOTO 9442
9585 TEXT : GOTO 1000: REM MAIN MENU
9590 PRINT CHR$(9);"40N"
9592 PRINT D$;"PR#0"
9595 TEXT : GOTO 1000
9600 REM PRINT DETAIL LINES
9605 IF ACCT$(A) = "" AND YTD(A) = 0 THEN RETURN
9610 PK = 1: V$ = STR$(A): GOSUB 80
9612 PK = 4: V$ = LEFT$(ACCT$(A),6): GOSUB 80
9615 P = BAL(A,CMO): GOSUB 2400: PK = 14 - C: V$ = STR$(BA
L(A,CMO)): GOSUB 80
9620 TC = TC + BAL(A,CMO)
9625 P = AVG(A): GOSUB 2400: PK = 22 - C: V$ = STR$(AVG(A)
): GOSUB 80
9630 TA = TA + AVG(A)
9635 P = PRCT(A): GOSUB 2400: PK = 29 - C: V$ = STR$(PRCT(
A)): GOSUB 80
9640 P = YTD(A): GOSUB 2400: PK = 35 - C: V$ = STR$(YTD(A)
): GOSUB 85
9645 TY = TY + YTD(A)
9650 PGCT = PGCT + 1: IF PRTYPE = 1 AND PGCT > 20 THEN GOSUB
2800: GOSUB 2850
9655 RETURN
9665 REM *PRINT TOTAL LINES
9670 P = TC: GOSUB 2400: PK = 14 - C: V$ = STR$(TC): GOSUB
80
9675 P = TA: GOSUB 2400: PK = 22 - C: V$ = STR$(TA): GOSUB
80
9680 P = TD: GOSUB 2400: PK = 29 - C: V$ = STR$(TD): GOSUB
80
9685 P = TY: GOSUB 2400: PK = 35 - C: V$ = STR$(TY): GOSUB
85
9690 RETURN
9800 REM ** LIST INCOME RECORDS
9810 SW = 1: GOTO 7010: REM GET INCOME RCDS
9820 IF PRTYPE = 1 THEN HOME : GOTO 9860: REM SKIP TURN
ING ON PRINTER
9830 PRINT D$;"PR#1"
9840 PRINT CHR$(9);"40N"
9845 IF PB$ = "1" THEN PRINT CHR$(2); CHR$(30)
9850 IF PB$ = "2" THEN PRINT CHR$(27); CHR$(19)
9855 IF PB$ = "3" THEN PRINT CHR$(30)
9860 A = 0: TA = 0
9865 IF PRTYPE = 1 THEN HOME : GOTO 9875: REM SKIP TURN
ING ON PRINTER
9868 PRINT D$;"PR#1"

```

continued on page 187

TIPS 'N TECHNIQUES

MAMA (Move Aside Mr. Ampersand)

by Sandy Mossberg, M.D.
50 Talcott Rd
Port Chester, NY 10573

My old friend, the ampersand, is not the simple, fun-loving character he used to be. Fame and fortune have jaded him. In order to attract his attention, even I must refer to him as **Mr. &**. This situation has become intolerable. Keen competition is a sure remedy for a swelled head. **CONTROL-Y** is a bit player. **CALL WITH PARAMETERS** shows promise but requires maturation. **MAMA** (Move Aside Mr. Ampersand), a lovely lady to whom I shall introduce you, may just have the panache to set **Mr. &** straight. If she succeeds, we can all heave a sigh of relief. Who knows, a romance might even be in the offing. Only time will tell. Read on and decide for yourself.

APPLESOFT CHRGET AND CHRGET

The subroutines most fundamental to Applesoft are **CHRGET** and **CHRGOT**, which may be accessed by entering the system monitor (**CALL -151**), typing **B1L** and examining the code from **\$B1** to **\$C8**. Reference to LISTING 1 should make the following description more easy to comprehend.

Whenever Applesoft wishes to fetch a character, it either points the **TEXT POINTER (TXTPTR)** to one location below the desired character and calls **CHRGET**, or it sets **TXTPTR** to the address of the desired character and calls **CHRGOT**. The only difference between the two subroutines is that **CHRGET** increments **TXTPTR** before obtaining the character (LINES 80-100), whereas **CHRGOT** does not. What, actually, is the **TEXT POINTER**? It represents a two byte address, **\$B8** (LSB) and **\$B9** (MSB), which is an integral part of **CHRGET/CHRGOT** and functions as the operand of the critical instruction that loads the target character into the accumulator where it can be processed. LINE 120 illustrates this point and should be read as "load accumulator with **TEXT POINTER**". The operand used in this example is **\$205**, a location within the **INPUT BUFFER**; it could as easily have been **\$805**, the first character of the first line of a typical Applesoft program, or any other meaningful address.

If the target character is a colon (end-of-statement marker), the **ZERO** and **CARRY** bits (flags) of the **PROCESSOR STATUS REGISTER (\$48)** are set (LINE 140), and a return to the calling routine is executed (LINE 150). Spaces are ignored (LINES 160-170). LINES 180-210 clear the carry flag if the accumulator contains a digit (**\$30-\$39**), set the carry flag if a non-digit is present and set the zero flag if a hex zero (end-of-line marker) is encountered. LINE 220 returns control to the calling routine.

Before proceeding, please study **CHRGET/CHRGOT** well. The effort will stand you in good stead. I continue to be delighted by the simple elegance of the code.

ENTER MAMA

The power of the **ampersand** results from its ability to cause an immediate jump to

\$3F5, at which location we may place a jump to a machine language routine of our choice. If we strategically side-tracked **CHRGOT/CHRGOT**, a similar situation could be effected. Enter **MAMA** (LISTING 2)! LINES 1160-1220 place "JMP MAMA" into **CHRGOT/CHRGOT** at locations **\$BA-\$BC**. **MAMA** is now able to screen each character and token picked up by **CHRGOT/CHRGOT** and, if appropriate, to call a machine language routine. Does that sound familiar, Mr. Ampersand?

Let us examine the workings of the sample **MAMA** program presented. The **COMMAND MENU** (LINES 1270-1400) searches for four characters: (1) the **GOSUB** token, (2) **CONTROL-L**, (3) **CONTROL-Z**, and (4) the **PERCENT** sign. If none of these symbols is found, the portions of **CHRGOT/CHRGOT** which were obliterated (LINES 140-150 of LISTING 1) are duplicated in **MAMA** (LINES 1370-1380). Thus, if the accumulator holds a colon, the **ZERO** and **CARRY** flags are set and control returns to the calling routine (LINE 1400). Otherwise, **CHRGOT/CHRGOT** is re-entered at **\$BE** (LINE 1390) and flow continues as if nothing had ever happened.

If one of the above-noted characters is detected, a **MAMA** routine is called (LINES 1440-1600) via a branch or jump instruction, depending upon the location of the routine. Although branches could have been used exclusively in this **MAMA** program, jumps are illustrated in LINES 1320-1330 and LINES 1350-1360. **GOSUB** (LINES 1440-1450) increments **GHOLD** (a work location) whenever a **GOSUB** token is found, **LBRACK** (LINES 1470-1490) prints the left bracket which is normally inaccessible from the keyboard, **GOHOME** (LINES 1510-1520) homes the cursor and clears the screen, and **PRINT** (LINES 1540-1600) outputs the message, "WATCH FOR DDT!". Regardless of complexity, the same tripartite construction (LINES 1140, 1250, 1420) should be employed when writing a **MAMA** program.

USING MAMA

To install **MAMA**, assemble the code in **FIGURE 2** or enter it directly. **BSAVE MAMA, AS300,LS5B** and **BRUN MAMA**. If all is well, you should now find "JUMP \$30D" at **\$BA-\$BC**.

In immediate (command) mode, type **CONTROL-Z RETURN** and observe that a **HOME** command has been executed. **CONTROL-L** will produce a left bracket on the line following the Applesoft prompt character, and the **PERCENT** sign will print the "DDT" message.

In deferred (program) mode, we have the ability to utilize the above three commands and to count **GOSUB** tokens. **LISTING 3** presents a short program to accomplish the latter feat. LINE 10 pokes a zero into the location that holds the **GOSUB** count and LINE 40 retrieves this count, prints a message and ends the program. For each pass through the **FOR-NEXT** loop (LINE 30), a dummy **GOSUB** is executed twice. Thus, the **GOSUB** count should always equal twice the value of **N**, which is input in LINE 20. A sample run is printed beneath the program. If you would like to count other tokens or characters, simply **POKE** the desired ASCII value into location **\$30E** (782) and replace "GOSUB"

(LINE 40) with the name of the new character.

LISTING 4 shows how **MAMA** commands may be imbedded in an Applesoft program. **CONTROL-Z** and **CONTROL-L** follow the **REM** statements in LINES 10 and 40 respectively. Both are invisible in a normal listing. The visible percent sign command is seen in LINE 20. The program homes the cursor and clears the screen (LINE 10), prints "WATCH FOR DDT!" (LINE 20), prints " - ON THE APPLE)" (LINE 30) and adds the left bracket to the message (LINE 40). In LINE 30 do not forget to suppress a linefeed by using the semi-colon. Again, a sample run is appended. **LISTING 5** gives the same result as **LISTING 4**. Here, **PRINT** statements precede the imbedded commands and only one line is necessary.

Many other variations will produce the same result. In particular, note what happens when a **MAMA** command is enclosed in quotes. Be aware that coldstarting Applesoft (FP from basic, **CONTROL-B** or **E000G** from monitor) will cause **CHRGOT/CHRGOT** to be rewritten. Do not despair, however since **CALL 768** will reset the **MAMA** jump vector. Note, too, that a soft entry into Applesoft (**E003G** from monitor) and the **MAXFILES** command will leave **MAMA** unscathed. Experiment freely, construct your own **MAMA** programs and be sure to let us all know about your methods of improving and refining these concepts.

Finally, natural curiosity should stimulate you to wonder what **DDT** is and where it may be found. The only clues I will give are: (1) it is not a pesticide and (2) keep reading **NIBBLE** every issue.

REFERENCES

- Baker, R., "PET-pourri", **KILOBAUD MICRO-COMPUTING** (September, 1981), p. 13
- Sander-Cederlof, B., **APPLE ASSEMBLY LINE** (September, 1981) p. 8
- Floeter, A., "Apple Trap/Step", **NIBBLE** (Vol. 2, No. 5, 1981), p. 15

Apple Flash

by Todd Livesey
1112 S. Cuyler
Oak Park, IL 60304

Here's a high resolution graphics subroutine which can be used to create some interesting color effects. The program actually has three sections which will change the color of each dot on the screen to one of the other three colors. To understand how the program works I refer the reader to pp. 19-20 of the **Apple Reference Manual** in order to understand how the Hi-Res screens are displayed.

The three routines are at 768 (\$300), 771 (\$303), and 774 (\$306). **CALLING** one of these routines will result as follows:

CALL 768 All violet dots will turn blue, and all green dots will turn red, and vice versa.

CALL 771 All violet dots will turn red, and all green dots will turn blue, all white dots will turn black, and vice versa.

CALL 774 All violet dots will turn green, all blue dots will turn red, all white dots will turn black, and vice versa.

LISTING 1

:ASM

Copyright© 1982 by Micro SPARC Inc.

```

0010 *-----
0020 * CHRGET AND CHRGT ROUTINES
0030 *-----
0040 .OR $B1 ;ORIGIN
0050
00B8- 0060 TXTPTR .EQ $B8 ;TEXT POINTER
0070 *-----
00B1- B6 B8 0080 CHRGET INC TXTPTR ;BUMP ADDRESS OF
00B3- D0 02 0090 BNE CHRGT ; TEXT POINTER
00B5- B6 B9 0100 INC TXTPTR+1
0110 *-----
00B7- AD 05 02 0120 CHRGT LDA $0205 ;GET CHAR AT TEXT POINTER
0130 * ; (OPERAND IS TEXT POINTER!)
0140 CMP #$3A ;IF COLON (EOS) THEN SET Z AND C
0150 BCS .1 ; FLAGS AND RETURN
0160 CMP #$20 ;IF SPACE THEN DISREGARD IT
0170 BEQ CHRGT ; AND GET NEXT CHAR
0180 SEC ;THESE 4 LINES LEAVE [A] INTACT,
0190 SBC #$30 ; CLEAR C FLAG IF DIGIT,
0200 SEC ; SET C FLAG IF NONDIGIT,
0210 SBC #$D0 ; SET C AND Z FLAGS IF $00 (EOL)
00C8- 60 0220 .1 RTS ;RETURN TO CALLING ROUTINE

```

LISTING 2

:ASM

```

1000 *-----
1010 * MOVE ASIDE MR. AMPERSAND (MAMA)
1020 *-----
1030
1040 .OR $300 ;PROGRAM ORIGIN
1050
1060 CHRGT .EQ $B1 ;GET CHARACTER ROUTINE
1070 TOMAMA .EQ $BA ;CHRGT/CHRGOT -> MAMA
1080 FRMAMA .EQ $BE ;MAMA -> CHRGT/CHRGOT
1090 GHOLD .EQ $3CF ;HOLD GOSUB COUNT
1100 OUTDO .EQ $DB5C ;OUTPUT CHARACTER
1110 HOME .EQ $FC58 ;HOME CURSOR, CLEAR SCREEN
1120
1130 *-----
1140 * SET MAMA JUMP VECTOR
1150 *-----
1160 LDA $4C ;PUT 'JMP MAMA' INTO
1170 STA TOMAMA ; CHRGT/CHRGOT ROUTINES
1180 LDA $MAMA ; AT $BA-$BC
1190 STA TOMAMA+1
1200 LDA $MAMA
1210 STA TOMAMA+2
1220 RTS
1230
1240 *-----
1250 * COMMAND MENU
1260 *-----
030D- C9 B0 1270 MAMA CMP $B0 ;GOSUB TOKEN
030F- F0 1A 1280 BEQ GOSUB
0311- C9 0C 1290 CMP $0C ;CTL-L
0313- F0 1A 1300 BEQ LBRACK
0315- C9 1A 1310 CMP $1A ;CTL-Z
0317- D0 03 1320 BNE .1
0319- 4C 37 03 1330 JMP GHOME

```

CALL 768 "flips" (changes) the color bit of each byte in the screen buffer. CALL 771 does an "Exclusive-Or" (EOR) with a mask of 255 on each byte in the screen buffer. This causes all the bits in each byte to flip. CALL 774 first EORs each byte then flips the color bit. Thus the color bit is flipped twice, bringing it back to its original setting.

One nice feature of the program is that it works completely independently of any other Hi-Res routines. It will work with any Hi-Res routines you might be using (RAM or ROM based, etc.). Another plus is that it is non-destructive to your Hi-Res picture. Executing any of the routines twice will restore the original picture. The program can be used to produce explosions, colorful displays and effects, or whatever. Lastly, the program will work on either Hi-Res page. A POKE 782,64 will cause it to work on page two, while a POKE 782,32 will return it to page one.

ENTERING THE PROGRAM

To enter Apple Flash into your Apple, you can type the Hex code directly into memory beginning at Hex \$300.

First, type CALL -151 to invoke the Monitor. Then type 300:4C 38 03 4C 40 03 4C 4D

etc. from the listing. Begin entering the code into memory. For more information on how to enter hex code into your Apple, see pages 43-44 of your Apple Reference Manual.

When the program is entered, you can save it to disk with the statement:

BSAVE APPLE FLASH, AS300,L\$71

ED. Try the following program:

10 PRINT CHR\$(4);"BLOAD APPLE FLASH"
:HGR

:ASM

```

1000 * APPLE FLASH
1010 * BY TODD LIVESLEY
1020 * COPYRIGHT (C) 1982
1030 * BY MICRO-SPARC, INC
1040 * LINCOLN, MA. 01773
1045 *
1050 .OR $300
1060 JMP $0338
1070 JMP $0340
1080 JMP $0340
1090 JMP $0340
1100 JMP $0340
1110 JMP $0340
1120 JMP $0340
1130 JMP $0340
1140 JMP $0340
1150 JMP $0340
1160 JMP $0340
1170 JMP $0340
1180 JMP $0340
1190 JMP $0340
1200 JMP $0340
1210 JMP $0340
1220 JMP $0340
1230 JMP $0340
1240 JMP $0340
1250 JMP $0340
1260 JMP $0340
1270 JMP $0340
1280 JMP $0340
1290 JMP $0340
1300 JMP $0340
1310 JMP $0340
1320 JMP $0340
1330 JMP $0340
1340 JMP $0340
1350 JMP $0340
1360 JMP $0340
1370 JMP $0340
1380 JMP $0340
1390 JMP $0340
1400 JMP $0340
1410 JMP $0340
1420 JMP $0340
1430 JMP $0340
1440 JMP $0340
1450 JMP $0340
1460 JMP $0340
1470 JMP $0340
1480 JMP $0340
1490 JMP $0340
1500 JMP $0340
1510 JMP $0340
1520 JMP $0340
1530 JMP $0340
1540 JMP $0340
1550 JMP $0340
1560 JMP $0340
1570 JMP $0340
1580 JMP $0340
1590 JMP $0340
1600 JMP $0340
1610 JMP $0340
1620 JMP $0340
1630 JMP $0340
1640 JMP $0340
1650 JMP $0340
1660 JMP $0340
1670 JMP $0340
1680 JMP $0340
1690 JMP $0340
1700 JMP $0340
1710 JMP $0340
1720 JMP $0340
1730 JMP $0340
1740 JMP $0340
1750 JMP $0340
1760 JMP $0340
1770 JMP $0340
1780 JMP $0340
1790 JMP $0340
1800 JMP $0340
1810 JMP $0340
1820 JMP $0340
1830 JMP $0340
1840 JMP $0340
1850 JMP $0340
1860 JMP $0340
1870 JMP $0340
1880 JMP $0340
1890 JMP $0340
1900 JMP $0340
1910 JMP $0340
1920 JMP $0340
1930 JMP $0340
1940 JMP $0340
1950 JMP $0340
1960 JMP $0340
1970 JMP $0340
1980 JMP $0340
1990 JMP $0340
2000 JMP $0340
2010 JMP $0340
2020 JMP $0340
2030 JMP $0340
2040 JMP $0340
2050 JMP $0340
2060 JMP $0340
2070 JMP $0340
2080 JMP $0340
2090 JMP $0340
2100 JMP $0340
2110 JMP $0340
2120 JMP $0340
2130 JMP $0340
2140 JMP $0340
2150 JMP $0340
2160 JMP $0340
2170 JMP $0340
2180 JMP $0340
2190 JMP $0340
2200 JMP $0340
2210 JMP $0340
2220 JMP $0340
2230 JMP $0340
2240 JMP $0340
2250 JMP $0340
2260 JMP $0340
2270 JMP $0340
2280 JMP $0340
2290 JMP $0340
2300 JMP $0340
2310 JMP $0340
2320 JMP $0340
2330 JMP $0340
2340 JMP $0340
2350 JMP $0340
2360 JMP $0340
2370 JMP $0340
2380 JMP $0340
2390 JMP $0340
2400 JMP $0340
2410 JMP $0340
2420 JMP $0340
2430 JMP $0340
2440 JMP $0340
2450 JMP $0340
2460 JMP $0340
2470 JMP $0340
2480 JMP $0340
2490 JMP $0340
2500 JMP $0340
2510 JMP $0340
2520 JMP $0340
2530 JMP $0340
2540 JMP $0340
2550 JMP $0340
2560 JMP $0340
2570 JMP $0340
2580 JMP $0340
2590 JMP $0340
2600 JMP $0340
2610 JMP $0340
2620 JMP $0340
2630 JMP $0340
2640 JMP $0340
2650 JMP $0340
2660 JMP $0340
2670 JMP $0340
2680 JMP $0340
2690 JMP $0340
2700 JMP $0340
2710 JMP $0340
2720 JMP $0340
2730 JMP $0340
2740 JMP $0340
2750 JMP $0340
2760 JMP $0340
2770 JMP $0340
2780 JMP $0340
2790 JMP $0340
2800 JMP $0340
2810 JMP $0340
2820 JMP $0340
2830 JMP $0340
2840 JMP $0340
2850 JMP $0340
2860 JMP $0340
2870 JMP $0340
2880 JMP $0340
2890 JMP $0340
2900 JMP $0340
2910 JMP $0340
2920 JMP $0340
2930 JMP $0340
2940 JMP $0340
2950 JMP $0340
2960 JMP $0340
2970 JMP $0340
2980 JMP $0340
2990 JMP $0340
3000 JMP $0340
3010 JMP $0340
3020 JMP $0340
3030 JMP $0340
3040 JMP $0340
3050 JMP $0340
3060 JMP $0340
3070 JMP $0340
3080 JMP $0340
3090 JMP $0340
3100 JMP $0340
3110 JMP $0340
3120 JMP $0340
3130 JMP $0340
3140 JMP $0340
3150 JMP $0340
3160 JMP $0340
3170 JMP $0340
3180 JMP $0340
3190 JMP $0340
3200 JMP $0340
3210 JMP $0340
3220 JMP $0340
3230 JMP $0340
3240 JMP $0340
3250 JMP $0340
3260 JMP $0340
3270 JMP $0340
3280 JMP $0340
3290 JMP $0340
3300 JMP $0340
3310 JMP $0340
3320 JMP $0340
3330 JMP $0340
3340 JMP $0340
3350 JMP $0340
3360 JMP $0340
3370 JMP $0340
3380 JMP $0340
3390 JMP $0340
3400 JMP $0340
3410 JMP $0340
3420 JMP $0340
3430 JMP $0340
3440 JMP $0340
3450 JMP $0340
3460 JMP $0340
3470 JMP $0340
3480 JMP $0340
3490 JMP $0340
3500 JMP $0340
3510 JMP $0340
3520 JMP $0340
3530 JMP $0340
3540 JMP $0340
3550 JMP $0340
3560 JMP $0340
3570 JMP $0340
3580 JMP $0340
3590 JMP $0340
3600 JMP $0340
3610 JMP $0340
3620 JMP $0340
3630 JMP $0340
3640 JMP $0340
3650 JMP $0340
3660 JMP $0340
3670 JMP $0340
3680 JMP $0340
3690 JMP $0340
3700 JMP $0340
3710 JMP $0340
3720 JMP $0340
3730 JMP $0340
3740 JMP $0340
3750 JMP $0340
3760 JMP $0340
3770 JMP $0340
3780 JMP $0340
3790 JMP $0340
3800 JMP $0340
3810 JMP $0340
3820 JMP $0340
3830 JMP $0340
3840 JMP $0340
3850 JMP $0340
3860 JMP $0340
3870 JMP $0340
3880 JMP $0340
3890 JMP $0340
3900 JMP $0340
3910 JMP $0340
3920 JMP $0340
3930 JMP $0340
3940 JMP $0340
3950 JMP $0340
3960 JMP $0340
3970 JMP $0340
3980 JMP $0340
3990 JMP $0340
4000 JMP $0340
4010 JMP $0340
4020 JMP $0340
4030 JMP $0340
4040 JMP $0340
4050 JMP $0340
4060 JMP $0340
4070 JMP $0340
4080 JMP $0340
4090 JMP $0340
4100 JMP $0340
4110 JMP $0340
4120 JMP $0340
4130 JMP $0340
4140 JMP $0340
4150 JMP $0340
4160 JMP $0340
4170 JMP $0340
4180 JMP $0340
4190 JMP $0340
4200 JMP $0340
4210 JMP $0340
4220 JMP $0340
4230 JMP $0340
4240 JMP $0340
4250 JMP $0340
4260 JMP $0340
4270 JMP $0340
4280 JMP $0340
4290 JMP $0340
4300 JMP $0340
4310 JMP $0340
4320 JMP $0340
4330 JMP $0340
4340 JMP $0340
4350 JMP $0340
4360 JMP $0340
4370 JMP $0340
4380 JMP $0340
4390 JMP $0340
4400 JMP $0340
4410 JMP $0340
4420 JMP $0340
4430 JMP $0340
4440 JMP $0340
4450 JMP $0340
4460 JMP $0340
4470 JMP $0340
4480 JMP $0340
4490 JMP $0340
4500 JMP $0340
4510 JMP $0340
4520 JMP $0340
4530 JMP $0340
4540 JMP $0340
4550 JMP $0340
4560 JMP $0340
4570 JMP $0340
4580 JMP $0340
4590 JMP $0340
4600 JMP $0340
4610 JMP $0340
4620 JMP $0340
4630 JMP $0340
4640 JMP $0340
4650 JMP $0340
4660 JMP $0340
4670 JMP $0340
4680 JMP $0340
4690 JMP $0340
4700 JMP $0340
4710 JMP $0340
4720 JMP $0340
4730 JMP $0340
4740 JMP $0340
4750 JMP $0340
4760 JMP $0340
4770 JMP $0340
4780 JMP $0340
4790 JMP $0340
4800 JMP $0340
4810 JMP $0340
4820 JMP $0340
4830 JMP $0340
4840 JMP $0340
4850 JMP $0340
4860 JMP $0340
4870 JMP $0340
4880 JMP $0340
4890 JMP $0340
4900 JMP $0340
4910 JMP $0340
4920 JMP $0340
4930 JMP $0340
4940 JMP $0340
4950 JMP $0340
4960 JMP $0340
4970 JMP $0340
4980 JMP $0340
4990 JMP $0340
5000 JMP $0340
5010 JMP $0340
5020 JMP $0340
5030 JMP $0340
5040 JMP $0340
5050 JMP $0340
5060 JMP $0340
5070 JMP $0340
5080 JMP $0340
5090 JMP $0340
5100 JMP $0340
5110 JMP $0340
5120 JMP $0340
5130 JMP $0340
5140 JMP $0340
5150 JMP $0340
5160 JMP $0340
5170 JMP $0340
5180 JMP $0340
5190 JMP $0340
5200 JMP $0340
5210 JMP $0340
5220 JMP $0340
5230 JMP $0340
5240 JMP $0340
5250 JMP $0340
5260 JMP $0340
5270 JMP $0340
5280 JMP $0340
5290 JMP $0340
5300 JMP $0340
5310 JMP $0340
5320 JMP $0340
5330 JMP $0340
5340 JMP $0340
5350 JMP $0340
5360 JMP $0340
5370 JMP $0340
5380 JMP $0340
5390 JMP $0340
5400 JMP $0340
5410 JMP $0340
5420 JMP $0340
5430 JMP $0340
5440 JMP $0340
5450 JMP $0340
5460 JMP $0340
5470 JMP $0340
5480 JMP $0340
5490 JMP $0340
5500 JMP $0340
5510 JMP $0340
5520 JMP $0340
5530 JMP $0340
5540 JMP $0340
5550 JMP $0340
5560 JMP $0340
5570 JMP $0340
5580 JMP $0340
5590 JMP $0340
5600 JMP $0340
5610 JMP $0340
5620 JMP $0340
5630 JMP $0340
5640 JMP $0340
5650 JMP $0340
5660 JMP $0340
5670 JMP $0340
5680 JMP $0340
5690 JMP $0340
5700 JMP $0340
5710 JMP $0340
5720 JMP $0340
5730 JMP $0340
5740 JMP $0340
5750 JMP $0340
5760 JMP $0340
5770 JMP $0340
5780 JMP $0340
5790 JMP $0340
5800 JMP $0340
5810 JMP $0340
5820 JMP $0340
5830 JMP $0340
5840 JMP $0340
5850 JMP $0340
5860 JMP $0340
5870 JMP $0340
5880 JMP $0340
5890 JMP $0340
5900 JMP $0340
5910 JMP $0340
5920 JMP $0340
5930 JMP $0340
5940 JMP $0340
5950 JMP $0340
5960 JMP $0340
5970 JMP $0340
5980 JMP $0340
5990 JMP $0340
6000 JMP $0340
6010 JMP $0340
6020 JMP $0340
6030 JMP $0340
6040 JMP $0340
6050 JMP $0340
6060 JMP $0340
6070 JMP $0340
6080 JMP $0340
6090 JMP $0340
6100 JMP $0340
6110 JMP $0340
6120 JMP $0340
6130 JMP $0340
6140 JMP $0340
6150 JMP $0340
6160 JMP $0340
6170 JMP $0340
6180 JMP $0340
6190 JMP $0340
6200 JMP $0340
6210 JMP $0340
6220 JMP $0340
6230 JMP $0340
6240 JMP $0340
6250 JMP $0340
6260 JMP $0340
6270 JMP $0340
6280 JMP $0340
6290 JMP $0340
6300 JMP $0340
6310 JMP $0340
6320 JMP $0340
6330 JMP $0340
6340 JMP $0340
6350 JMP $0340
6360 JMP $0340
6370 JMP $0340
6380 JMP $0340
6390 JMP $0340
6400 JMP $0340
6410 JMP $0340
6420 JMP $0340
6430 JMP $0340
6440 JMP $0340
6450 JMP $0340
6460 JMP $0340
6470 JMP $0340
6480 JMP $0340
6490 JMP $0340
6500 JMP $0340
6510 JMP $0340
6520 JMP $0340
6530 JMP $0340
6540 JMP $0340
6550 JMP $0340
6560 JMP $0340
6570 JMP $0340
6580 JMP $0340
6590 JMP $0340
6600 JMP $0340
6610 JMP $0340
6620 JMP $0340
6630 JMP $0340
6640 JMP $0340
6650 JMP $0340
6660 JMP $0340
6670 JMP $0340
6680 JMP $0340
6690 JMP $0340
6700 JMP $0340
6710 JMP $0340
6720 JMP $0340
6730 JMP $0340
6740 JMP $0340
6750 JMP $0340
6760 JMP $0340
6770 JMP $0340
6780 JMP $0340
6790 JMP $0340
6800 JMP $0340
6810 JMP $0340
6820 JMP $0340
6830 JMP $0340
6840 JMP $0340
6850 JMP $0340
6860 JMP $0340
6870 JMP $0340
6880 JMP $0340
6890 JMP $0340
6900 JMP $0340
6910 JMP $0340
6920 JMP $0340
6930 JMP $0340
6940 JMP $0340
6950 JMP $0340
6960 JMP $0340
6970 JMP $0340
6980 JMP $0340
6990 JMP $0340
7000 JMP $0340
7010 JMP $0340
7020 JMP $0340
7030 JMP $0340
7040 JMP $0340
7050 JMP $0340
7060 JMP $0340
7070 JMP $0340
7080 JMP $0340
7090 JMP $0340
7100 JMP $0340
7110 JMP $0340
7120 JMP $0340
7130 JMP $0340
7140 JMP $0340
7150 JMP $0340
7160 JMP $0340
7170 JMP $0340
7180 JMP $0340
7190 JMP $0340
7200 JMP $0340
7210 JMP $0340
7220 JMP $0340
7230 JMP $0340
7240 JMP $0340
7250 JMP $0340
7260 JMP $0340
7270 JMP $0340
7280 JMP $0340
7290 JMP $0340
7300 JMP $0340
7310 JMP $0340
7320 JMP $0340
7330 JMP $0340
7340 JMP $0340
7350 JMP $0340
7360 JMP $0340
7370 JMP $0340
7380 JMP $0340
7390 JMP $0340
7400 JMP $0340
7410 JMP $0340
7420 JMP $0340
7430 JMP $0340
7440 JMP $0340
7450 JMP $0340
7460 JMP $0340
7470 JMP $0340
7480 JMP $0340
7490 JMP $0340
7500 JMP $0340
7510 JMP $0340
7520 JMP $0340
7530 JMP $0340
7540 JMP $0340
7550 JMP $0340
7560 JMP $0340
7570 JMP $0340
7580 JMP $0340
7590 JMP $0340
7600 JMP $0340
7610 JMP $0340
7620 JMP $0340
7630 JMP $0340
7640 JMP $0340
7650 JMP $0340
7660 JMP $0340
7670 JMP $0340
7680 JMP $0340
7690 JMP $0340
7700 JMP $0340
7710 JMP $0340
7720 JMP $0340
7730 JMP $0340
7740 JMP $0340
7750 JMP $0340
7760 JMP $0340
7770 JMP $0340
7780 JMP $0340
7790 JMP $0340
7800 JMP $0340
7810 JMP $0340
7820 JMP $0340
7830 JMP $0340
7840 JMP $0340
7850 JMP $0340
7860 JMP $0340
7870 JMP $0340
7880 JMP $0340
7890 JMP $0340
7900 JMP $0340
7910 JMP $0340
7920 JMP $0340
7930 JMP $0340
7940 JMP $0340
7950 JMP $0340
7960 JMP $0340
7970 JMP $0340
7980 JMP $0340
7990 JMP $0340
8000 JMP $0340
8010 JMP $0340
8020 JMP $0340
8030 JMP $0340
8040 JMP $0340
8050 JMP $0340
8060 JMP $0340
8070 JMP $0340
8080 JMP $0340
8090 JMP $0340
8100 JMP $0340
8110 JMP $0340
8120 JMP $0340
8130 JMP $0340
8140 JMP $0340
8150 JMP $0340
8160 JMP $0340
8170 JMP $0340
8180 JMP $0340
8190 JMP $0340
8200 JMP $0340
8210 JMP $0340
8220 JMP $0340
8230 JMP $0340
8240 JMP $0340
8250 JMP $0340
8260 JMP $0340
8270 JMP $0340
8280 JMP $0340
8290 JMP $0340
8300 JMP $0340
8310 JMP $0340
8320 JMP $0340
8330 JMP $0340
8340 JMP $0340
8350 JMP $0340
8360 JMP $0340
8370 JMP $0340
8380 JMP $0340
8390 JMP $0340
8400 JMP $0340
8410 JMP $0340
8420 JMP $0340
8430 JMP $0340
8440 JMP $0340
8450 JMP $0340
8460 JMP $0340
8470 JMP $0340
8480 JMP $0340
8490 JMP $0340
8500 JMP $0340
8510 JMP $0340
8520 JMP $0340
8530 JMP $0340
8540 JMP $0340
8550 JMP $0340
8560 JMP $0340
8570 JMP $0340
8580 JMP $0340
8590 JMP $0340
8600 JMP $0340
8610 JMP $0340
8620 JMP $0340
8630 JMP $0340
8640 JMP $0340
8650 JMP $0340
8660 JMP $0340
8670 JMP $0340
8680 JMP $0340
8690 JMP $0340
8700 JMP $0340
8710 JMP $0340
8720 JMP $0340
8730 JMP $0340
8740 JMP $0340
8750 JMP $0340
8760 JMP $0340
8770 JMP $0340
8780 JMP $0340
8790 JMP $0340
8800 JMP $0340
8810 JMP $0340
8820 JMP $0340
8830 JMP $0340
8840 JMP $0340
8850 JMP $0340
8860 JMP $0340
8870 JMP $0340
8880 JMP $0340
8890 JMP $0340
8900 JMP $0340
8910 JMP $0340
8920 JMP $0340
8930 JMP $0340
8940 JMP $0340
8950 JMP $0340
8960 JMP $0340
8970 JMP $0340
8980 JMP $0340
8990 JMP $0340
9000 JMP $0340
9010 JMP $0340
9020 JMP $0340
9030 JMP $0340
9040 JMP $0340
9050 JMP $0340
9060 JMP $0340
9070 JMP $0340
9080 JMP $0340
9090 JMP $0340
9100 JMP $0340
9110 JMP $0340
9120 JMP $0340
9130 JMP $0340
9140 JMP $0340
9150 JMP $0340
9160 JMP $0340
9170 JMP $0340
9180 JMP $0340
9190 JMP $0340
9200 JMP $0340
9210 JMP $0340
9220 JMP $0340
9230 JMP $0340
9240 JMP $0340
9250 JMP $0340
9260 JMP $0340
9270 JMP $0340
9280 JMP $0340
9290 JMP $0340
9300 JMP $0340
9310 JMP $0340
9320 JMP $0340
9330 JMP $0340
9340 JMP $0340
9350 JMP $0340
9360 JMP $0340
9370 JMP $0340
9380 JMP $0340
9390 JMP $0340
9400 JMP $0340
9410 JMP $0340
9420 JMP $0340
9430 JMP $0340
9440 JMP $0340
9450 JMP $0340
9460 JMP $0340
9470 JMP $0340
9480 JMP $0340
9490 JMP $0340
9500 JMP $0340
9510 JMP $0340
9520 JMP $0340
9530 JMP $0340
9540 JMP $0340
9550 JMP $0340
9560 JMP $0340
9570 JMP $0340
9580 JMP $0340
9590 JMP $0340
9600 JMP $0340
9610 JMP $0340
9620 JMP $0340
9630 JMP $0340
9640 JMP $0340
9650 JMP $0340
9660 JMP $0340
9670 JMP $0340
9680 JMP $0340
9690 JMP $0340
9700 JMP $0340
9710 JMP $0340
9720 JMP $0340
9730 JMP $0340
9740 JMP $0340
9750 JMP $0340
9760 JMP $0340
9770 JMP $0340
9780 JMP $0340
9790 JMP $0340
9800 JMP $0340
9810 JMP $0340
9820 JMP $0340
9830 JMP $0340
9840 JMP $0340
9850 JMP $0340
9860 JMP $0340
9870 JMP $0340
9880 JMP $0340
9890 JMP $0340
9900 JMP $0340
9910 JMP $0340
9920 JMP $0340
9930 JMP $0340
9940 JMP $0340
9950 JMP $0340
9960 JMP $0340
9970 JMP $0340
9980 JMP $0340
9990 JMP $0340

```

```

031C- C9 25 1340 .1 CMP #'% ;%
031E- D0 03 1350 BNE .2
0320- 4C 3D 03 1360 JMP PRINT
0323- C9 3A 1370 .2 CMP #'% ;NONE OF ABOVE COMMANDS,
0325- B0 03 1380 BCS .3 ; SO PROCESS COLON AND
0327- 4C BE 00 1390 JMP FRMAMA ; RETURN TO CHRGT/CHRGOT
032A- 60 1400 .3 RTS ; OR TO CALLING ROUTINE
1410
1420 *-----
1430 * MAMA ROUTINES
1440
032B- EE CF 03 1440 GOSUB INC GHOLD ;BUMP GOSUB COUNT
032E- 60 1450 RTS ;BACK TO CALLING ROUTINE
1460
032F- A9 5B 1470 LBRACK LDA #$5B ;LEFT BRACKET
0331- 20 5C DB 1480 JSR OUTDO ;PRINT IT
0334- 4C B1 00 1490 JMP CHRGT ;BACK TO CHRGT
1500
0337- 20 58 FC 1510 GHOME JSR HOME ;HOME CURSOR, CLEAR SCREEN
033A- 4C B1 00 1520 JMP CHRGT ;BACK TO CHRGT
1530
033D- A2 00 1540 PRINT LDX #00 ;CHARACTER INDEX
033F- BD 4D 03 1550 .1 LDA .2,X ;GET CHARACTER
0342- 20 5C DB 1560 JSR OUTDO ;PRINT IT
0345- E8 1570 INX ;BUMP INDEX
0346- B0 0E 1580 CPX #.3-.2 ;DONE?
0348- 90
```


AMPER FREE SPACE CATALOG

Amper Free Space Catalog

by Ben E. Colley
Rt. #1 Box 67B
Hallsville, MO 65255

Do you ever get tired of keying CATALOG? Perhaps you are like I am and get tired of miskeying CATALOG, all the while wishing you could save some keystrokes and get the same result. I also think it would be nice to know how much space is left on the diskette whenever you catalog it. The solution to these two little inconveniences is called Free Space Catalog, another program in the seemingly endless march of Applesoft ampersand utilities. Free Space Catalog (called FSC hereafter) allows you to obtain a disk catalog and free space information by simply keying &<RETURN>.

I really decided to write FSC for two reasons; 1) the little inconveniences cited above, and 2) (the main reason) because I was itching for an excuse to try my hand at the RWTS routine and some of the nifty subroutines somewhat hidden in Applesoft.

HOW IT WORKS

Very briefly, here is how FSC works. The program is loaded into page 3 of the Apple's memory, and does nothing except initialize the ampersand vector at \$3F5 when you BRUN it. When you call FSC by keying an ampersand in immediate mode, Applesoft passes control to FSC.

The routine executed by FSC is very straightforward. It saves the registers, and immediately does a subroutine call to the DOS CATALOG routine at location \$A56E (48K). Upon return, it requests the address of DOS's IOB (JSR \$3E3, line 70), reads track 17, sector 0 of the VTOC into the keyboard buffer (\$200), and calculates the number of sectors available. Then FSC simply calls several Applesoft and Monitor internal routines (CRDO, PRBLNK, LINPRT, STROUT, and DATA) to print the number of free sectors, restores the registers, and returns.

You can find the particulars of PRBLNK in the Apple Reference Manual; the rest (and a bunch more) are documented in the March/April 1980 issue of Apple Orchard. (This issue is highly recommended if you are at all interested in Applesoft internals.)

A nifty little feature of FSC is the call to DATA (JSR DATA, line 128). This allows you to key in comments following the ampersand. For example:

& GAMES DISKETTE, VOL #1

Applesoft gives control to FSC when it encounters the Ampersand; FSC performs your catalog, prints the free space available, and politely ignores any commentary you may have keyed. This feature is very handy when using FSC to print catalogs on a printer — any added comments will be printed, but will not cause a SYNTAX ERROR.

HOW TO ENTER FSC

As with most Assembly Language programs, what you need to know to enter the program is the starting memory location and the hexadecimal codes to enter into your

```
0000: 2 * * * * *
0000: 3 *          AMPER FREE SPACE CATALOG
0000: 4 *
0000: 5 *          BY BEN E. COLLEY
0000: 6 *          COPYRIGHT (C) 1982
0000: 7 *          BY MICRO-SPARC INC
0000: 8 *          LINCOLN, MA. 01773
0000: 9 * THIS PROGRAM CALLS THE DOS CATALOG ROUTINE
0000: 10 * AND UPON RETURN READS THE VTOC TO DETER-
0000: 11 * MINE THE NUMBER OF FREE SECTORS REMAINING
0000: 12 * ON THE DISKETTE. YOU MAY PLACE ANYTHING
0000: 13 * AFTER THE AMPERSAND, THE PROGRAM IGNORES
0000: 14 * IT. (HANDY FOR TITLING PRINTER LISTINGS)
0000: 15 *
0000: 16 * * * * *

0000: 18 * DEFINE ADDRESS CONSTANTS

03F5: 20 AMPERV EQU $3F5    AMPERSAND JUMP ADDR
DAFB: 21 CRDO EQU $DAFB    APPLESOFT CARRIAGE RETURN
D995: 22 DATA EQU $D995    APPLESOFT TEXT POINTER MOVE
A56E: 23 DOSCAT EQU $A56E    DOS CATALOG ROUTINE
03E3: 24 DOSIOB EQU $3E3    REQUEST DOS IOB ADDRESS
00F9: 25 IOBADR EQU $F9      IOB INDIRECT ADDR
ED24: 26 LINPRT EQU $ED24    APPLESOFT INTEGER OUTPUT
F948: 27 PRBLNK EQU $F948    MONITOR PRINT 3 BLANKS
03D9: 28 RWTS EQU $3D9    DOS RWTS ROUTINE
DB3A: 29 STROUT EQU $DB3A    APPLESOFT STRING PRINT

0000: 31 * DEFINE OTHER CONSTANTS

00C4: 33 VTOCND EQU $C4    1ST BYTE AFTER BIT MAPS

0000: 35 * DEFINE IOB OFFSETS

0003: 37 IBVOL EQU $03
0004: 38 IBTRK EQU $04
0005: 39 IBSECT EQU $05
000D: 40 IBSTAT EQU $0D    DOS RETURN CODE
000B: 41 IBBUFP EQU $0B
000C: 42 IBCMD EQU $0C
0000: 43 *
0000: 44 * DEFINE IOB VALUES
0000: 45 *
0000: 46 VOLID EQU $00    0=> IGNORE VOL ID
0011: 47 TRACK EQU $11
0000: 48 SECTOR EQU $00
0200: 49 BUFADR EQU $200    USE KEYBOARD BUF !!
0001: 50 DOSRD EQU $01

----- NEXT OBJECT FILE NAME IS AMPER FREE SPACE CATALOG.OBJO
0300: 52 ORG $300
0300: 54 FSCI EQU *      ***INITIALIZATION***
0300:A9 0B 55 LDA #>FSC    SET AMPERSAND VECTOR
0302:BD F6 03 56 STA AMPERV+1
0305:A9 03 57 LDA #<FSC
0307:BD F7 03 58 STA AMPERV+2
030A:60 59 RTS
030B: 60 *
030B: 61 * CALL DOS CATALOG, THEN CALC FREE SPACE
030B: 62 *
030B: 63 FSC EQU *
030B: 64 PHA
030C:9B 65 TYA          SAVE REGS
030D:4B 66 PHA
030E:8A 67 TXA
030F:4B 68 PHA
0310:20 6E A5 69 JSR DOSCAT
0313:20 E3 03 70 JSR DOSIOB    UPON RETURN
0316:84 F9 71 STY IOBADR    Y-REG HAS LOW BYTE AND
0318:85 FA 72 STA IOBADR+1    A-REG HAS HIGH BYTE OF IOB
031A:A0 03 73 LDY #IBVOL    TELL DOS TO IGNORE
031C:A9 00 74 LDA #VOLID    THE VOLUME ID
031E:91 F9 75 STA (IOBADR),Y
0320:A0 04 76 LDY #IBTRK    SET TRACK TO VTOC ($11)
0322:A9 11 77 LDA #TRACK
0324:91 F9 78 STA (IOBADR),Y
0326:A0 05 79 LDY #IBSECT    DITTO SECTOR
0328:A9 00 80 LDA #SECTOR
032A:91 F9 81 STA (IOBADR),Y
032C:A0 0B 82 LDY #IBBUFP    SET BUFFER TO READ
032E:A9 00 83 LDA #>BUFADR    THE DATA INTO
0330:91 F9 84 STA (IOBADR),Y
0332:C8 85 INY
0333:A9 02 86 LDA #<BUFADR
0335:91 F9 87 STA (IOBADR),Y
0337:A0 0C 88 LDY #IBCMD    SET DOS COMMAND
0339:A9 01 89 LDA #DOSRD    TO READ THE SECTOR
033B:91 F9 90 STA (IOBADR),Y
033D:A4 F9 91 LDY IOBADR
```



```

033F:A5 FA 92 LDA IOBADR+1 INIT REGS FOR RWTS
0341:20 D9 03 93 JSR RWTS GO READ THE VTOC
0344:A0 0D 94 LDY #IBSTAT CHECK DOS RETURN CODE
0346:B1 F9 95 LDA (IOBADR),Y
0348:F0 02 96 BEQ FSC10
034A:D0 54 97 BNE DOSERR GO PRINT ERROR MSG
034C:A9 00 99 FSC10 LDA #0 CLEAR FREE SECTOR COUNT
034E:BD 90 03 100 STA FRESEC
0351:BD 91 03 101 STA FRESEC+1
0354:A0 38 102 LDY ##38 INFO BEGINS HERE
0356:B9 00 02 104 FSC20 LDA BUFADR,Y GET NEXT BYTE INTO A-REG
0359:A2 07 105 LDX #7 INIT BIT COUNT REG
035B:0A 106 FSC30 ASL A SHIFT BIT INTO CARRY
035C:90 08 107 BCC FSC40 CARRY CLEAR, SECT USED
035E:EE 91 03 108 INC FRESEC+1 COUNT=COUNT+1
0361:D0 03 109 BNE FSC40 NO NEED TO INCR HI-BYTE
0363:EE 90 03 110 INC FRESEC HAPPENS EVERY 256 FREE SECTORS
0366:CA 111 FSC40 DEX BIT COUNT - 1
0367:10 F2 112 BPL FSC30 GO AGAIN
0369:C8 113 INY CHECK NEXT BYTE IN VTOC
036A:C0 C4 114 CPY #VTOCND SCANNED ALL OF VTOC?
036C:90 E8 115 BLT FSC20
036E: 116 *
036E: 117 * FREE SPACE CALC DONE, PRINT RESULTS
036E: 118 *
036E:20 FB DA 119 JSR CRDD DO CARRIAGE RETURN
0371:20 48 F9 120 JSR PRBLNK GO PRINT 3 BLANKS
0374:AD 90 03 121 LDA FRESEC
0377:AE 91 03 122 LDX FRESEC+1
037A:20 24 ED 123 JSR LINPRT GO PRINT FREE SECTOR COUNT
037D:A9 92 124 LDA #>MSG POINT TO STRING
037F:A0 03 125 LDY #<MSG
0381:20 3A DB 126 JSR STROUT GO PRINT MSG
0384:20 FB DA 127 JSR CRDD GO DO CARRIAGE RETURN
0387:20 95 D9 128 JSR DATA TELL APPLESOFT TO MOVE TXTPTR
038A:68 129 PLA RESTORE REGS
038B:AA 130 TAX
038C:68 131 PLA
038D:A8 132 TAY
038E:68 133 PLA
038F:60 134 RTS ALL DONE!
0390:00 00 135 FRESEC DFB 0,0 COUNT CARRIED HI-BYTE FIRST
0392: 136 MSB OFF
0392:20 46 52 137 MSG ASC ' FREE SECTORS'
0395:45 45 20
0398:53 45 43
039B:54 4F 52
039E:53
039F:00 138 DFB 0
03A0: 140 DOSERR EQU *
03A0: 141 *
03A0: 142 * ROUTINE PRINT ERROR MESSAGE
03A0: 143 *
03A0:A9 AB 144 LDA #>ERRMSG GO PRINT ERROR MSG
03A2:A0 03 145 LDY #<ERRMSG
03A4:20 3A DB 146 JSR STROUT
03A7:60 147 RTS AND GO BACK
03AB:20 45 52 148 ERRMSG ASC ' ERROR READING VTOC'
03AB:52 4F 52
03AE:20 52 45
03B1:41 44 49
03B4:4E 47 20
03B7:56 54 4F
03BA:43
03BB:00 149 DFB 0

```

*** SUCCESSFUL ASSEMBLY: NO ERRORS

Amper Free Space Catalog (Cont.)

Apple. This code begins at Hex memory location \$300.

From the listing, you can see that the first codes entered into \$300 are:

300:A9 0B 8D F6 03 A9 03 8D F7 03 etc.

By continuing through the listing and entering the Hex codes, you can enter the program into your system. Then, when it is entered, you can save it to disk with the command:

BSAVE FSC, A\$300,L\$BC SYSTEM REQUIREMENTS

FSC runs in immediate mode on 48K Apples with **DOS 3.2 or 3.3** and Applesoft in ROM or in the language system. (Maybe some of you "wizards" can give me a clue about why it will not run consistently within a program?? It results in a PROGRAM TOO LARGE DOS message unless there is at least one preceding DOS command.) I think after you use FSC, the convenience will spoil you as it has me!

J& DOS 3.2 SYSTEM MASTER

DISK VOLUME 254

```

*I 002 HELLO
*I 053 APPLE-TREK
*I 018 ANIMALS
*B 009 UPDATE 3.2
*I 014 COPY
*I 009 COLOR DEMO
*I 053 BRICK OUT
*I 026 SPACE WAR
*I 050 THE INFINITE NO. OF MONKEYS
*I 051 COLOR SKETCH
*I 053 SUPERMATH
*I 026 APPLEVISON
*I 017 BIORHYTHM
*I 027 PINBALL

```

3 FREE SECTORS



APPLE M.L.E.

Apple M.L.E. (Machine Language Editor)

by Doug Sprinkle
4269 Bristol
Troy, MI 48098

Many publications, including this magazine, publish assembly language programs for use by their readers. In many cases the entire assembly program is published, but in other cases only a hexadecimal memory dump is provided. The entry of these programs into the reader's own computer, however, is sometimes a major undertaking. Often the user does not own an Assembler which allows him or her to enter the program in assembly language, compile it, and then run the program. Although the hexadecimal memory dump (also printed in an assembly listing) can be directly entered from the Apple II monitor, this process is both tedious and prone to error. Usually an omitted byte or line requires retyping a good portion of the program.

Apple MLE (Machine Language Editor) eliminates these problems by providing a machine (not assembly) language editor which facilitates the entry and editing of machine language programs. A machine language program differs from an assembly language program in that machine language consists entirely of hexadecimal bytes, while an assembly language program is written with mnemonics (such as LDA, AND and the like), and is subsequently compiled or assembled into a machine language program.

MLE is initially called by an Applesoft program, but the Applesoft program does nothing more than handle the disk functions as will be subsequently described. **All data entry and editing of the machine code program is handled by MLE** which is written in machine code. My apologies to Applesoft programmers, but the necessary speed could not be obtained (at least by me) in Applesoft. MLE resides in just under 2.5K of memory, requires Applesoft and 48K of memory.

HOW THE PROGRAM WORKS

MLE is "run" by simply typing **RUN MLE.DRIVER** from Applesoft. MLE.DRIVER is an Applesoft program which automatically loads MLE into memory and then immediately calls MLE.B, i.e. the machine code portion of MLE. The user is presented with the following menu, each section of which is described below:

START NEW FILE	1
EDIT/LIST WORKFILE	2
LOAD BINARY FILE	3
SAVE FILE TO DISC	4
EXIT TO MONITOR	5

START NEW FILE

As its name implies, the first option enables the user to begin a new workfile. Selecting this option will destroy any workfile present, but a warning is first displayed which permits the selection to be aborted. Assuming that the selection is not aborted, **the user is required to enter both the starting address and**

the number of columns of bytes desired. Any starting address is acceptable, even if it conflicts with MLE, and **up to eight columns of bytes** will be displayed. The variable number of columns is designed to enter programs from hexadecimal "dumps" with different numbers of columns. You will usually choose eight.

MLE displays the **current address** (starting with the beginning address selected by the user) at the beginning of each line and prompts you to enter the machine code for the program. **MLE accepts only valid hex characters (0-F)** and the left and right arrows are fully supported for any necessary minor corrections. Major corrections, however, should be left to the EDIT/LIST option since major corrections are, of course, its forte.

The prompt line at the bottom of the screen reminds the user that **Control E will exit** from this section of the program. The prompt also displays the length of the file, i.e. the number of bytes entered in hexadecimal. Control E should be remembered since it is used to exit from virtually all sections of the program.

EDIT/LIST WORKFILE

After the program has been entered, either from the keyboard as described above or from the disk as described below, the errors in the program may be corrected by selecting this option. Most of the code in MLE is devoted to this section of the program.

After selection of this option, up to the **first fifteen lines** of the binary file in columnar format are displayed and the cursor homes on the first byte in the file. The binary file length as well as the prompt line (**X,D,I,<-,->,;/,E**) are also displayed at the bottom of the screen. The prompt line identifies the available commands. These commands can be generally divided into **two types: Cursor Commands and Data Altering Commands.**

CURSOR COMMANDS (<- ->; /)

All cursor commands are automatically constrained within the limits of the binary file plus one byte. The additional byte is allowed to permit appending data to the existing workfile.

The **left arrow (<-)** moves the cursor one byte to the left. Similarly, the **right arrow (->)** moves the cursor one byte to the right. The **semi-colon (;)** moves the cursor up and the **down arrow (/)** moves the cursor down one full line. To accommodate long binary files, the **cursor will scroll up OR down** as required. Therefore, there is no need for a "return to beginning" command. You will also find that the scrolling is very fast.

All cursor commands move the cursor to the first hex digit of the leftward, rightward, upward or downward byte depending upon the cursor command selected.

DATA ALTERING COMMANDS (I,X,D)

The **Insert Command (Control I)** immediately creates a space at the current cursor position and shifts all following data rightward. The **left arrow** is supported only after one character has been entered. After two hex characters have been entered, MLE shifts rightward to the next column, opens a new

space and the above process is repeated. The letter I on the prompt line is set to the flashing mode to indicate that the insert mode is in effect. **The Insert command is terminated by Control E**, as always, and the space created by Insert is closed.

The **Delete Command (Control D)** deletes the byte at the current cursor position and redisplay the binary file. Since the entire operation is performed once the Delete Command is invoked, it is not necessary to terminate the command with Control E.

The **Xchange Command (Control X)** allows data within the text file to be altered on a per byte basis. The right arrow is fully supported to copy over data and the "X" in the prompt line is set to the flashing mode to indicate that the Xchange command is in effect. Control E terminates the Xchange command.

Both the Xchange Command and the Insert Command require that an entire byte, i.e. two hex digits, be exchanged or inserted (as the case may be) before the command can be terminated by Control E. Pressing Control E at the second byte will have no effect other than beeping the speaker to indicate that an illegal entry was attempted.

Finally, Control E returns the user to the Main Menu.

LOAD BINARY FILE

The third option from the Main Menu allows the user to load a binary file from disk. After prompting to insert the correct disk, a disk catalog is displayed and the user is asked to enter the disk file name. Any errors return the user to the Main Menu.

After the binary file is loaded, the user must enter the starting memory address and number of display columns desired. It makes no difference what the actual starting address of the binary file is, since all binary files are loaded into memory beginning at \$5000 (see THEORY OF OPERATION below).

Please be certain that the actual name of the binary file is entered by the user as it appears on the disk catalog. This is somewhat confusing when loading a previously saved binary program using Option 4 from the Main Menu (described below).

For example, assuming that a binary file was saved under the filename **TEST**, MLE actually saves two files under the names **TEST.IMD** and **TEST.ACT**. **When reloading this file** under this option (Option 3), **it is necessary to enter the filename TEST.IMD or TEST.ACT** (it doesn't matter which one) or else a disk error is indicated and MLE returns to the Main Menu.

The length of the binary file under the **LOAD FROM DISK** option is determined automatically by loading the contents of \$AA60 and \$AA61 (the Basic equivalent of PEEKing) and storing the loaded values in file length counter of MLE (\$4374 and \$4375).

SAVE FILE TO DISK

The fourth option from the Main Menu saves the current workfile to disk under the filename **FILE** or whatever other name is selected by the user. **MLE actually saves two disk files** under the names **FILE.ACT** and

Apple M.L.E. (Machine Language Editor) (Cont.)

FILE.IMD. MLE first saves the workfile in the disk file FILE.IMD with a starting address of \$5000. MLE then reloads this binary file at the address selected by the user and then resaves the file under the filename FILE.ACT.

This involved process is designed to enable binary files to be created by MLE even though such programs conflict with MLE, Applesoft or the Applesoft driving program. For example, if the binary program has a starting address between \$800 and \$5000, it will almost certainly "kill" the Applesoft driving program and/or MLE. However, even if this happens, the user will have a back-up program under the name FILE.IMD which can be loaded into the correct memory locations using the DOS AS option.

This section of MLE is the only portion with which I am not happy. I feel certain that a more elegant method exists for altering the starting address in the binary file as opposed to reloading and resaving the file. If the reader is inclined to program modifications, this is the place to do it.

EXIT TO MONITOR

The fifth and final option from the Main Menu is self-explanatory. It is designed to allow storage to cassette for diskless Apple users. It also enables the binary program entered by the user to be run directly from the monitor by using the monitor G command.

THEORY OF OPERATION

MLE.DRIVER, the Applesoft portion of MLE, immediately sets **HIMEM** to **\$3FF0** to prevent collisions between Applesoft variables and the machine code portion (MLE.B) of MLE. MLE.DRIVER then BLOADs MLE.B and CALLS MLE.B. Upon entry to MLE.B, the Main Menu is displayed.

The following memory map of MLE may prove useful:

MLE.DRIVER	\$801 - \$3FF0
HIMEM	\$3FF0
Variables	\$3FF1 - \$3FFF
MLE.B (start)	\$4000
Workfile (start)	\$5000

As shown above, the machine code portion of MLE (MLE.B), which handles all operations except disk interfacing, occupies \$4000 to about \$4A00, and the workfile begins at \$5000 and grows upward.

Since the Applesoft program **MLE.DRIVER** handles the disk interfacing, it is necessary to pass variables from MLE.B to Applesoft. MLE accomplishes this by reserving fifteen bytes (\$3FF1 to \$3FFF) above HIMEM but before the beginning of MLE.B. MLE.B stores the proper value in the variable space and MLE.DRIVER determines the value of the variable by executing a PEEK instruction. Not all fifteen bytes of the variable space are used.

Location **\$3FF1** is used as the "switch" to determine if a Save file or Load file command is desired (1 for save and 2 for load). MLE.DRIVER executes a **N = PEEK(16369)** in line 50 and then **ON N GOTO** in line 60. (Note: \$3FF1 is the same as decimal 16369.) Quite obviously, this same technique can be used to select more than just two options.

Locations **\$3FF2** and **\$3FF3** contain the file length and are used during the Save file

option. Locations **\$3FF4** and **\$3FF5** contain the beginning address and are also used during the Save file option.

Lastly, locations **\$3FF6** and **\$3FF7** are used by the Load file option as a "jump vector" (Applesoft variable **JP%** in line 160) for returning to the Load file routine portion of MLE.B rather than the Main Menu. As noted above, **MLE loads the binary file into memory starting at \$5000 regardless of the actual starting address of the binary file.** After Bloading the file, Applesoft executes a **CALL JP%** to return to MLE.B in the load file routine to input the starting address and number of columns by the user.

Reserving a small segment of memory, in this case fifteen bytes, to pass variables between a machine code program and Applesoft, is a handy technique. Likewise, the method for using a "jump vector" to return to different portions of a machine code program from Applesoft should be in every programmer's bag of tricks.

Error trapping is extensively employed throughout MLE. For example, it is not possible to enter anything other than hex characters (0-9 and A-F) when hex data is expected. A beep from the Apple speaker will usually remind you when an incorrect key is pressed from the keyboard. **If you nevertheless crash MLE by pressing the reset key for example, RUN MLE.DRIVER should return you to the Main Menu with your workfile intact.** If you crash the Applesoft program, restart MLE by typing 4000G from the monitor, but note that the SAVE and LOAD disk operations will no longer be operational. The workfile, however, can still be saved by exiting to the monitor (Option 5), returning to Basic (3D0G) and using the BSAVE FILE, A\$5000, L\$ (your program length) DOS command. The file length in hex is always displayed, and updated as required, during the EDIT/LIST Main Menu option.

THE APPLESOFT DRIVER

The Applesoft driving program MLE.DRIVER is rather straightforward but contains some odd looking DOS commands. These DOS commands use variables as the filename (F\$), the starting address (A) and the file length (L). The values of these variables are determined by PEEKing the appropriate memory locations between \$3FF1 and \$3FFF. The use of multiple variables in the DOS commands is a legal and powerful, but not often used, capability of Apple DOS.

The Applesoft driver also traps disk errors and displays the appropriate disk error code before returning to the Main Menu. See your DOS manual for the proper interpretation. MLE is really an exercise in maintaining and continually updating pointers. Locations **\$F9** and **\$FA (PNTR)** represent the actual address in the workfile (\$5000+) while locations **\$436E** and **\$436F (IMADD)** represent the Image Address depending upon the starting location selected by the user. Obviously, both **PNTR** and **IMADD** must be updated during all cursor commands, data entry and data altering commands. Conversely, the file length (**FCNT**) must be updated only during the data entry and data altering commands since the file length does not change during cursor commands.

When I first began to write MLE, I did everything on a straight line approach, i.e. each section of the program was self-contained

and complete. This, however, was a mistake, since MLE grew to almost 4K in length before the disk routines were written. It was also very difficult to keep track of all the pointers for each routine.

I subsequently rewrote MLE into numerous subroutines which reduced its length to about 2.5K and simultaneously made life much easier. For example, the semi-colon routine, after rewriting, does nothing more than determine the number of columns (COLMAX) and then execute the same number of left arrows. Since all pointer updating and error checking is performed by the left arrow routine (ARRL), duplication in the up arrow routine is eliminated. I strongly suggest you take this approach when writing your own programs; I know I will.

Finally, the source listing is very well documented for any necessary or desired modifications.

SAMPLE RUN

Now let's use MLE to enter the following machine code program:

```
300:20 58 FC A2 00 A0 27 A9
308:20 20 ED FD 88 D0 F8 E8
310:E0 18 F0 03 4C 05 03 A9
318:0D 8D 3A 05 A9 0C 8D 3B
320:05 A9 05 8D 3C 05 60
```

Run MLE and select Option 1 to start a new workfile. The beginning address is **\$300** (hex), and the number of display columns is **eight**. Then enter the program below and, when you finish, press Control E to return to the Main Menu.

Now select the Edit file option (Option 2) and the following is displayed:

```
300:10 20 58 FC A2 00 A0 27
308:A9 20 20 ED FD 89 D0 F8
310:E8 E0 18 F0 03 4C 05 03
318:A9 8D 3A 05 A9 0C 8D 3B
320:05 A9 05 8D 3C 05 60
```

Now, let's say that you have made some mistakes which, fortunately, the editor allows you to correct.

First of all, byte 10 has been entered (incorrectly) before the first byte of the file. Press Control D to delete this byte and the file is relisted. Further corrections, however, are still required.

Location **\$30C** in the original listing is **\$88** but somehow you have entered **\$89**. No problem. Use the right arrow and down arrow (/) to position the cursor at **\$30C**. Now press **Control X**, press the right arrow to copy over the "8" and type 8 to replace the erroneous 9. Press Control E to exit the Xchange function (the blinking "X" should disappear on the prompt line). Something, however, is still not right.

After comparing the listing in the magazine with the listing on the screen, you discover that you have failed to enter the byte at **\$318**. Use the cursor commands to move the cursor to **\$318** and press **Control I** for **Insert**. In the space created by Insert, type the missing byte (0D) and then press Control E to exit the Insert command.

After examining the listing and correcting any other errors, press Control E to return to the Main Menu. Save the file to your work disk (don't use the program disk; that's living too dangerously) and then exit to monitor (Op-

continued on next page

Apple M.L.E. (Machine Language Editor) (Cont.)

tion 5). Type 300G to run the program. Please note: If you do not or cannot Save to disk, then the program must be moved using the monitor move command to location \$300 before this program will run.

CONCLUSION

After entering MLE from the monitor, try entering your own machine language program (perhaps from this publication) using

MLE. You will be amazed at the difference between entering a machine code program using MLE and doing the same from the monitor.



1

```

1 REM *****
2 REM * M.L.E. DRIVER *
3 REM * BY DOUG SPRINKLE *
4 REM * COPYRIGHT (C) 1982 *
5 REM * BY MICRO-SPARC INC *
6 REM * LINCOLN, MA. 01773 *
7 REM *****
10 HIMEM: 16368
14 FOR I = 1 TO 9: READ A: POKE 18943 + I,A: NEXT
15 DATA 173,245,63,172,244,63,76,72,163
20 ONERR GOTO 200
24 D$ = CHR$(4): PRINT D$;"MON C,I,O": PRINT D$;"BLOAD
    MLE.B"
34 CALL 16384: REM MLE
35 PRINT : HOME : PRINT D$;"CATALOG": PRINT : PRINT
50 N = PEEK (16369): HTAB 10: PRINT "'EXIT' TO RETURN
    ": PRINT : HTAB 11: INPUT "ENTER FILE NAME ";F$: IF
    F$ = "" THEN F$ = "NO NAME"
55 IF F$ = "EXIT" THEN 34
60 ON N GOTO 70,150
70 REM SAVE RTN
80 L% = PEEK (16370) + 256 * PEEK (16371)
85 POKE 41794,76: POKE 41795,0: POKE 41796,74: REM MAK
    E PATCH
90 PRINT D$;"BSAVE";F$;"A20480,L"%L%
125 POKE 41794,173: POKE 41795,115: POKE 41796,170: REM
    CLEAR PATCH
130 GOTO 34
140 REM LOAD RTN
150 PRINT D$;"BLOAD";F$;"A$5000"
160 JP% = PEEK (16374) + 256 * PEEK (16375)
170 CALL JP%: GOTO 34
200 EC% = PEEK (222): HOME : VTAB 11: HTAB 12
210 PRINT "DISK ERROR CODE ";EC%: PRINT : HTAB 14: INPUT
    "PRESS RETURN ";A$: GOTO 34

```

00B4	33	COND	EDU #B4	;CONTROL D
00BB	34	SCOLON	EDU #BB	;SEMICOLON
00AF	35	SLASH	EDU #AF	;SLASH
00C1	36	LEFT	EDU #C1	;LETTER A
00C7	37	LEFT	EDU #C7	;LETTER F+1
00B8	38	LARRW	EDU #B8	;LEFT ARROW
0095	39	RARRW	EDU #95	;RIGHT ARROW
5000	40	STORE	EDU \$5000	;DATA
FF3A	41	BELL	EDU #FF3A	;BELL
F94B	42	PRBLNK	EDU #F94B	;PNT 3 BLANKS
FC5B	43	HOME	EDU #FC5B	;CLR SCREEN
4000	44			
4000	45			
4000	46			
4000 20 CA 44	47	MENU	JSR FULLSC	;RESET TO FULL SCR
4003 20 5B FC	48		JSR HOME	;CLEAR SCREEN
4006 20 BE FD	49		JSR CROUT	;MOVE DOWN ONE LINE
4009 A9 OE	50		LDA #14	;MOVE CURSOR
400B B5 24	51		STA CH	
400D A9 F1	52		LDA #MENUF	;MENU STRING
400F B5 FD	53		STA PNTADD	
4011 A9 43	54		LDA #MENUF	
4013 B5 FE	55		STA PNTADD+1	
4015 20 BE 44	56		JSR PRINT	
401B A2 04	57		LDX #4	;PRINT 4 CARR RETS
401A 20 BE FD	58	MLP	JSR CROUT	
401D CA	59		DEX	
401E D0 FA	60		BNE MLP	
4020 A9 18	61		LDA #24	;SET WINDOW WDTN
4022 B5 21	62		STA WNDWDTN	
4024 A9 09	63		LDA #9	;SET LEFT MARGIN
4026 B5 20	64		STA WNDLFT	
4028 20 BE FD	65		JSR CROUT	;MOVE DOWN TWO LINES
402B 20 BE FD	66		JSR CROUT	
402E A9 FD	67		LDA #STARTF	;START FILE STRING
4030 B5 FD	68		STA PNTADD	
4032 A9 43	69		LDA #STARTF	
4034 B5 FE	70		STA PNTADD+1	
4036 20 BB 44	71		JSR PRINT	
4039 20 BE FD	72		JSR CROUT	;PRINT 2 CRS
403C 20 BE FD	73		JSR CROUT	
403F A9 42	74		LDA #EDITF	;EDIT FILE STRING
4041 B5 FD	75		STA PNTADD	
4043 A9 44	76		LDA #EDITF	
4045 B5 FE	77		STA PNTADD+1	
4047 20 BB 44	78		JSR PRINT	;PRINT IT
404A 20 BE FD	79		JSR CROUT	
404D 20 BE FD	80		JSR CROUT	
4050 A9 14	81		LDA #LOADF	;LOAD FILE STRING
4052 B5 FD	82		STA PNTADD	
4054 A9 44	83		LDA #LOADF	
4056 B5 FE	84		STA PNTADD+1	
4058 20 BB 44	85		JSR PRINT	;PRINT IT
405B 20 BE FD	86		JSR CROUT	
405E 20 BE FD	87		JSR CROUT	
4061 A9 2B	88		LDA #SAVEF	;SAVE FILE STRING
4063 B5 FD	89		STA PNTADD	
4065 A9 44	90		LDA #SAVEF	
4067 B5 FE	91		STA PNTADD+1	
4069 20 BB 44	92		JSR PRINT	;PRINT IT
406C 20 BE FD	93		JSR CROUT	
406F 20 BE FD	94		JSR CROUT	
4072 A9 57	95		LDA #EXITF	;PRINT NXT LINE
4074 B5 FD	96		STA PNTADD	
4076 A9 44	97		LDA #EXITF	
4078 B5 FE	98		STA PNTADD+1	
407A 20 BB 44	99		JSR PRINT	;PRINT IT
407D 20 BE FD	100		JSR CROUT	
4080 20 BE FD	101		JSR CROUT	;MOVE DOWN 2 LINES
4083 20 BE FD	102		JSR CROUT	
4086 20 BE FD	103		JSR CROUT	
4089 A9 70	104		LDA #WNTSEL	
408B B5 FD	105		STA PNTADD	
408D A9 44	106		LDA #WNTSEL	
408F B5 FE	107		STA PNTADD+1	
4091 20 BB 44	108		JSR PRINT	;PRINT IT
4094 A9 27	109		LDA #39	;RESET WINDOW WIDTH
4096 B5 21	110		STA WNDWDTN	
4098 A9 00	111		LDA #0	;RESET LEFT MARGIN
409A B5 20	112		STA WNDLFT	
409C 20 0C FD	113	SELINP	JSR RDKEY	;INPUT CHAR
409F C9 B1	114		CMR #NUM1	;START NEW FILE?
40A1 D0 64	115		BNE M1	;NO
40A3 20 ED FD	116		JSR COUT	
40A6 20 5B FC	117		JSR HOME	
40A9 A9 0A	118		LDA #10	;SET CURSOR
40AB 20 5B FB	119		JSR TABV	
40AE A9 0D	120		LDA #13	
40B0 B5 24	121		STA CH	
40B2 A9 BB	122		LDA #WARN1	
40B4 B5 FD	123		STA PNTADD	
40B6 A9 44	124		LDA #WARN1	
40B8 B5 FE	125		STA PNTADD+1	
40BA 20 BB 44	126		JSR PRINT	
40BD 20 BE FD	127		JSR CROUT	
40C0 20 BE FD	128		JSR CROUT	
40C3 A9 97	129		LDA #WARN2	
40C5 B5 FD	130		STA PNTADD	
40C7 A9 44	131		LDA #WARN2	
40C9 B5 FE	132		STA PNTADD+1	;PRINT WARNING
40CB A9 OE	133		LDA #14	
40CD B5 24	134		STA CH	
40CF 20 BB 44	135		JSR PRINT	
40D2 20 BE FD	136		JSR CROUT	
40D5 20 BE FD	137		JSR CROUT	
40D8 20 BE FD	138		JSR CROUT	
40DB A9 B3	139		LDA #OKS	
40DD B5 FD	140		STA PNTADD	
40DF A9 44	141		LDA #OKS	

LISA 2.5 APPLE M.L.E. (MACHINE LANGUAGE EDITOR) - BY DOUG SPRINKLE
COPYRIGHT (C) 1982 BY MICRO-SPARC, INC.
LINCOLN, MA. 01773 - ALL RIGHTS RESERVED

0800	1	TTL "MACHINE LANGUAGE EDITOR"	
4000	2	ORG \$4000	
4000	3	OBJ \$800	
0772	4	FCOUNT EQU 1906	;SCREEN ADDR
F85B	5	TABV EQU \$F85B	;TAB ROUTINE
0025	6	CV EPZ \$25	;VERT CURSOR POS
0024	7	CH EPZ \$24	;HORIZ CURSOR POS
FDDA	8	PRBYTE EQU \$FDDA	;PRINT BYTE IN ACC
FC42	9	CLREOP EQU \$FC42	;CLR END OF PG
0004	10	FOUR EQU #4	;FOUR
0005	11	FIVE EQU #5	;CONSTANT FIVE
00A0	12	SPACE EQU #A0	;SPACE
00AD	13	DASH EQU #AD	;DASH
0023	14	WNDBTM EPZ \$23	;WINDOW BOTTOM
0020	15	WMDLFT EPZ \$20	;LEFT MARGIN
0021	16	WMDWDTN EPZ \$21	;WINDOW WIDTH
FD0D	17	COUT EQU \$FD0D	;SCREEN OUTPUT
FD0E	18	CROUT EQU \$FD0E	;CARR RET
FD0C	19	RDKEY EQU \$FD0C	;READ KEY
00B5	20	CONC EQU #B5	;CONTROL E
0098	21	CONX EQU #98	;CONTROL X
00B0	22	NUM0 EQU #B0	;NUMBER ZERO
00B1	23	NUM1 EQU #B1	;NUMBER ONE
00B2	24	NUM2 EQU #B2	;NUMBER TWO
00B3	25	NUM3 EQU #B3	;NUMBER THREE
00B4	26	NUM4 EQU #B4	;NUMBER FOUR
00B5	27	NUM5 EQU #B5	;ASCII 5
00B6	28	NUM9 EQU #B6	;ASC NINE + 1
00FD	29	PNTADD EPZ \$FD	;PRINTER POINTER
0028	30	BASL EPZ \$28	;SCREEN POS
00F9	31	PNTF EPZ \$F9	;POINTER FOR STORAGE
00B9	32	CONT EQU #B9	;CONTROL I

Apple M.L.E. (Machine Language Editor) (Cont.)

40E1 85 FE	142	STA PNTADD+1		41E3 20 5B FB	259	JSR TABV		: SCROLL WINDOW
40E3 A9 13	143	LDA #19		41E6 A9 15	260	LDA #15		: SET BOTTOM WINDOW
40E5 85 24	144	STA CH		41E8 85 23	261	STA WNDPTH		
40E7 20 8B 44	145	JSR PRINT		41EA 20 5B FC	262	JSR HOME		
40EA 20 0C FD	146	JSR RDKEY	: OK TO CLEAR?	41ED A9 00	263	LDA #STORE		: STORAGE PNTR
40ED C9 D9	147	CMP #D9	: "Y"	41EF 85 F9	264	STA PNTR		
40EF F0 03	148	BEQ M0	: OK TO CLEAR?	41F1 A9 50	265	LDA /STORE		: HIGH BYTE
40F1 4C 00 40	149	JMP MENU		41F3 85 FA	266	STA PNTR+1		
40F4 20 27 41	150	JSR NEWFILE		41F5 A9 00	267	LDA #0		
40F7 20 CD 41	151	JSR SETUP		41F7 8D 74 43	268	STA FCNT		: SET FCNT TO ZERO
40FA 20 FE 41	152	JSR SET1	: SET COL INPUT	41FA 8D 75 43	269	STA FCNT+1		
40FD 20 16 42	153	JSR SET2	: INPUT CHARACTERS	41FD 60	270	RTS		
4100 A9 18	154	LDA #18	: RESET WINDOW	41FE	271			
4102 85 23	155	STA WNDPTH	: BOTTOM	41FE	272	: SET COLUMN INPUT		
4104 4C 00 40	156	JMP MENU	: DONE W/NEWFILE	41FE	273			
4107 C9 82	157	CMP #NUM2	: EDIT FILE?	41FE AD 84 43	274	SET1 LDA COLMAX		: COL MAX
4109 D0 03	158	BNE M2	: NO	4201 8D 73 43	275	STA COLCT		: COL CNTR
410B 4C D7 44	159	JMP EFILE		4204 8D 84 43	276	STA COLMAX		: MAX COLS
410E C9 83	160	CMP #NUM3	: LOAD FILE?	4207 AA	277	TAX		: NUM OF COL IN X-REG
4110 D0 03	161	BNE M3	: NO	4208 18	278	CLC		
4112 4C 87 49	162	JMP LDFILE		4209 A9 00	279	LDA #0		: SET ACC TO ZERO FOR MULT
4115 C9 84	163	CMP #NUM4	: SAVE FILE?	420B 69 04	280	ADC #FOUR		: ADD FOUR
4117 D0 03	164	BNE M4		420D CA	281	DEX		: DEC COLCNTR
4119 4C 66 49	165	JMP SVFILE		420E D0 FB	282	BNE CLP		: MULTIPLY COMPLETE
411C C9 85	166	CMP #NUM5	: EXIT?	4210 69 04	283	CLP1 STA #4		: ADD 4
411E D0 01	167	BNE M5		4212 8D 81 43	284	STA RTCHAR		: RIGHTMOST CHAR
4120 00	168	BRK		4215 60	285	RTS		
4121 20 3A FF	169	JSR BELL	: ERROR	4216	286			
4124 4C 9C 40	170	JMP SELINP	: READ SELECTION AGAIN	4216	287	: PRINT HEX ADDRESS		
4127	171			4216	288			
4127	172	: GET START ADDR AND COL CNT		4216 AD 84 43	289	SET2 LDA COLMAX		: RESET COL NUM
4127	173			4219 8D 73 43	290	STA COLCT		
4127 20 5B FC	174	NEWFILE JSR HOME	: CLEAR SCREEN	421C 20 EB 45	291	PTADDR JSR PRTADR		: PRINT INADD
412A A9 0A	175	LDA #A	: SET VERT POS	421F	292			
412C 20 5B FB	176	JSR TABV		421F	293	: TWO CHARACTER INPUT LOOP		
412F A9 08	177	LDA #8	: SET HORIZ POS	421F	294			
4131 85 24	178	STA CH		421F A9 01	295	CHIN LDA #1		: CHAR CNTR
4133 A9 89	179	LDA #STAD	: PRINT START ADDR	4221 8D 72 43	296	STA CHCNT		
4135 85 FD	180	STA PNTADD		4224 A9 00	297	LDA #0		: RESET TEMP TO ZERO
4137 A9 43	181	LDA /STAD+1		4226 8D 76 43	298	STA TEMP		
4139 85 FE	182	STA PNTADD+1		4229	299			
413B 20 8B 44	183	JSR PRINT	: PRINT IT	4229	300	: SINGLE CHARACTER INPUT LOOP		
413E	184			4229 20 0C FD	302	LOOP1 JSR RDKEY		: GET CHAR
413E	185	: INPUT STARTING ADDRESS		422C C9 85	303	CMP #CONE		: CONTROL E?
413E	186			422E D0 08	304	BNE C1		: CONTINUE
413E A9 01	187	ADDINP LDA #1	: TEMP CNTR	4230 A9 01	305	LDA #1		
4140 8D 85 43	188	STA RTNFLG	: SET RET FLG	4232 CD 72 43	306	CMP CHCNT		: SECOND CHAR?
4143 A2 01	189	LDX #1		4235 D0 01	307	BNE C1		: FIRST CHAR
4145 A9 01	190	LDA #1		4237 60	308	RTS		
4147 8D 72 43	191	STA CHCNT	: CHAR CNTR	4238 C9 88	309	C1 CMP #LARRW		: LEFT ARROW?
414A A9 00	192	LDA #0		423A D0 6C	310	BNE RTCH		: NO
414C 8D 76 43	193	STA TEMP	: RESET TEMP TO ZERO	423C 8D 80 43	311	STA T1		: STORE --
414F 20 0C FD	194	JSR RDKEY	: GET INPUT	423F AD 72 43	312	LDA CHCNT		: CHAR CNT
4152 20 C2 42	195	JSR VALHEX		4242 C9 00	313	CMP #0		: SECOND CHAR?
4155 C9 FF	196	CMP #FF		4244 D0 0C	314	BNE FSTCH		: NO
4157 F0 F6	197	BEQ ADD1		4246 AD 80 43	315	LDA T1		: LOAD --
4159 20 EC 42	198	JSR STCH	: STORE CHAR	4249 20 ED FD	316	JSR COUT		: PRINT IT
415C AD 72 43	199	LDA CHCNT		424C 4C 1F 42	317	JMP CHIN		
415F C9 FF	200	CMP #FF		424F 4C 29 42	318	LOOP4 JMP LOOP1		
4161 D0 EC	201	BNE ADD1	: FULL BYTE INPUTTED	4252	319			
4163 AD 76 43	202	LDA TEMP		4252	320			
4166 9D 4E 43	203	STA IMADD, X	: STORE IT	4252	321	: LEFT ARROW ROUTINE		
4169 9D 70 43	204	STA BEGADD, X	: SAVE BEGIN ADDR	4252	322			
416E F0 04	205	CPX #0	: DONE?	4252	323			
4170 CA	206	BEQ ADD2		4252 AD 73 43	324	FSTCH LDA COLCT		: COLUMN CNT
4171 4C 45 41	207	DEX	: BYTE CNTR	4255 CD 84 43	325	CMP COLMAX		: FIRST COLUMN?
4174	208	JMP ADD0	: GET NXT BYTE	4258 D0 21	326	BNE FSTCH1		: NO
4174	209			425A AD 81 43	327	LDA RTCHAR		: RIGHTMOST CUR POS
4174	210	: ENTER COL CNTR (COLMAX)		425D 85 24	328	STA CH		
4174	211			425F A5 25	329	LDA CV		: CUR VERT POSITION
4174 20 8E FD	212	ADD2 JSR CROUT	: PRINT 2 CRS	4261 38	330	SEC		
4177 20 8E FD	213	JSR CROUT		4262 E9 01	331	SBC #1		: SUBTRACT ONE
417A A9 08	214	LDA #8	: CUR HORIZ POS	4264 B0 07	332	BGE TOPLINE		: TOP OF PAGE
417C 85 24	215	STA CH		4266 A9 07	333	LDA #7		: LEFT CUR POS
417E A9 9D	216	LDA #COLNUM	: PRINT FOR COL NUM	4268 85 24	334	STA CH		
4180 85 FD	217	STA PNTADD		426A 4C 08 43	335	JMP ERROR1		: TOP OF PAGE
4182 A9 43	218	LDA /COLNUM		426D 20 5B FB	336	JSR TABV		: MONITOR TABV ROUTINE
4184 85 FE	219	STA PNTADD+1		4270 A9 00	337	LDA #0		
4186 20 8B 44	220	JSR PRINT	: PRINT IT	4272 8D 72 43	338	STA CHCNT		: RESET CHAR CNT
4189 20 0C FD	221	JSR RDKEY	: READ KYBD	4275 8D 73 43	339	STA COLCT		: RESET COL CNT
418C 8D 80 43	222	STA T1		4278 4C 87 42	340	JMP FSO		
418F 38	223	SEC	: CONVERT ASCII	427B C6 24	341	FSTCH1 DEC CH		: DEC CH BY 4
4190 E9 B0	224	SBC #B0		427D C6 24	342	DEC CH		
4192 8D 84 43	225	STA COLMAX	: STORE COL CNTR	427F C6 24	343	DEC CH		
4195 C9 09	226	CMP #9	: MAX COL NUM-1	4281 A9 00	344	LDA #0		: RESET CHAR CNT
4197 90 06	227	BLT NEWFL1	: START INPUT RTN	4283 8D 72 43	345	STA CHCNT		: SET CARRY FOR SBC
4199 20 3A FF	228	JSR BELL	: ERROR	4286 38	346	SEC		
419C 4C 89 41	229	JMP ADD4		4287 EE 73 43	347	FS0 INC COLCT		
419F A9 00	230	LDA #0	: CLR FLG	428A CE 74 43	348	DEC FCNT		
41A1 8D 85 43	231	STA RTNFLG	: SET RET FLAG	428D AD 74 43	349	LDA FCNT		
41A4 AD 80 43	232	LDA T1	: PRINT CHA	4290 C9 F7	350	CMP #FF		
41A7 20 ED FD	233	JSR COUT	: MOVE DOWN	4292 D0 03	351	BNE FS1		
41AA 20 8E FD	234	JSR CROUT		4294 CE 75 43	352	DEC FCNT+1		: DEC IMADD & PNTR
41AD 20 8E FD	235	JSR CROUT		4297 20 85 46	353	JSR DECP1		: PRESET X REG FOR LOAD
41B0 20 8E FD	236	JSR CROUT		429A A2 00	354	LDX #0		: LOAD VAL AT LOC
41B3 A9 13	237	LDA #19		429C A1 F9	355	LDA (PNTR, X)		: MASK LOWER NIBBLE
41B5 85 24	238	STA CH		429E 29 F0	356	AND #F0		: STORE VALUE
41B7 A9 83	239	LDA #OKS		42A0 81 F9	357	STA (PNTR, X)		: STORE IN TEMP VALUE
41B9 85 FD	240	STA PNTADD		42A2 8D 76 43	358	STA TEMP		
41BB A9 44	241	LDA /OKS		42A5 4C 29 42	359	JMP LOOP1		
41BD 85 FE	242	STA PNTADD+1		42A8	360			
41BF 20 8B 44	243	JSR PRINT	: PRINT IT	42A8	361	: DETERMINE IF VALID HEX NUM		
41C2 20 0C FD	244	JSR RDKEY		42A8	362			
41C5 C9 D9	245	CMP #D9	: YES	42A8 C9 95	363	RTCH CMP #ARRRW		: -->?
41C7 D0 01	246	BNE ADD7		42AA D0 04	364	BNE NARR		
41C9 60	247	RTS		42AC AA 24	365	LDY CH		
41CA 4C 27 41	248	JMP NEWFILE		42AE B1 28	366	LDA (BASL, Y)		: SCN CHAR
41CD A9 17	249	LDA #17	: MOVE TO BOTTOM LINE	42B0 20 C2 42	367	NARR JSR VALHEX		: CHK FOR HEX NO.
41CF 20 5B FB	250	JSR TABV		42B3 C9 FF	368	CMP #FF		: ERROR?
41D2 A9 05	251	LDA #5	: RESET HORIZ POS	42B5 F0 98	369	BEQ LOOP4		: YES-ERROR
41D4 85 24	252	STA CH		42B7 20 EC 42	370	JSR STCH		: STORE CHAR
41D6 A9 D3	253	LDA #PMTLN	: PROMPT LINE	42BA AD 72 43	371	LDA CHCNT		: 1ST OR 2ND CHAR?
41D8 85 FD	254	STA PNTADD		42BD F0 90	372	BEQ LOOP4		: 1ST CHAR
41DA A9 43	255	LDA /PMTLN		42BF 4C 1F 42	373	JMP CHIN		
41DC 85 FE	256	STA PNTADD+1		42C2	374			
41DE 20 8B 44	257	JSR PRINT	: PRINT IT	42C2	375	: CHK FOR VALID HEX CHAR		
41E1 A9 05	258	LDA #5	: MOVE CURSOR INTO	42C2	376			
				42C2 C9 80	377	VALHEX CMP #NUM0		: JMP FOR ERROR
				42C4 90 42	378	BLT ERROR1		

continued on next page

Apple M.L.E. (Machine Language Editor) (Cont.)

ADDRESS	OPERATION	OPERAND	COMMENT
423A C9 BA	379	CMP #0100	LESS THAN NINE?
423B 80 0D	380	BGE LETTER	NO - ITS A LETTER
423C 8D 80 43	381	STA T1	STORE BEFORE COUT
423D 20 ED FD	382	JSR COUT	PRINT CHAR
423E AD 80 43	383	LDA T1	GET CHAR
423F 38	384	SEC	SET CARRY FOR SBC
4240 E9 80	385	SBC #080	CONVERT ASCII
4241 60	386	RTS	
4242 C9 C1	387	CMP #LETA	LETTER A?
4243 90 2D	388	BLT ERROR1	
4244 C9 C7	389	CMP #LETF	LETTER F?
4245 80 29	390	BGE ERROR1	
4246 8D 80 43	391	STA T1	STORE CHAR
4247 20 ED FD	392	JSR COUT	PRINT IT
4248 AD 80 43	393	LDA T1	GET CHAR BACK
4249 38	394	SEC	SET CARRY FOR SUBTRACT
424A E9 B7	395	SBC #0B7	CONVERT ASCII
424B 60	396	RTS	
424C	397	*	
424D	398	* DETERMINE IF FIRST OR	
424E	399	* SECOND CHAR HAS BEEN ENTERED	
424F	400	*	
4250 18	401	STCH	
4251 6D 76 43	402	ADC TEMP	TEMP STORAGE
4252 8D 76 43	403	STA TEMP	
4253 AD 72 43	404	LDA CHCNT	CHAR COUNTER
4254 C9 00	405	CMP #00	ZERO
4255 F0 14	406	BEQ CHAR2	SECOND CHAR
4256 AD 76 43	407	LDA TEMP	GET TEMP BACK
4257 0A	408	ASL	SHIFT LEFT FOUR
4258 0A	409	ASL	
4259 0A	410	ASL	
425A 00 0A	411	ASL	
425B 8D 76 43	412	STA TEMP	STORE CHAR
425C 0E 72 43	413	DEC CHCNT	DEC CHAR COUNTER
425D 60	414	RTS	
425E 20 3A FF	415	ERROR1 JSR BELL	
425F A9 FF	416	LDA #0FF	
4260 60	417	RTS	ERROR FLAG
4261	418	*	
4262	419	* SECOND CHAR INPUTTED - NOW STORE	
4263	420	*	
4264 CE 72 43	421	CHAR2 DEC CHCNT	SET TO 0FF
4265 AD 85 43	422	LDA RTNPLG	RET FLAG
4266 F0 01	423	BEQ CHAR3	
4267 60	424	RTS	RETURN
4268 AD 76 43	425	CHAR3 LDA TEMP	LOAD CHAR
4269 A2 00	426	LDX #0	CLEAR X REG
426A B1 F9	427	STA (PNTR,X)	STORE IN ARRAY
426B EE 74 43	428	INC FCNT	
426C D0 03	429	BNE CT2	
426D EE 75 43	430	INC FCNT+1	INC FILE COUNT
426E 20 51 46	431	CT2 JSR INCP0	INC PNTR & IMADD
426F CE 73 43	432	CONT DEC COLCT	END OF LINE?
4270 D0 0B	433	BNE CONT1	NOT END OF LINE
4271 20 41 43	434	JSR FLDIS	DIS FILE LEN
4272 20 8E FD	435	JSR CROUT	PRINT CARR RET
4273 68	436	PLA	SIMULATE RTS
4274 68	437	PLA	
4275 4C 16 42	438	JMP SET2	GET NT CHAR - RSET COLCT
4276 E6 24	439	CONT1 INC CH	PRINT 2 BLKS
4277 E6 24	440	INC CH	
4278 20 41 43	441	JSR FLDIS	DISPLAY FILE LEN
4279 60	442	RTS	
427A 85 25	443	FLDIS LDA CVT	STORE CUR VERT POS
427B 8D 77 43	444	STA CVT	
427C 85 24	445	LDA CH	STORE CUR HOR POS
427D 8D 78 43	446	STA CHT	
427E A9 22	447	LDA #34	HORIZ POS FOR FCNT
427F 85 24	448	STA CH	
4280 A9 17	449	LDA #17	CUR VERT POS
4281 20 5B FB	450	TEST JSR TABV	
4282 AD 75 43	451	LDA FCNT+1	
4283 20 DA FD	452	JSR PRBYTE	
4284 8A 74 43	453	LDA FCNT	
4285 20 DA FD	454	JSR PRBYTE	PRINT FILE LEN
4286 AD 78 43	455	LDA CHT	RESET CUR POS
4287 85 24	456	STA CH	
4288 8A 77 43	457	LDA CVT	
4289 20 5B FB	458	JSR TABV	
428A 60	459	RTS	
428B	460	*	
428C	461	* VARIABLES	
428D	462	*	
428E	463	ENDADR DFS #2	END ADDRESS
428F	464	IMADD DFS #2	IMAGE STORAGE ADDR
4290	465	BEGADD DFS #2	BEGINNING ADDRESS
4291	466	CHCNT DFS #1	CHAR CNT
4292	467	COLCT DFS #1	COL CNT
4293	468	FCNT DFS #2	BYTES IN FILE
4294	469	TEMP DFS #1	TEMP INPUT STORAGE
4295	470	CVT DFS #1	
4296	471	CHT DFS #1	TEMP CUR POS
4297	472	TPTR DFS #2	TEMP STORAGE FOR DNSCR
4298	473	TIMD DFS #2	TEMP STOR FOR DNSCR
4299	474	T2 DFS #1	TEMP STORAGE
429A	475	T3 DFS #1	TEMP STOR FOR SCROLL
429B	476	T4 DFS #1	

44E7 A9 43	532	LDA /CMMDL		460D 20 3A FF	661	JSR BELL	
44E8 B5 FE	533	STA PNTADD+1		4610 A0 00	662	LDY #0	
44EB 20 BB 44	534	JSR PRINT	:PRINT IT	4612 60	663	RTS	
44EE A9 05	535	LDA #5	:MOVE CUR INTO SCROLL	4613 A0 00	664	ARRR1	:END ADDR FLG
44F0 20 5B FB	536	JSR TABV		4615 AD 73 43	665	LDA COLCT	:COL CNT
44F3 A9 15	537	LDA #15	:SET BOTTOM WINDOW	4618 C9 01	666	CMP #1	:LAST COL?
44F5 B5 23	538	STA WNDRTM		461A F0 0C	667	BEQ ARRR2	:NOT LAST COL
44F7 A9 00	539	LDA #STORE	:RESET PNTR	461C CE 73 43	668	DEC COLCT	
44F9 B5 F9	540	STA PNTR		461F E6 24	669	INC CH	:MOVE CURSOR RT BY 4
44FB A9 50	541	LDA /STORE		4621 E6 24	670	INC CH	
44FD B5 FA	542	STA PNTR+1		4623 E6 24	671	INC CH	
44FF 1B	543	CLC	:SET END ADDR	4625 E6 24	672	INC CH	
4500 A5 F9	544	LDA PNTR		4627 60	673	RTS	
4502 6D 74 43	545	ADC FCNT		4628 A5 25	674	ARRR2	:LAST LINE?
4505 BD 6C 43	546	STA ENDADR		462A C9 14	675	CMP #14	
4508 A5 FA	547	LDA PNTR+1		462C D0 09	676	BNE ARRR3	:NOT LAST LINE
450A 6D 75 43	548	ADC FCNT+1		462E 20 CB 46	677	JSR SCROLL	:PNT NEXT LINE
450D BD 6D 43	549	STA ENDADR+1		4631 A9 14	678	LDA #14	:RESET VERT CUR
4510 20 5B FC	550	JSR HOME	:CLR SCREEN	4633 A2 01	679	LDX #1	:CNTR FOR SCROLL
4513 20 41 43	551	JSR FLDIS	:PRINT FILE CNT	4635 D0 04	680	BNE ARRR4	
4516 20 92 45	552	JSR LIST	:LIST ONE PAGE	4637 E6 25	681	ARRR3	:VERT CUR POS
4519 A9 07	553	LDA #7	:RESET CURSOR	4639 A5 25	682	LDA CV	
451B B5 24	554	STA CH		463B 20 5B FB	683	ARRR4	:RESET HOR CUR POS
451D A9 00	555	LDA #0		463E A9 07	684	LDA #7	
451F 20 5B FB	556	JSR TABV		464 85 24	685	STA CH	
4522 A9 00	557	LDA #STORE	:RESET PNTR	4642 AD B4 43	686	LDA COLMAX	
4524 B5 F9	558	STA PNTR		4645 BD 73 43	687	STA COLCT	
4526 A9 50	559	LDA /STORE		4648 60	688	RTS	
4528 B5 FA	560	STA PNTR+1		4649	689		
452A AD 70 43	561	LDA BEGADD	:RESET IMADD	4649	690		
452D BD 6E 43	562	STA IMADD		4649	691	INCREMENT POINTERS	
4530 AD 71 43	563	LDA BEGADD+1		4649 20 E9 48	692	INCP	:LAST ADDR?
4533 BD 6F 43	564	STA IMADD+1		464C D0 03	693	BNE INCP0	
4536 AD 84 43	565	LDA COLMAX	:SET COL CNTR	464E A0 01	694	LDY #1	:LAST ADDR FLG
4539 BD 73 43	566	STA COLCT		4650 60	695	RTS	
453C 20 0C FD	567	EMENU	:GET CMMD	4651 E6 F9	696	INCP0	:INC PNTR
453F C9 B5	568	CMP #CONE	:DONE?	4653 D0 02	697	BNE INCP1	
4541 F0 03	569	BNE EM1	:NP	4655 E6 FA	698	INC PNTR+1	
4543 4C 00 40	570	JMP MENU		4657 EE 6E 43	699	INCP1	:INC IMADD
4546 C9 B8	571	EM1	:LFT ARROW?	465A D0 03	700	BNE INCP2	
4548 D0 06	572	BNE EM2	:NO	465C EE 6F 43	701	INC IMADD+1	
454A 20 62 46	573	JSR ARRL		465F A0 00	702	INCP2	:LDY #0
454D 4C 3C 45	574	JMP EMENU		4661 60	703	RTS	
4550 C9 95	575	EM2	:RT ARROW?	4662	704		
4552 D0 06	576	BNE EM3	:NO	4662	705	LEFT ARROW RTN	
4554 20 04 46	577	JSR ARRL		4662	706		
4557 4C 3C 45	578	JMP EMENU		4662 A0 00	707	ARRL	:ERROR FLG
455A C9 AF	579	EM3	:DOWN ARROW?	4664 20 A4 46	708	JSR DECP	
455C D0 06	580	BNE EM4	:NO	4667 C0 00	709	CPY #0	:ERROR?
455E 20 43 47	581	JSR ARRD		4669 F0 06	710	BEQ ARRL1	:NO
4561 4C 3C 45	582	JMP EMENU		466B 20 3A FF	711	JSR BELL	
4564 C9 B8	583	EM4	:UP ARROW?	466E A0 00	712	LDY #0	
4566 D0 06	584	BNE EM5	:NO	4670 60	713	RTS	
4568 20 52 47	585	JSR ARRU		4671 AD B4 43	714	ARRL1	:LDA COLMAX
456B 4C 3C 45	586	JMP EMENU		4674 CD 73 43	715	CMP COLCT	:FIRST COL?
456E C9 98	587	CMP #CONX	:XCHANGE?	4677 F0 0C	716	BEQ ARRL2	:YES
4570 D0 06	588	BNE EM6	:NO	4679 EE 73 43	717	INC COLCT	:INC COL CNTR
4572 20 FD 48	589	JSR XCHG		467C C6 24	718	DEC CH	
4575 4C 3C 45	590	JMP EMENU		467E C6 24	719	DEC CH	:MOVE CUR LFT
4578 C9 B9	591	EM6	:INSERT?	4680 C6 24	720	DEC CH	
457A D0 06	592	BNE EM7	:NO	4682 C6 24	721	DEC CH	
457C 20 61 47	593	JSR INSRT		4684 60	722	RTS	
457F 4C 3C 45	594	JMP EMENU		4685 A5 25	723	ARRL2	:LDA CV
4582 C9 B4	595	EM7	:DELETE?	4687 C9 00	724	CMP #0	
4584 D0 06	596	BNE EM8	:NO	4689 D0 03	725	BNE ARRL3	:NOT TOPLINE
4586 20 00 48	597	JSR DELETE		468B 20 06 47	726	JSR DMSCR	:DOWNSCROLL
4589 4C 3C 45	598	JMP EMENU		468E AD B1 43	727	ARRL3	:LDA RTCHAR
458C 20 3A FF	599	EM8	:ERROR	4691 38	728	SEC	
458F 4C 3C 45	600	JMP EMENU		4692 E9 01	729	SBC #1	:SUB 1
4592	601			4694 B5 24	730	STA CH	:HOR CUR POS
4592	602	LIST RTN		4696 A5 25	731	LDA CV	
4592	603	LIST		4698 38	732	SEC	
4592 A2 15	604	LDX #15	:LINE CNTR	4699 E9 01	733	SBC #1	
4594 AD 70 43	605	LDA BEGADD	:NOW RESET THE	469B 20 5B FB	734	JSR TABV	:RESET VERT CUR
4597 BD 6E 43	606	STA IMADD	:BEGIN ADDRESS TO	469E A9 01	735	LDA #1	:RESET COL CNTR
459A AD 71 43	607	LDA BEGADD+1	:THE IMAGE CNTR	46A0 BD 73 43	736	STA COLCT	
459D BD 6F 43	608	STA IMADD+1		46A3 60	737	RTS	
45A0 E0 00	609	CPX #0	:DONE?	46A4	738		
45A2 D0 01	610	BNE LST0	:NO	46A4	739	DECREMENT POINTERS	
45A4 60	611	RTS		46A4	740		
45A5 A0 00	612	LST0	:ADDRESS OFFSET	46A4 A9 00	741	DECP	:CHK FOR BEG ADDR
45A7 4C AD 45	613	JMP LST1		46A6 C5 F9	742	CMP PNTR	
45AA 20 BE FD	614	JSR CROUT		46A8 D0 09	743	BNE DECP1	
45AD AD B4 43	615	LST1	:LOAD # OF COLS	46AA A9 50	744	LDA /STORE	
45B0 3D 73 43	616	STA COLCT		46AC C5 FA	745	CMP PNTR+1	
45B3 20 EB 45	617	LST2		46AE D0 03	746	BNE DECP1	
45B6 AD 6C 43	618	LST3	:LAST BYTE?	46B0 A0 01	747	LDY #1	:ERROR FLG
45B9 C5 F9	619	CMP PNTR		46B2 60	748	RTS	
45BB D0 08	620	BNE LSTB		46B3 C6 F9	749	DECP1	
45BD AD 6D 43	621	LDA ENDADR+1		46B5 A5 F9	750	DEC PNTR	
45C0 C5 FA	622	CMP PNTR+1		46B7 C9 FF	751	CMP #FF	
45C2 D0 01	623	BNE LSTB		46B9 D0 02	752	BNE DECP2	
45C4 60	624	RTS		46BB C6 FA	753	DEC PNTR+1	
45C5 B1 F9	625	LSTB	:LOAD BYTE	46BD CE 6E 43	754	DECP2	:LDA IMADD
45C7 20 DA FD	626	JSR PRBYTE	:PRINT IT	46C0 AD 6E 43	755	DEC IMADD	
45CA CE 73 43	627	DEC COLCT	:DEC COL CNTR	46C3 C9 FF	756	CMP #FF	
45CD D0 0C	628	BNE LST4	:NOT END OF LINE	46C5 D0 03	757	BNE DECP3	
45CF 20 51 46	629	JSR INCP0	:INC IMADD & PNTR	46C7 CE 6F 43	758	DEC IMADD+1	
45D2 EE BB 43	630	INC TCNTR	:INC SCROLL CNTR	46CA 60	759	DECP3	:RTS
45D5 CA	631	DEX	:DEC LN CNTR	46CB	760		
45D6 E0 00	632	CPX #0	:END OF PAGE?	46CB	761	NORMAL SCROLL RTN	
45D8 D0 D0	633	BNE LST	:NO	46CB	762		
45DA 60	634	RTS	:RETURN FOR CMMD	46CB BE 87 43	763	SCROLL	:SAVE X
45DB E6 24	635	LST4	:MOVE RT 3 SPACES	46CE AD 73 43	764	LDA COLCT	
45DD E6 24	636	INC CH		46D1 BD 86 43	765	STA TCLCT	:STORE COL CT
45DF 20 51 46	637	JSR INCP0	:INC IMADD & PNTR	46D4 A9 00	766	LDA #0	:TEMP CNTR
45E2 EE BB 43	638	INC TCNTR	:INC SCROLL CNTR	46D6 BD 8B 43	767	STA TCNTR	:FOR DECREMENT
45E5 4C B6 45	639	JMP LST3		46D9 A5 24	768	LDA CH	
45EB	640			46DB BD 78 43	769	STA CHT	:SAVE CUR POS
45EB	641	PRINT IMAGE ADDRESS		46DE A2 01	770	LDX #1	:PNT ONE LINE IN LST
45EB	642			46E0 20 AA 45	771	JSR LST	:PNT ONE LINE
45EB AD 6F 43	643	PRTRADR	:PRINT ADDR	46E3 AD 78 43	772	LDA CHT	
45EB 20 DA FD	644	LDA PRBYTE	:PRINT IT	46E6 B5 24	773	STA CH	:RESET CURSOR
45EE AD 6E 43	645	LDA IMADD		46E8 A9 13	774	LDA #13	:ONE LINE UP
45F1 20 DA FD	646	JSR PRBYTE		46EA 20 5B FB	775	JSR TABV	
45F4 A9 A0	647	LDA #SPACE	:PRINT	46ED AE 8B 43	776	LDX TCNTR	
45F6 20 ED FD	648	JSR COUT		46F0 E0 00	777	CPX #0	
45F9 A9 AD	649	LDA #DASH		46F2 F0 08	778	BEQ SCROLL3	
45FB 20 ED FD	650	JSR COUT		46F4 20 B3 46	779	SCROLL2	:JSR DECP1
45FE A9 A0	651	LDA #SPACE		46F7 CA	780	DEX	:RESTORE PNTRS
4600 20 ED FD	652	JSR COUT		46FB E0 00	781	CPX #0	:DONE?
4603 60	653	RTS		46FA D0 F8	782	BNE SCROLL2	:NO
4604	654			46FC AE 87 43	783	SCROLL3	:LDX TEMPX
4604	655	RIGHT ARROW RTN		46FF AD 86 43	784	LDA TCLCT	:RESTORE COL CT
4604	656			4702 BD 73 43	785	STA COLCT	
4604 A0 00	657	ARRR	:ERROR REG	4705 60	786	RTS	
4606 20 49 46	658	JSR INCP	:INCREMENT POINTERS	4706	787		
4609 C0 00	659	CPY #0	:ERROR DETECTED?				
460B F0 06	660	BEQ ARRR1	:NO ERROR				

continued on next page

480A	60	910		RTS		
480B	20	4E	48	JSR MOVMMU		;MOVE MEM UP
480E	CE	6C	43	DEC ENDADR		;DEC LAST ADDR
4811	AD	6C	43	LDA ENDADR		
4814	C9	FF	914	CMP ##FF		;DEC HI BYTE?
4816	D0	03	915	BNE DE1		
4818	CE	6D	43	DEC ENDADR+1		
481B	CE	74	43	FCNT	DE1	;DEC FILE LEN
481E	AD	74	43	LDA FCNT		
4821	C9	FF	919	CMP ##FF		;DEC HI BYTE?
4823	D0	03	920	BNE DE2		;NO
4825	CE	75	43	DEC FCNT+1		
4828	20	41	43	JSR FLDTS	DE2	;PRINT FILE LEN
482B	20	C7	48	JSR RESCUR		;RESTORE POINTERS
482E	20	E9	48	JSR ENDCCHK		
4831	C9	00	926	CMP ##0		;CHK FOR LAST ADDR
4833	D0	04	925	BNE DE7		
4835	20	62	46	JSR ARRL		;MOVE LEFT
4838	60		928	RTS		
4839	A9	15	929	LDA ##15	DE7	;PREP FOR LIST
483B	38		930	SEC		
483C	E5	25	931	SBC CV		
483E	AA		932	TAX		
483F	A0	00	933	LDY #0		;ADDR OFFSET FOR LIST
4841	20	C5	45	JSR LSTB		;LIST
4844	20	C7	48	JSR RESCUR		;RESTORE CURSOR
4847	AD	77	43	LDA CVT		
484A	20	5B	F8	JSR TABV		;RESET VERT CUR
484D	60		938	RTS		
484E			939	#		
484E			940	#		MOVE MEM UP RTN
484E			941	#		
484E	20	A0	48	MOVMMU	JSR SAVCUR	;SAVE POINTERS
4851	A0	01	943	MU1	LDY #1	;OFFSET
4853	B1	F9	944		LDA (PNTR),Y	
4855	B8		945		DEY	
4856	91	F9	946		STA (PNTR),Y	
4858	A5	F9	947		LDA PNTR	;CHK FOR LAST ADDR
485A	CD	6C	43		CMP ENDADR	
485D	D0	0B	949		BNE MU2	
485F	A5	FA	950		LDA PNTR+1	
4861	CD	6D	43		CMP ENDADR+1	
4864	D0	01	952		BNE MU2	
4866	60		953		RTS	
4867	E6	F9	954	MU2	INC PNTR	;INC PNTR
4869	D0	E6	955		BNE MU1	
486B	E6	FA	956		INC PNTR+1	
486D	4C	51	48		JMP MU1	
4870			958	#		
4870			959	#		MOVE RTN
4870			960	#		
4870	20	A0	48	MOVMMU	JSR SAVCUR	;SAVE POINTERS
4873	AD	6C	43		LDA ENDADR	;SET POINTER FOR END
4876	B5	F9	963		STA PNTR	
4878	AD	6D	43		LDA ENDADR+1	
487B	B5	FA	965		STA PNTR+1	
487D	A0	00	966	MV1	LDY #0	;PRESET OFFSET
487F	B1	F9	967		LDA (PNTR),Y	
4881	C8		968		INY	
4882	91	F9	969		STA (PNTR),Y	
4884	A5	F9	970		LDA PNTR	
4886	CD	79	43		CMP TPTR	
4889	D0	0B	972		BNE MV2	;DONE ?
488B	A5	FA	973		LDA PNTR+1	
488D	CD	7A	43		CMP TPTR+1	
4890	D0	01	975		BNE MV2	
4892	60		976		RTS	;MEM MOVE DONE
4893	C6	F9	977	MV2	DEC PNTR	;DEC PNTR
4895	A5	F9	978		LDA PNTR	
4897	C9	FF	979		CMP ##FF	
4899	D0	E2	980		BNE MV1	
489B	C6	FA	981		DEC PNTR+1	
489D	4C	7D	48		JMP MV1	;LOOP BACK
48A0			983	#		
48A0			984	#		SAVE CURSOR POSITION RTN
48A0			985	#		
48A0	A5	24	986	SAVCUR	LDA CH	
48A2	BD	7B	43		STA CHT	
48A5	A5	25	988		LDA CV	
48A7	BD					


```

4912 C9 85 1041    CMP #CONE
4914 F0 40 1042    BEQ X9
4916 C9 88 1043    CMP #LARRW
4918 D0 12 1044    BNE X3
491A AD 72 43 1045    LDA CHCNT
491D C9 01 1046    CMP #1
491F D0 06 1047    BNE X4
4921 20 3A FF 1048    JSR BELL
4924 4C 02 49 1049    JMP X1
4927 C6 24 1050    DEC CH
4929 4C 02 49 1051    JMP X1
492C C9 95 1052    CMP #RARRW
492E D0 04 1053    BNE X7
4930 A4 24 1054    LDY CH
4932 B1 28 1055    LDA (BASL),Y
4934 20 C2 42 1056    JSR VALHEX
4937 C9 FF 1057    CMP #FF
4939 F0 D4 1058    BEQ X2
493B 20 EC 42 1059    JSR STCH
493E AD 72 43 1060    LDA CHCNT
4941 C9 FF 1061    CMP #FF
4943 D0 CA 1062    BNE X2
4945 AD 76 43 1063    LDA TEMP
4948 A0 00 1064    LDY #0
494A 91 F9 1065    STA (PNTR),Y
494C C6 24 1066    DEC CH
494E C6 24 1067    DEC CH
4950 20 04 46 1068    JSR ARRR
4953 4C 02 49 1069    JMP X1
4956 AD 72 43 1070    LDA CHCNT
4959 F0 B4 1071    BEQ X2
495B A9 00 1072    LDA #0
495D BD 85 43 1073    STA RTNFLG
4960 A9 D8 1074    LDA #D8
4962 BD D2 07 1075    STA #D2
4965 60 1076    RTS
4966 1077    #
4966 1078    # SAVE FILE
4966 1079    # $1 IN $3FF1
4966 1080    #
4966 A9 01 1081    SVFILE LDA #1
4968 BD F1 3F 1082    STA $3FF1
496B AD 74 43 1083    LDA FCNT
496E BD F2 3F 1084    STA $3FF2
4971 AD 75 43 1085    LDA FCNT+1
4974 BD F3 3F 1086    STA $3FF3
4977 AD 70 43 1087    LDA BEGADD
497A BD F4 3F 1088    STA $3FF4
497D AD 71 43 1089    LDA BEGADD+1
4980 BD F5 3F 1090    STA $3FF5

;DONE?
;NOT <--
;FIRST CHAR?
;NO
;ERROR
;MOVE LEFT
;RT ARROW?
;LOAD SCREEN CHAR
;VALID?
;NOT VALID
;DONE?
;STORE CHAR
;STORE CHAR
;MOVE CUR BACK
;RT ARRW RTN
;NORMAL "X"
;FOR ON GOTO
;STORE LEN IN
;STORE BEGIN ADDRESS

4983 20 C1 49 1091    JSR CHGD
4986 60 1092    RTS
4987 1093    #
4987 1094    # LOAD FILE RTN
4987 1095    # $1 IN $3FF1
4987 1096    #
4987 A9 02 1097    LDFILE LDA #2
4989 BD F1 3F 1098    STA $3FF1
498C A9 9A 1099    LDA #LD1
498E BD F6 3F 1099    STA $3FF6
4991 A9 49 1101    LDA /LD1
4993 BD F7 3F 1102    STA $3FF7
4996 20 C1 49 1103    JSR CHGD
4999 60 1104    RTS
499A 20 27 41 1105    LD1 JSR NEWFILE
499D 20 CD 41 1106    JSR SETUP
49A0 20 FE 41 1107    JSR SET1
49A3 AD 60 AA 1108    LDA #AA60
49A6 BD 74 43 1109    STA FCNT
49A9 AD 61 AA 1110    LDA #AA61
49AC BD 75 43 1111    STA FCNT+1
49AF 18 1112    CLC
49B0 A9 00 1113    LDA #STORE
49B2 BD 60 AA 1114    ADC #AA60
49B5 BD 6C 43 1115    STA ENADDR
49B8 A9 50 1116    LDA /STORE
49BA AD 61 AA 1117    ADC #AA61
49BD BD 6D 43 1118    STA ENADDR+1
49C0 60 1119    RTS
49C1 1120    #
49C1 1121    # CHG DISK LINE
49C1 1122    #
49C1 20 5B FC 1123    CHGD JSR HOME
49CA A9 0C 1124    LDA #12
49CC 20 5B FB 1125    JSR TABV
49CD A9 09 1126    LDA #9
49CE BD 54 24 1127    STA CH
49CF A9 A4 1128    LDA #CDP
49D0 BD 5D 24 1129    STA PNTADD
49D1 A9 44 1130    LDA /CDP
49D3 BD 5E 24 1131    STA PNTADD+1
49D5 20 BB 44 1132    JSR PRINT
49D8 20 0C FD 1133    JSR RDKEY
49DB 60 1134    RTS
49DC 1135    END

***** END OF ASSEMBLY

```

M.L.E. Driver II

by W.T. Doyle

Physics Department Dartmouth College
Hanover, NH 03755

The Machine Language Editor, Apple M.L.E. (D. Sprinkle, Nibble, Vol. 3/No. 2) is one of those rare utilities that make you wonder how you managed to get by without it. I have used mine constantly since I finally got it entered correctly. If only it could have been used on itself! As I became more dependent upon it however, M.L.E.'s little idiosyncracies became more bothersome. One of these is the awkward method the driver program uses to get a binary file to its destination on the disk. Two copies of every file are saved, and three separate disk transactions are carried out before we get a working copy. A related difficulty becomes apparent when we attempt to continue editing a previously saved file and are asked to specify a starting address. As often as not, I find that I have forgotten it. Since we have just loaded the file into the image space at hex \$5000, the address is not even available at \$AA72, \$AA73.

Quite a different kind of difficulty arises, curiously because M.L.E. works so well! As one begins to pick up touch-typing speed on hexadecimal characters under the benign influence of M.L.E., a new and insidious form of typing error begins to occur. The error is not particularly easy to correct, even with this good editor. I find myself either omitting or adding a character in a pair . . . Often by merely brushing a neighboring key. Then, with M.L.E.'s auto-spacing help, I go on for dozens of character pairs before discovering that all of the characters are being mis-paired. Fortunately, these difficulties may all be dealt with in the driver program, without tampering directly with the machine language portion of M.L.E. itself.

THE DRIVER MOD'

A simple modification of the BSAVE command allows us to save a copy of our binary

file directly from the image space in M.L.E. to the desired destination on the disk. This eliminates the need for duplicate files and reduces the number of disk operations by two thirds. Another modification of the BLOAD command then allows us to load our files directly into M.L.E.'s image space at \$5000, and still have available, the original disk starting address for reference. Since the machine language program requires it in that form, the starting address will be reported in hexadecimal notation.

Finally, a key-click routine that clicks only on the second character of a pair is added to give us an immediate warning that a mis-pairing has occurred. This helps cut down on long strings of mis-paired characters.

HOW IT WORKS

The new drive program, M.L.E. Driver II, is shown in Listing 1. It involves no direct modifications to the M.L.E. machine language program, and no special instructions are needed to use it. As much as possible, it operates just like the original M.L.E. driver.

Line 150 patches in both the key click routine and a short hexadecimal converter for use in the address report. The key click patch uses the Apple bell routine. The value 09 in line 150 controls the duration of the beep. It may be changed to provide a more or a less noticeable click or beep. The modifications to the BSAVE and BLOAD commands are more generally useful. They are discussed in detail elsewhere in this issue.

A CAUTION

One caution: These modifications involve patches that could leave a non-standard version of DOS in memory, if you were to hit the Reset key at an inopportune moment. A warm start, via PR#6, will then restore DOS and should leave your binary file intact in memory.

```

10 REM *****
11 REM # M.L.E. DRIVER II #
12 REM # BY W. T. DOYLE #
13 REM # COPYRIGHT (C) 1982 #
14 REM # BY MICRO-SPARC INC #
15 REM # LINCOLN, MA 01773 #
16 REM *****
100 HIMEM: 16384
110 ONERR GOTO 440
120 REM *** BLOAD M.L.E.D ***
130 LET D$ = CHR$(4): HOME: VTAB 12: HTAB 10: PRINT
    "ONE MOMENT PLEASE..." : PRINT D$:"BLOAD M.L.E.D"
140 REM *** SET UP PATCHES ***
150 LET A$ = "3F5:4C 00 03 N 300:20 7B D0 20 52 E7 A6
    50 A5 51 4C 41 F9 N 430:20 DC 49 N 49DC:CE 72 4
    3 A0 09 20 E4 FB AD 30 DB 60 N DB230"
160 FOR I = 1 TO LEN (A$): POKE 511 + I, ASC ( MID$
    (A$,I,1)) + 128: NEXT I: POKE 72,0: CALL - 144
170 REM *** CALL MLE ***
180 CALL 16384
190 REM *** FILE LOADING ***
200 PRINT: HOME: PRINT D$:"CATALOG": PRINT: PRINT
210 LET N = PEEK (16369): PRINT "ENTER 'EXIT' TO RET
    URN TO MENU": PRINT: PRINT "ENTER FILE NAME ": INPUT
    F$: IF F$ = "" THEN F$ = "NO NAME"
220 IF F$ = "EXIT" THEN 100
230 ON N GOTO 240,320
240 REM *** BSAVE ROUTINE ***
250 LET LZ = PEEK (16370) + 256 # PEEK (16371)
260 LET BA = PEEK (16372) + 256 # PEEK (16373): LET
    LO = BA - 256 # INT (BA / 256): LET HI = INT (B
    A / 256): POKE 6,LO: POKE 7,HI
270 HOME: VTAB 12: HTAB 6: PRINT "BSAVING 'F$':A$"
    I: & BA: PRINT ".L$": & LZ: PRINT
280 POKE 41795,7: POKE 41796,0: POKE 41798,6: POKE 41
    799,0
290 PRINT D$:"BSAVE":F$":A20400,L":LX
300 POKE 41795,115: POKE 41796,170: POKE 41798,114: POKE
    41799,170
310 GOTO 100
320 REM *** BLOAD ROUTINE ***
330 HOME: VTAB 12: HTAB 10: PRINT "BLOADING 'F$'
340 LET AD = 20400:LO = AD - 256 # INT (AD / 256):HI
    = INT (AD / 256): POKE 43634,LO: POKE 43635,HI
350 POKE 41853,6: POKE 41854,0: POKE 41856,7: POKE 41
    857,0
360 PRINT D$:"BLOAD":F$":A20400,L":LX
370 POKE 41853,114: POKE 41854,170: POKE 41856,115: POKE
    41857,170
380 LET BA = PEEK (6) + 256 # PEEK (7)
390 LET LN = PEEK (43616) + 256 # PEEK (43617)
400 HOME: VTAB 12: HTAB 6: PRINT F$":A$": & BA: PRINT
    ".L$": & LN: PRINT "BLOADED": PRINT: PRINT VTAB
    20: HTAB 7: PRINT "PRESS ANY KEY TO CONTINUE": DET
    AB: PRINT A$
410 LET JPZ = PEEK (16374) + 256 # PEEK (16375)
420 CALL JPZ: GOTO 100
430 REM *** ERROR HANDLING ***
440 LET ECX = PEEK (222): HOME: VTAB 11: HTAB 12
    POKE 41795,115: POKE 41796,170: POKE 41798,114: POKE
    41799,170: POKE 41853,114: POKE 41854,170: POKE 4
    1856,115: POKE 41857,170: REM FIX DOS
460 PRINT "DISK ERROR CODE "ECX: PRINT: HTAB 14: INPUT
    "PRESS RETURN ":A$: GOTO 100

```


APPLE CAL



January						
sun	mo	tue	wed	thur	fri	sot
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

APPLE CAL

APPLE CAL The Time Machine

by Glenn Teman
106 Selwyn Road
Newton, MA 02161

Apple CAL is a comprehensive date keeper for planning your schedule, holidays, and appointments days, months, and even years into the future. You can produce a calendar for one month or any range of months, for 1982, 1985, 1989, etc. Moreover, you can have the holidays printed on the calendar for you, and/or you can enter your own important dates — your spouse's birthday, your parents anniversary, the Smith's dinner party, and so on.

APPLE CAL is an Applesoft program requiring at least one disk drive and a printer. Type RUN APPLE CAL to start it up. You will see a title page, and after pressing any key, the following menu:

- 1) PRINT CALENDAR
- 2) ENTER/EDIT IMPORTANT DATES
- 3) ENTER HOLIDAYS
- 4) LIST IMPORTANT DATES
- 5) QUIT

Enter the option of your choice.

THE OPTIONS

PRINT CALENDAR

This option initiates the printing of a calendar. Four questions are asked before the printing starts:

FROM MONTH (JAN):
THRU MONTH (DEC):
YEAR (19XX):
OK? (Y):

For FROM and THRU MONTH you may enter <RETURN> to accept the default (JAN and DEC respectively), or a three letter month abbreviation.

For example, you may enter SEP and <RETURN> to print from September to December or you may enter JUN and JUN to print only

the month of June. Similarly, for the YEAR prompt, you may enter <RETURN> to default to the year in parentheses or you may enter any year between 1981 and 2000. The default year is the year of the "date file" which is currently loaded into the APPLE's memory. If you enter a different year, you will be notified: 19XX FILE NOT IN MEMORY, and will be further asked, "LOAD FROM DISK?". Answer Y to load in a previously created "date file" for that year or answer N to clear the file in memory and proceed without any "date file". Finally, you will be asked OK? and if you answer Y or <RETURN>, the printing will proceed. **To any of these prompts, if you enter an up-arrow, "/", the option will be aborted and you will be returned to the menu.**

A sample of the calendar which is printed follows. Note that the year and month appear on top, small calendars appear on the bottom for the previous and following months, and holidays and important dates appear at the bottom of the appropriate day's box. Long dates are split in the middle, hyphenated, and are displayed on two lines. If a date is entered with a slash (/) in it, it is divided into two dates at the point of the slash and each is preceded by a dot.

ENTER/EDIT IMPORTANT DATES

This option is used to enter or change the dates in your date file. You will be asked for a YEAR, a MONTH, a DAY, and the TEXT of the date. The YEAR will default to the year of the date file in memory and, as with the PRINT CALENDAR option, you will be asked if you wish to LOAD the file from disk if this year is not in memory. Again, an up-arrow will get you back to the menu. This option will not allow you to enter an invalid day, for example MAY 32 or NOVEMBER 31.

At the TEXT prompt, if there is already a date stored for the entered year-month-day, its text will appear below the dotted line on which you will enter your text. If you enter <RETURN>, the old text will be preserved. If you enter an asterisk (*), that date will be deleted. You may enter any text characters so long as the length of the text is less than 19 characters and as long as there are no embedded slashes (/). If there is an embedded slash, this will indicate to APPLE CAL that you wish to enter 2 dates for this year-month-day, and each slash-piece must not be more than 9 characters long. If there is some default text present, any text you enter will replace that text.

When you are finished entering/editing text, enter an up-arrow (). You will be asked if you wish to SAVE FILE TO DISK, and if you answer Y, a disk file will be created called CAL.FILE-19XX where XX is the year you were working on. This file may later be recalled for editing or printing, of course.

ENTER HOLIDAYS

ENTER HOLIDAYS is used to enter some or all of the holidays for a given year into the date file. **If the holidays are to be entered for a year prior to 1987, the dates for these holidays will be obtained from APPLE CAL and displayed as defaults while each holiday is entered.** For 1987 or beyond, the name of the holiday will be presented, but you must enter the day that it occurs on. However, for such days as NEW YEAR's DAY and CHRISTMAS which occur always on the same date in the month, that day will be given for any year. The days given for holidays that fall on a different day each year will usually be the MONDAY after the

holiday, which we usually have as a vacation day.

You will be asked for a YEAR. As with ENTER/EDIT IMPORTANT DATES, if the year entered does not match the year of the date file in memory, you will be asked if you wish to load that file from disk. A holiday's name will be displayed, then its month, and you will either be asked for the day, or the day will be displayed (if the year is prior to 1987). You then will be asked **OK?**. Enter **Y** if you wish this holiday to appear on your calendar. If this day matches a day for which you already have a date saved, you will be shown the date and asked if you wish to **REPLACE** it. Answer **Y** or **N**. If you are asked for a day and you do not wish this holiday to appear on your calendar, enter an asterisk (*). When you have entered all the holidays or if you enter an up-arrow, you will be asked if you wish to save this date file, as in ENTER/EDIT IMPORTANT DATES, above.

LIST IMPORTANT DATES

This option is used to view on your monitor or print the date file. Answer the YEAR, FROM MONTH, and TO MONTH questions as before. If necessary, you will be asked if you wish to load a date file from disk. Then you will be asked **PRINTER OR SCREEN**. Enter **P** to print a list of dates on a printer in APPLE SLOT 1 or enter **S** to view the list on your screen.

QUIT

Use this option to exit APPLE CAL. If you have not saved the date file on which you were working, do so (go into ENTER/EDIT IMPORTANT DATES and type an up-arrow at your first opportunity), because QUIT will not do it for you. You will be asked **ARE YOU SURE?** before APPLE CAL says GOODBYE.

HOW APPLE CAL TICKS

APPLE CAL starts by initializing some important variables:

N\$(12,31)	holds text for a month, day
MO\$(12)	holds names of months
LS	CHR\$(124) = I
DS	CHR\$(4) for DOS
DA\$	String of 2 digit entries for the number of days in each month
SLOT	Printer slot — starts at 1

Two functions are defined, also: one is a modulus-4 check for leap year testing and the other is a modulus-7 check. Then, a title page is displayed with a GOSUB 20000. The moving-asterisk marquee-like effect is created by FOR loops printing a space and an asterisk alternately. A keypress is checked for while printing each side of the marquee, and if it is detected, this causes a RETURN from the subroutine.

On line 115, a GOSUB 20400 is performed which sets up a short machine language routine from the APPLE REFERENCE MANUAL (page 136) which is intended to handle **STACK** problems caused by errors handled by an ONERR GOTO command. There are a lot of GOSUB's and FOR loops in APPLE CAL, and each makes return address entries onto the STACK. If an error occurs a few times in a row, the stack has a tendency to get messed up. An **OUT OF MEMORY** error may appear, for example, if you answer the LOAD FROM DISK question several times with YES and APPLE CAL tries to load a non-existent file

from disk several times in a row. The stack will be fine for a couple of commands and the error will be handled correctly by the error handling routine ("FILE NOT FOUND"), but then the system may do strange things. **If an error should occur, type GOTO 500 to restart without losing the variables set up.** Lastly, when the program starts, an attempt is made to load the 1981 date file from disk. The ONERR on line 130 protects against this file being undefined. The error handling routine starting on line 11000 is meant to handle just disk input/output errors.

The **MENU** is displayed from line 500. On line 620, an **ON I GOTO** is executed instead of an **ON I GOSUB** because the **STACK** was filling up with GOSUB's and nested FOR loops needed to print the calendar. This is also why several loops are "hard-coded" instead of using a FOR/NEXT loop. Because of the **ON I GOTO**, each option must execute a GOTO 500 when it has finished its job.

The **PRINT CALENDAR** routine starts on line 1000. To achieve a formatted screen, all of the prompts are printed together (lines 1020 to 1050). Then the questions are asked one by one. On line 1235, if the YEAR entered does not match the year loaded into memory, a GOSUB 7000 is executed to ask "LOAD FROM DISK?". On line 7060, ONERR is enabled in case the requested file is not on the disk. As can be seen, the date file is simple:

12 (months) * 31 (days) = 372 sequential data fields, mostly null, but some with text. The file's name is "CAL.FILE-19XX" where XX is the second part of the year chosen.

Whenever months are entered, a GOSUB 1280 is executed to check the syntax of the input against the string of month abbreviations (line 1310) and return the number corresponding to the month. ERR is set equal to 1 if the input is invalid.

THE DAY OF THE WEEK

The next subroutine is called from line 1240 with a GOSUB 1360. It is one of the most important routines because it calculates the day of the week for the first day of the month (for the year entered). The variable **KK** is started at 5 because Jan 1, 1981 is a Thursday, the 5th day of the week. Then, the number of years between 1981 and the year entered is calculated and 365 times this number is added to **KK** (lines 1390 to 1420) with provision made for leap years (GOSUB 2600, line 1410). The **leap year calculation** returns the variable **LEAP** = 0 or 1, depending on whether a leap year has occurred (i.e., if the year MOD 4 = 0 and the year MOD 100 NOT = 0). Next, the months between JANUARY and the FROM MONTH entered are calculated to add the correct number of days (from the **DA\$** string) to **KK** (lines 1450 to 1470), again with consideration to the months of FEBRUARY which have extra leap year days. Lastly, **KK** (the Day) is calculated MOD 7 to find what day of the week it is (line 1480).

PRINTING THE CALENDAR

Armed with this information, the calendar is ready to be printed. Between lines 1500 and 1570, the Program loops through the FROM/THRU MONTHS to print each month. A GOSUB 2240 is executed to print the previous and following months at the bottom of each calendar. The actual printing starts on line 1580. Most of this code is self-explanatory.

On line 1750, the **DAY** variable (from **KK**) is negated and 2 is added to it. Thus, if it were 3 (= TUESDAY), it becomes -1. It is incremented for each day. When it becomes > 0, the day numbers start (with 1) so that the first day will fall on the correct weekday (in this case, TUESDAY). Three nested FOR loops control the printing: 1) Line 1760 for printing the 6 boxes vertically; 2) Line 1770 for printing the six lines per box; and 3) for printing the 7 days horizontally. The latter is actually three loops, one used only for **RR**=1 (first line) to print day numbers (line 1790), one used for **RR**=2 to **RR**=4 to print spaces (line 1880), and one used for **RR**=5 and **RR**=6 to print any date text that is stored in the **N\$** array by month and day (line 1920). The last loop, if it detects a non-null **N\$** value, does a GOSUB 1990 to print the calendar "notes". This routine splits the text into two lines if it is too long and looks for an imbedded slash which indicates two separate dates. It then prints the appropriate text, depending if **RR** is on line 5 or 6. When the calendar is finished, the bottom is printed with the GOSUB 2240. The routine at 1360 is used to find when the first day of the previous and following months occur, and loops at 2320, 2370, and 2450 print these months.

ENTERING DATA

ENTER/EDIT IMPORTANT DATES starts at line 3000. Any dates entered are checked to ensure that they are valid (lines 3170 to 3190). If text is present for that date, it is displayed. On lines 3250 to 3280, length checks are performed on the text. This routine is not complicated nor is it very long.

The **ENTER HOLIDAYS** routine starts on line 4000. All entries are based on the year under consideration. If it is less than 1987, a computation is done (line 4095) to yield a number which can be used to set the **DATA** statement pointer to the correct place. Otherwise, this number is 0. A loop is executed to read the **DATA** statements which start at line 4500. Their structure is as follows:

There are 17 dates per year. Each is a padded day concatenated to a holiday, concatenated to a padded month number. There may be no padded day if it is unknown (for years > 1986).

As the **DATA** is read, the month and day are displayed if they are known, and the user is asked "OK?". If he or she says YES, the holiday is filed into **N\$(month,day)**. If the day is 0 (line 4130), an exact date is sought from the user. This two-way branch is decided on line 4130. If a day is known, the program proceeds to line 4300. Otherwise it continues on to line 4140. When the user is finished entering holidays, he or she is asked "SAVE FILE TO DISK?". If the answer is YES, a GOSUB 6000 is executed to handle the save. The **ONERR** flag is also set in this routine.

The **LIST DATES** routine which starts on line 5000 is short and simple. After getting the YEAR and FROM/TO months (converted to numbers), it asks the user whether he or she wants to PRINT or view on the SCREEN. Then it uses the **N\$** array to print the important dates.

Lastly, **QUIT** starts on line 10000. It asks "ARE YOU SURE?" and says GOODBYE if you are. No file is saved by this routine.

continued on next page

1

```

1 REM *****
2 REM * APPLE CAL *
3 REM * BY GLENN TEMAN *
4 REM * COPYRIGHT (C) 1982 *
5 REM * BY MICRO-SPARC INC *
6 REM * LINCOLN, MA. 01773 *
7 REM *****
10 DIM N$(12,31),M$(12)
20 L$ = CHR$(124):D$ = CHR$(4)
30 DA$ = "3128313031303130313031":SLOT = 1
40 FOR I = 1 TO 12: READ A$:M$(I) = A$: NEXT I
45 DATA JANUARY,FEBRUARY,MARCH,APRIL,MAY,JUNE,JULY,AUG
UST,SEPTEMBER,OCTOBER,NOVEMBER,DECEMBER
60 DEF FN M(X) = ((X / 4 - INT (X / 4)) * 4): DEF FN
W(X) = ((X / 7 - INT (X / 7)) * 7)
100 GOSUB 20000: REM TITLE PAGE
110 HOME: VTAB 10: HTAB 11: PRINT "(JUST A MOMENT...)"

115 GOSUB 20400
120 YE = 1981:NDO = 1
130 ONERR GOTO 11000
140 SAV = 0
150 Y = YE: GOSUB 7070:Y = 0
160 POKE 216,0
170 NDO = 0
500 REM ** MENU **
510 HOME: PRINT: PRINT
515 INVERSE
520 PRINT "APPLE CAL": HTAB 24: PRINT "BY GLENN TEMAN"

525 NORMAL
530 VTAB 8
540 PRINT "1) PRINT CALENDAR"
550 VTAB 10: PRINT "2) ENTER/EDIT IMPORTANT DATES"
560 VTAB 12: PRINT "3) ENTER HOLIDAYS"
570 VTAB 14: PRINT "4) LIST IMPORTANT DATES"
580 VTAB 16: PRINT "5) QUIT"
590 VTAB 22: INVERSE: PRINT "CHOICE: ";: NORMAL
600 INPUT " ";A$:I = VAL (A$)
610 IF I < 1 OR I > 5 THEN PRINT CHR$(7): GOTO 510
620 ON I GOTO 1000,3000,4000,5000,10000
1000 REM ** PRINT CALENDAR **
1010 A$ = "PRINT CALENDAR": GOSUB 8000
1020 VTAB 8: HTAB 1: PRINT "FROM MONTH (JAN): ";: CALL
- 868
1030 VTAB 10: HTAB 1: PRINT "THRU MONTH (DEC): ";: CALL
- 868
1040 VTAB 12: HTAB 1: PRINT "YEAR (":YE:"): ";: CALL -
868
1050 VTAB 14: HTAB 1: PRINT "OK? (Y): ";: CALL -
868
1060 VTAB 8: HTAB 19: CALL - 868: INPUT "":A$
1070 IF A$ = "" THEN 500
1080 IF A$ = "Y" THEN A$ = "JAN": VTAB 8: HTAB 19: PRINT
"JAN"
1090 GOSUB 1200: IF ERR THEN 1060
1100 F = VAL (A$)
1110 VTAB 10: HTAB 19: CALL - 868: INPUT "":A$
1120 IF A$ = "" THEN 500
1130 IF A$ = "Y" THEN A$ = "DEC": VTAB 10: HTAB 19: PRINT
"DEC"
1140 GOSUB 1200: IF ERR THEN 1110
1150 T = VAL (A$)
1160 IF F > T THEN PRINT CHR$(7): GOTO 1060
1170 VTAB 12: HTAB 14: CALL - 868: INPUT "":Y$
1180 IF Y$ = "" THEN 500
1190 IF Y$ = "Y" THEN Y$ = STR$(YE): VTAB 12: HTAB 14:
PRINT Y$
1200 Y = VAL (Y$): IF Y < 1981 OR Y > 2000 THEN PRINT
CHR$(7): GOTO 1170
1210 VTAB 14: HTAB 15: CALL - 868: INPUT "":A$
1220 IF A$ < " " AND LEFT$(A$,1) < "Y" THEN 500
1230 VTAB 14: HTAB 15: PRINT "YES"
1233 REM LOAD FROM DISK?
1235 IF YE < Y THEN YE = Y: GOSUB 7000
1240 GOSUB 1360: REM FIND 1ST DAY OF MONTH
1250 GOTO 1500
1255 PRINT D$;"PR00"
1260 GOTO 500
1270 REM
1280 REM ** MONTH ERROR **
1290 ERR = 0
1300 FOR I = 1 TO 12
1310 IF A$ = MID$( "JANFEBMARAPR MAYJUNJUL AUGSEP OCTNOV
DEC",I * 3 - 2,3) THEN 1340
1320 NEXT I
1330 PRINT CHR$(7):ERR = 1: RETURN
1340 A$ = STR$(I): RETURN

```

```

1350 REM
1360 REM ** LOOP ON MONTHS **
1370 KK = 5
1380 YR = Y - 1981 - 1: IF YR < 0 THEN 1430
1390 FOR I = 0 TO YR
1400 KK = KK + 365
1410 YY = 1981 + I: GOSUB 2600:KK = KK + LEAP
1420 NEXT
1430 MN = F - 1: IF MN < 1 THEN 1480
1450 FOR I = 1 TO MN
1460 KK = KK + VAL ( MID$( DA$,I * 2 - 1,2))
1465 IF I = 2 THEN YY = Y: GOSUB 2600:KK = KK + LEAP: REM
FEB IN LEAP YR
1470 NEXT
1480 DAY = INT ( FN W(KK) + .5): REM WEEKDAY OF 1ST DA
Y OF MONTH
1485 IF DAY = 0 THEN DAY = 7
1490 RETURN
1500 M = F - 1
1505 M = M + 1: IF M > T THEN 1570
1510 GOSUB 1500
1515 KH = KK: GOSUB 2240: REM LAST & NEXT MONTH
1520 KK = KH + VAL ( MID$( DA$,M * 2 - 1,2))
1530 IF M = 2 THEN YY = Y: GOSUB 2600:KK = KK + LEAP
1540 DAY = INT ( FN W(KK) + .5)
1545 IF DAY = 0 THEN DAY = 7
1550 PRINT: PRINT: PRINT: PRINT: PRINT: PRINT: REM
NEXT PAGE
1560 GOTO 1505
1570 GOTO 1255
1580 REM ** PRINTING **
1590 PRINT: PRINT D$;"PR00";SLOT
1600 PRINT: PRINT: PRINT
1610 M$ = M$(M):L = LEN (M$) * 2:NDO = VAL ( MID$( DA$,
M * 2 - 1,2))
1620 IF M = 2 THEN YY = Y: GOSUB 2600:NDO = NDO + LEAP
1630 S = (78 - L - 3) / 2
1640 PRINT TAB( S): LEFT$ ( "*****",L +
3)
1650 PRINT "APPLE J": CHR$(91):" ";
1660 PRINT TAB( S):" ";
1670 FOR I = 1 TO L / 2: PRINT MID$( M$,I,1):" "; NEXT
I: PRINT " ";
1680 PRINT SPC( 67 - S - L):Y
1690 PRINT TAB( S): LEFT$ ( "*****",L +
3)
1700 PRINT: FOR I = 1 TO 78: PRINT "=: NEXT
1710 PRINT: PRINT L$;" SUN ";L$;" MON ";
1720 PRINT L$;" TUE ";L$;" WED ";L$;" THR
";L$;
1730 PRINT " FRI ";L$;" SAT ";L$;
1740 FOR I = 1 TO 78: PRINT "=: NEXT: PRINT
1750 DAY = - DAY + 2
1760 FOR R = 1 TO 6
1770 FOR RR = 1 TO 6
1780 PRINT L$;: IF RR > 1 THEN 1870
1790 FOR I = 1 TO 7
1800 IF DAY < 1 OR DAY > NDO THEN PRINT " ": GOTO 183
0
1810 IF DAY < 10 THEN PRINT " ";
1820 PRINT DAY;
1830 DAY = DAY + 1
1840 PRINT " ";L$;
1850 NEXT I
1860 GOTO 2130
1870 IF RR > 4 THEN 1920
1880 FOR I = 1 TO 7
1890 PRINT " ";L$;
1900 NEXT I
1910 GOTO 2130
1920 FOR D = DAY - 7 TO DAY - 1
1930 IF D < 1 OR D > NDO THEN SP = 10: GOTO 1960
1940 SP = 10
1950 IF N$(M,D) < " " THEN GOSUB 1990:SP = LFT
1960 PRINT SPC( SP):L$;
1970 NEXT D
1980 GOTO 2130
1990 REM ** NOTES ON CAL **
2000 A$ = N$(M,D)
2005 L = 0
2007 L = L + 1: IF L > LEN (A$) THEN 2030
2010 IF MID$( A$,L,1) = "/" THEN 2110
2020 GOTO 2007
2030 IF LEN (A$) < 10 THEN LFT = 10: IF RR = 6 THEN PRINT
A$;:LFT = 10 - LEN (A$)
2035 IF LEN (A$) < 10 THEN RETURN
2040 L = LEN (A$) / 2:LFT = 10 - L
2050 IF L < INT (L) THEN 2080
2060 IF RR = 5 THEN PRINT LEFT$( A$,L):"-":LFT = LFT
- 1: RETURN
2070 PRINT RIGHT$( A$,L):": RETURN
2080 IF RR = 5 THEN PRINT LEFT$( A$,L):"-": RETURN

```

continued on next page

APPLE CAL

The Time Machine (Cont.)

```

2090 PRINT RIGHT$(A$,L+1); RETURN
2100 REM 2 LINES OF DATES (?)
2110 IF RR = 5 THEN PRINT ". "; LEFT$(A$,L-1);:LFT =
10 - L: RETURN
2120 PRINT ". "; RIGHT$(A$, LEN(A$) - L);:LFT = 9 - LEN
(A$) + L: RETURN
2130 PRINT
2140 NEXT RR
2150 PRINT L$;
2160 FOR I = 1 TO 7
2170 PRINT "-----";
2180 IF I < 7 THEN PRINT "+";
2190 IF I = 7 THEN PRINT L$
2200 NEXT I
2210 NEXT R
2220 RETURN
2230 REM
2240 REM ** PRINT LAST & NEXT MONTH **
2250 IF M = 1 AND Y = 1981 THEN D1 = - 2 + 2:N1 = 31::
GOTO 2290: REM DEC '80
2260 F = M - 1: IF F < 1 THEN F = 12:Y = Y - 1
2270 GOSUB 1360: IF F = 12 THEN Y = Y + 1
2280 D1 = - DAY + 2:N1 = VAL ( MID$(DA$,F * 2 - 1,2))
: IF F = 2 THEN YY = Y: GOSUB 2600:N1 = N1 + LEAP
2290 F = M + 1: IF F > 12 THEN F = 1:Y = Y + 1
2300 GOSUB 1360: IF F = 1 THEN Y = Y - 1
2310 D2 = - DAY + 2:N2 = VAL ( MID$(DA$,F * 2 - 1,2))
: IF F = 2 THEN YY = Y: GOSUB 2600:N2 = N2 + LEAP
2320 FOR I = 1 TO 6
2330 IF I = 3 THEN PRINT " * NIBBLE MAGAZINE * ";: GOTO
2360
2340 IF I = 4 THEN PRINT " (C) GLENN TEMAN 1981";: GOTO
2360
2350 PRINT SPC( 23);
2360 PRINT SPC( 4);
2370 FOR J = 1 TO 7
2380 IF D1 < 1 THEN PRINT " ";:D1 = D1 + 1: GOTO 243
0
2400 IF D1 > N1 THEN PRINT " ";: GOTO 2430
2410 IF D1 < 10 THEN PRINT " ";
2420 PRINT " ";:D1::D1 = D1 + 1
2430 NEXT J
2440 PRINT SPC( 5);
2450 FOR J = 1 TO 7
2460 IF D2 < 1 THEN PRINT " ";:D2 = D2 + 1: GOTO 251
0
2480 IF D2 > N2 THEN PRINT " ";: GOTO 2510
2490 IF D2 < 10 THEN PRINT " ";
2500 PRINT " ";:D2::D2 = D2 + 1
2510 NEXT J
2520 PRINT
2530 NEXT I
2540 PRINT SPC( 35);MO$(M - 1 + 12 * (M = 1));
2545 I = LEN(MO$(M - 1 + 12 * (M = 1)))
2550 PRINT SPC( 60 - 35 - I);MO$(M + 1 - 12 * (M = 12)
)
2560 RETURN
2600 REM ** LEAP YR? **
2610 LEAP = 0
2620 IF INT ( FN M(YY) + .5) = 0 AND YY / 100 ( ) INT
(YY / 100) THEN LEAP = 1
2630 RETURN
3000 REM ** ENTER/EDIT IMPORTANT DATES **
3010 A$ = "ENTER/EDIT IMPORTANT DATES": GOSUB 8000:L = 0
3020 VTAB 8: HTAB 1: CALL - 868: PRINT "YEAR (*;YE;*):
"
3030 VTAB 12: HTAB 1: CALL - 868: PRINT "MONTH:"
3040 VTAB 14: HTAB 1: CALL - 868: PRINT "DAY:"
3050 VTAB 16: HTAB 1: CALL - 868: PRINT "TEXT:"
3060 IF L THEN 3110
3070 VTAB 8: HTAB 14: CALL - 868: INPUT "Y$;
3080 IF Y$ = "" THEN Y$ = STR$(YE): VTAB 8: HTAB 14: PRINT
Y$;
3090 IF Y$ = "" THEN 500
3100 Y = VAL (Y$): IF Y < 1981 OR Y > 2000 THEN PRINT
CHR$(7);: GOTO 3070
3105 IF Y < > YE THEN GOSUB 7000:YE = Y
3110 VTAB 12: HTAB 8: CALL - 868: INPUT "Y$;A$
3120 IF A$ = "" THEN 3400
3130 GOSUB 1280: IF ERR THEN 3110
3140 M = VAL (A$)
3150 VTAB 14: HTAB 6: CALL - 868: INPUT "Y$;A$
3160 IF A$ = "" THEN 500
3170 DAY = VAL (A$): IF DAY = 0 THEN PRINT CHR$(7);:
GOTO 3150
3180 D = VAL ( MID$(DA$,M * 2 - 1,2)): IF DAY > D AND
M < > 2 THEN PRINT CHR$(7);: GOTO 3150
3190 IF M = 2 THEN YY = Y: GOSUB 2600: IF DAY > D + LEA
P THEN PRINT CHR$(7);: GOTO 3150
3200 Y$ = "": IF N$(M,DAY) < > "" THEN Y$ = N$(M,DAY): VTAB
17: HTAB 7: CALL - 868: PRINT "(*;Y$;*);
3210 VTAB 16: HTAB 7: CALL - 868: PRINT ".....
....."; HTAB 7: INPUT "Y$;A$

```

```

3220 IF A$ = "" AND Y$ < > "" THEN CALL - 868: VTAB
16: HTAB 7: PRINT Y$;: GOTO 3300
3225 IF A$ = "" THEN CALL - 868: PRINT "(DELETED)";:
N$(M,DAY) = "": GOTO 3300
3230 IF A$ = "" THEN 3400
3240 IF A$ = "" THEN 3300
3250 IF LEN (A$) > 18 THEN PRINT CHR$(7);: GOTO 320
0
3260 FOR I = 1 TO LEN (A$): IF MID$(A$,I,1) = "/" THEN
3280
3270 NEXT I: GOTO 3290
3280 IF I > 10 OR LEN (A$) - I > 9 THEN PRINT CHR$(
7);: GOTO 3200
3290 N$(M,DAY) = A$
3300 FOR I = 1 TO 750: NEXT I
3310 L = 1
3320 VTAB 11: CALL - 958: GOTO 3030
3400 VTAB 20: CALL - 958: INPUT "SAVE FILE TO DISK? ";
A$
3410 IF LEFT$(A$,1) = "Y" THEN GOSUB 6000
3420 GOTO 500
4000 REM ** ENTER HOLIDAYS **
4010 A$ = "ENTER HOLIDAYS": GOSUB 8000
4020 VTAB 8: HTAB 1: PRINT "YEAR (*;YE;*);"
4030 VTAB 12: HTAB 1: PRINT "HOLIDAY:"
4040 VTAB 14: HTAB 5: PRINT "MONTH:";: HTAB 28: PRINT "
DAY:"
4060 VTAB 8: HTAB 14: CALL - 868: INPUT "Y$;
4070 IF Y$ = "" THEN Y$ = STR$(YE): VTAB 8: HTAB 14: PRINT
Y$;
4080 IF Y$ = "" THEN 500
4090 Y = VAL (Y$): IF Y < 1981 OR Y > 2000 THEN PRINT
CHR$(7);: GOTO 4060
4092 IF Y < > YE THEN GOSUB 7000:YE = Y
4095 R = 0: IF Y < 1987 THEN R = (6 + Y - 1986) * 17
4100 RESTORE : FOR I = 1 TO 12 + R: READ A$: NEXT
4120 FOR I = 1 TO 17
4130 READ A$: IF VAL (A$) > 0 THEN 4300
4140 M = VAL ( RIGHT$(A$,2)):A$ = LEFT$(A$, LEN (A$)
- 2)
4150 VTAB 12: HTAB 10: PRINT A$;
4160 VTAB 14: HTAB 13: PRINT MO$(M);
4170 VTAB 14: HTAB 34: CALL - 868: INPUT "Y$;
4180 IF Y$ = "" THEN 4400: REM SAVE?
4190 IF Y$ = "" THEN 4410: REM NEXT
4200 DAY = VAL (Y$): IF DAY = 0 THEN PRINT CHR$(7);:
GOTO 4170
4210 D = VAL ( MID$(DA$,M * 2 - 1,2)): IF DAY > D AND
M < > 2 THEN PRINT CHR$(7);: GOTO 4170
4220 IF M = 2 THEN YY = Y: GOSUB 2600: IF DAY > D + LEA
P THEN PRINT CHR$(7);: GOTO 4170
4230 IF N$(M,DAY) = "" THEN 4400: REM FILE
4240 VTAB 20: HTAB 1: CALL - 868: PRINT "REPLACE *;N$(
M,DAY)
4250 PRINT " WITH *;A$;: INPUT " OK? ";B$
4260 IF LEFT$(B$,1) = "Y" THEN 4400: REM FILE
4270 VTAB 21: CALL - 868: PRINT " (NOT REPLACED)";
4280 GOTO 4410: REM NEXT
4300 M = VAL ( RIGHT$(A$,2)):DAY = VAL ( LEFT$(A$,2)
):A$ = MID$(A$,3, LEN (A$) - 4)
4310 VTAB 12: HTAB 10: PRINT A$;
4320 VTAB 14: HTAB 13: PRINT MO$(M);
4330 HTAB 34: PRINT DAY
4340 IF N$(M,DAY) = "" OR A$ = N$(M,DAY) THEN 4370
4350 VTAB 20: HTAB 1: CALL - 868: PRINT "REPLACE *;N$(
M,DAY)
4360 PRINT " WITH *;A$;
4370 INPUT " OK? (Y) ";B$
4375 IF B$ = "" THEN 4400
4380 IF LEFT$(B$,1) = "Y" OR B$ = "" THEN 4400: REM
FILE
4390 VTAB 21: CALL - 868: PRINT " (NOT FILED)";: FOR
J = 1 TO 500: NEXT J: GOTO 4410: REM NEXT
4400 N$(M,DAY) = A$
4410 VTAB 10: CALL - 958: VTAB 12: HTAB 1: PRINT "HOLI
DAY:"
4420 VTAB 14: HTAB 5: PRINT "MONTH:";: HTAB 28: PRINT "
DAY:"
4430 NEXT I
4440 VTAB 20: CALL - 958: INPUT "SAVE FILE TO DISK? ";
A$
4450 IF LEFT$(A$,1) = "Y" THEN GOSUB 6000
4460 GOTO 500
4500 REM >1986
4510 DATA 01NEW YEAR'S DAY01,14VALENTINE'S DAY02,WASHI
NGTON'S BDAY02,17ST PATRICK'S DAY03,PASSOVER04,EAST
ER04,MOTHER'S DAY05,MEMORIAL DAY05,FATHER'S DAY06
4520 DATA 04INDEPENDENCE DAY07,LABOR DAY09,COLUMBUS DAY
10,31HALLOWEEN10,11VETERAN'S DAY11,THANKSGIVING11,H
ANUKKAH12,25CHRISTMAS12
4530 REM 1981
4540 DATA 01NEW YEAR'S DAY01,14VALENTINE'S DAY02,14WAS
HINGTON'S BDAY02,17ST PATRICK'S DAY03,19PASSOVER04,
19EASTER04,10MOTHER'S DAY05,25MEMORIAL DAY05,21FATH
ER'S DAY06
4550 DATA 04INDEPENDENCE DAY07,07LABOR DAY09,12COLUM
BUS DAY10,31HALLOWEEN10,11VETERAN'S DAY11,26THANKSG
IVING11,21HANUKKAH12,25CHRISTMAS12
4560 REM 1982

```



```

4570 DATA @1NEW YEAR'S DAY@1,14VALENTINE'S DAY@2,15WA
SHINGTON'S BDAY@2,17ST PATRICK'S DAY@3,08PASSOVER@4
,11EASTER@4,09MOTHER'S DAY@5,31MEMORIAL DAY@5,20FAT
HER'S DAY@6
4580 DATA @4INDEPENDENCE DAY@7,06LABOR DAY@9,11COLU
BUS DAY@10,31HALLOWEEN@10,11VETERAN'S DAY@11,25THANKS
GIVING@11,01HANUKKAH@12,25CHRISTMAS@12
4590 REM 1983
4600 DATA @1NEW YEAR'S DAY@1,14VALENTINE'S DAY@2,21W
ASHINGTON'S BDAY@2,17ST PATRICK'S DAY@3,29PASSOVER@
3,08EASTER@4,08MOTHER'S DAY@5,30MEMORIAL DAY@5,19FA
THER'S DAY@6
4610 DATA @4INDEPENDENCE DAY@7,05LABOR DAY@9,10COLU
MBUS DAY@10,31HALLOWEEN@10,11VETERAN'S DAY@11,24THANKS
GIVING@11,01HANUKKAH@12,25CHRISTMAS@12
4620 REM 1984
4630 DATA @1NEW YEAR'S DAY@1,14VALENTINE'S DAY@2,20W
ASHINGTON'S BDAY@2,17ST PATRICK'S DAY@3,17PASSOVER@
4,22EASTER@4,13MOTHER'S DAY@5,28MEMORIAL DAY@5,17FA
THER'S DAY@6
4640 DATA @4INDEPENDENCE DAY@7,03LABOR DAY@9,08COLU
MBUS DAY@10,31HALLOWEEN@10,11VETERAN'S DAY@11,22THANKS
GIVING@11,19HANUKKAH@12,25CHRISTMAS@12
4650 REM 1985
4660 DATA @1NEW YEAR'S DAY@1,14VALENTINE'S DAY@2,18W
ASHINGTON'S BDAY@2,17ST PATRICK'S DAY@3,06PASSOVER@
4,07EASTER@4,12MOTHER'S DAY@5,27MEMORIAL DAY@5,16FA
THER'S DAY@6
4670 DATA @4INDEPENDENCE DAY@7,02LABOR DAY@9,14COLU
MBUS DAY@10,31HALLOWEEN@10,11VETERAN'S DAY@11,28THANKS
GIVING@11,08HANUKKAH@12,25CHRISTMAS@12
4680 REM 1986
4690 DATA @1NEW YEAR'S DAY@1,14VALENTINE'S DAY@2,17W
ASHINGTON'S BDAY@2,17ST PATRICK'S DAY@3,24PASSOVER@
4,30EASTER@4,11MOTHER'S DAY@5,26MEMORIAL DAY@5,15FA
THER'S DAY@6
4700 DATA @4INDEPENDENCE DAY@7,01LABOR DAY@9,13COLU
MBUS DAY@10,31HALLOWEEN@10,11VETERAN'S DAY@11,27THANKS
GIVING@11,27HANUKKAH@12,25CHRISTMAS@12
5000 REM ** LIST DATES **
5010 A$ = "LIST DATES": GOSUB 8000
5020 VTAB 8: HTAB 1: PRINT "YEAR (*;YE;*):"
5030 VTAB 10: HTAB 1: PRINT "FROM MONTH (JAN):"
5040 VTAB 12: HTAB 1: PRINT "TO MONTH (DEC):"
5050 VTAB 14: HTAB 1: PRINT "PRINTER OR SCREEN:"
5060 VTAB 8: HTAB 14: CALL - 868: INPUT "":Y$
5070 IF Y$ = "" THEN 500
5080 IF Y$ = "" THEN Y$ = STR$(YE): VTAB 8: HTAB 14: PRINT
Y$
5090 Y = VAL (Y$): IF Y < 1981 OR Y > 2000 THEN PRINT
CHR$(7): GOTO 5060
5100 IF Y < > YE THEN GOSUB 7000:YE = Y: REM LOAD
5110 VTAB 10: HTAB 19: CALL - 868: INPUT "":A$
5120 IF A$ = "" THEN 500
5130 IF A$ = "" THEN A$ = "JAN": VTAB 10: HTAB 19: PRINT
A$
5140 GOSUB 1200: IF ERR THEN 5110
5150 F = VAL (A$)
5160 VTAB 12: HTAB 17: CALL - 868: INPUT "":A$
5170 IF A$ = "" THEN 500
5180 IF A$ = "" THEN A$ = "DEC": VTAB 12: HTAB 17: PRINT
A$
5190 GOSUB 1200: IF ERR THEN 5160
5200 T = VAL (A$)
5210 VTAB 14: HTAB 20: CALL - 868: INPUT "":A$
5220 IF A$ = "" THEN 500
5230 IF LEFT$(A$,1) = "S" THEN 5260
5240 IF LEFT$(A$,1) < > "P" THEN PRINT CHR$(7): GOTO
5210
5250 PRINT D$:"PR#":SLOT
5255 PRINT : PRINT : PRINT
5260 HOME
5270 A$ = "DATES LIST: " + STR$(Y)
5280 PRINT TAB(12):A$
5290 A$ = MO$(F) + " THRU " + MO$(T)
5300 PRINT TAB(40 - LEN(A$)) / 2):A$
5310 PRINT : PRINT
5320 FOR M = F TO T
5330 PRINT : PRINT MO$(M)
5340 FOR D = 1 TO 31
5350 IF N$(M,D) = "" THEN 5370
5360 PRINT TAB(3):D: TAB(7):N$(M,D)
5370 NEXT D
5380 NEXT M
5385 PRINT CHR$(12)
5390 PRINT D$:"PR#0"
5400 PRINT : INPUT " (HIT 'RETURN' TO CONTINUE...":A$
5410 GOTO 500
6000 REM ** SAVE TO DISK **
6010 ONERR GOTO 11000
6015 SAV = 1
6020 PRINT " (*;YE;: INPUT *) ARE YOU SURE? ":B$
6030 IF LEFT$(B$,1) = "N" THEN 6160
6040 IF LEFT$(B$,1) < > "Y" THEN PRINT CHR$(7): GOTO
6020
6050 B$ = "CAL.FILE-" + STR$(YE)
6060 PRINT D$:"UNLOCK":B$
6070 PRINT D$:"OPEN":B$
6080 PRINT D$:"WRITE":B$
6090 FOR M = 1 TO 12
6100 FOR D = 1 TO 31
6110 PRINT N$(M,D)
6120 NEXT D

```

```

6130 NEXT M
6140 PRINT D$:"CLOSE":B$
6150 PRINT D$:"LOCK":B$
6160 POKE 216,0: REM OFF ONERR
6170 PRINT CHR$(7):
6180 RETURN
7000 REM **LOAD FROM DISK**
7010 SAV = 0
7020 VTAB 20: PRINT Y:" FILE NOT IN MEMORY."
7030 INPUT "LOAD FROM DISK? ":B$
7040 IF LEFT$(B$,1) = "N" THEN 7200
7050 IF LEFT$(B$,1) < > "Y" THEN PRINT CHR$(7): GOTO
7030
7060 ONERR GOTO 11000
7070 B$ = "CAL.FILE-" + STR$(Y)
7080 PRINT D$:"OPEN":B$
7090 PRINT D$:"READ":B$
7100 FOR M = 1 TO 12
7110 FOR D = 1 TO 31
7120 INPUT N$(M,D)
7130 NEXT D
7140 NEXT M
7150 PRINT D$:"CLOSE":B$
7160 PRINT CHR$(7): POKE 216,0: REM OFF ONERR
7170 RETURN
7200 PRINT " (CLEARING OLD DATE FILE)"
7210 FOR M = 1 TO 12
7220 FOR D = 1 TO 31
7230 N$(M,D) = ""
7240 NEXT D
7250 NEXT M
7260 PRINT CHR$(7):
7270 RETURN
8000 HOME : PRINT
8010 A$ = " " + A$ + " ": HTAB (40 - LEN(A$)) / 2
8020 INVERSE : PRINT A$: NORMAL
8030 PRINT
8040 RETURN
10000 REM ** QUIT **
10005 A$ = "QUIT": GOSUB 8000
10010 VTAB 5: CALL - 868: INPUT "ARE YOU SURE? ":A$
10020 IF LEFT$(A$,1) = "N" THEN 500
10030 IF LEFT$(A$,1) < > "Y" THEN PRINT CHR$(7):
GOTO 10010
10040 VTAB 10: PRINT "APPLE CAL": TAB(23):"BY GLENN TE
MAN"
10050 VTAB 20: HTAB 10: PRINT "<G O O D B Y E !>"
10060 PRINT : END
11000 REM **ONERR**
11010 ERR = PEEK (222)
11020 L = PEEK (218) + PEEK (219) * 256: CALL 768: REM
FIX STACK
11030 ON (ERR) GOTO 11040,11040,11040,11000,11130,11190
,11040,11220,11260
11040 REM -UNIDENT ERROR
11050 PRINT : PRINT "ERROR: ": CHR$(7):ERR
11060 PRINT " ON LINE ":L: CHR$(7)
11070 PRINT : END
11080 REM -WRT PROTECTED
11090 PRINT " (WRITE PROTECTED - INSERT NEW DISK..."
11100 INPUT " (AND HIT 'RETURN': ":A$
11120 GOSUB 6070: GOTO 500
11130 REM -END OF DATA
11135 REM ('LOAD FROM DISK' ERROR)
11140 IF SAV = 1 THEN 11040
11150 PRINT D$:"DELETE":B$
11160 VTAB 20: CALL - 868
11165 IF NOD THEN 160
11170 PRINT " (FILE NOT FOUND!)"
11180 POKE 216,0: GOSUB 7200: GOTO 500
11190 REM -FILE NOT FOUND
11195 REM ('SAVE TO DISK' ERROR)
11200 IF SAV = 0 THEN 11040
11210 GOSUB 6070: REM SKIP 'UNLOCK'
11215 GOTO 500
11220 REM -I/O ERROR
11230 PRINT " (I/O ERROR - INSERT NEW DISK..."
11240 INPUT " (AND HIT 'RETURN': ":A$
11245 IF SAV = 1 THEN GOSUB 6060: GOTO 500
11250 IF SAV = 0 THEN GOSUB 7000: GOTO 500
11260 REM -DISK FULL
11270 PRINT D$:"DELETE":B$
11280 PRINT " (DISK FULL - INSERT NEW DISK..."
11290 INPUT " (AND HIT 'RETURN': ":A$
11300 GOSUB 6070: GOTO 500
20000 REM ** TITLE PAGE **
20005 TEXT : HOME :L = 25
20007 PRINT CHR$(7):
20010 VTAB 8: HTAB 9: PRINT "*****"
20020 VTAB 12: HTAB 9: PRINT "*****"
20030 VTAB 9: FOR I = 1 TO 3
20040 HTAB 9: PRINT "": HTAB 29: PRINT ""
20050 NEXT I
20055 FLASH
20060 VTAB 10: HTAB 11: PRINT "A P P L E C A L"
20065 NORMAL
20070 VTAB 10: HTAB 23: PRINT "BY GLENN TEMAN"
20080 POKE - 16368,0
20090 VTAB 23: PRINT " (HIT ANY KEY TO CONTINUE...)

```

continued on page 188

Peaceful Coexistence

by Robert V. Anderson
10 Anse-a-Valleau
Ste-Dorothee, Laval
Quebec
Canada H7Y 1V5

DESCRIPTION

The program FIFTY50 creates a diskette on which it is possible to save both DOS 3.3 and Pascal files. When running Pascal it will think that it is a Pascal diskette and under DOS it will behave as a Basic diskette. The files created under each system will be invisible to the other and all normal disk operations are possible in either mode.

HOW TO USE THE PROGRAM

It is essential to follow all of the following steps:

1. Initialize a DOS 3.3 diskette with a short 'HELLO' program.
2. Format or Zero a Pascal diskette.
3. Change the name of the Pascal diskette to FIFTY.
4. From the FILER M)ake a 144 block file with the name BASIC.SPACE.
5. K)runch the diskette FIFTY starting at block 6.
6. Type in and save the program FIFTY50 on APPLE1.
7. Execute FIFTY50 and follow the instructions.

The program will prevent you from making any monumental errors (Such as destroying files on the wrong diskette!).

After execution the diskette may be used with either Pascal or DOS 3.3. There will be 130 blocks available for Pascal files and 136 for DOS files. **The tracks 0-3 have been reserved for Pascal; therefore, you should not use this diskette to boot DOS** (Boot from a master and then catalog this new diskette). After the creation process, the name of the diskette may be changed from Pascal and copies may be made from either DOS or Pascal.

HOW IT WORKS

First, the program copies all the directory information from the Basic diskette. Then, on the Pascal diskette, two things occur. The file BASIC.SPACE has its classification changed from **DATA** to **BAD**. This prevents the K)runch command from moving it so that the hidden DOS files will be protected from Pascal disk operations. Next the recorded DOS directory is modified so that DOS thinks tracks 0-16 are in use. Note that DOS can't change this state of affairs because space can only be freed up by the DELETE command but DOS sees nothing to delete in these sectors!

POSSIBLE IMPROVEMENTS

Instead of reserving a fixed number of blocks for each system, it should not be too difficult to specify which tracks are to be reserved for each. However, it is always necessary to reserve track 17 for DOS and blocks 0-5 for Pascal.

```
PROGRAM FIFTY50;
  (* This program allows BASIC and Pascal programs to reside on *)
  (* the same diskette in harmonious ignorance of each other. *)

TYPE BUFFER1=PACKED ARRAY[0..8191] OF CHAR;
   BUFFER2=PACKED ARRAY[0..2047] OF CHAR;

VAR  A      : BUFFER1;
     B      : BUFFER2;
     I      : INTEGER;
     CH     : CHAR;
     FLAG   : BOOLEAN;
     NAME   : STRING[5];

BEGIN
  REPEAT
    PAGE(OUTPUT); GOTOXY(0,5);
    WRITE('Place the Basic diskette in drive 2 <CR>'); READLN;
    (*I-*)
    UNITREAD(5,A,8192,136);
    (*I+*)
    IF (ORD(AC13) < 17) AND (ORD(AC23) < 15) THEN
      BEGIN
        GOTOXY(0,10);
        WRITELN('Basic diskette not found!');
        WRITE('Hit a key and try again. ');
        UNITCLEAR(1); READ(KEYBOARD,CH);
        FLAG:=FALSE
      END
    ELSE FLAG:=TRUE
  UNTIL FLAG;
  REPEAT
    GOTOXY(0,15); WRITE(CHR(11)); GOTOXY(0,15);
    WRITE('Place the Pascal diskette in drive 2 <CR>'); READLN;
    (*I-*)
    UNITREAD(5,B,2048,2);
    (*I+*)
    NAME:=''; (* 5 spaces *)
    (* Check the name of the diskette in drive 2 *)
    (* If not correct then repeat or exit. *)
    FOR I:=1 TO 5 DO NAME[I]:=BC[6+I];
    IF NAME<>'FIFTY' THEN
      BEGIN
        GOTOXY(0,20);
        WRITELN('I can't find the diskette 'FIFTY' !');
        WRITE('Hit space to repeat <esc> to quit ');
        UNITCLEAR(1); READ(KEYBOARD,CH);
        IF CH=CHR(27) THEN EXIT(PROGRAM);
        FLAG:=FALSE
      END
    ELSE FLAG:=TRUE
  UNTIL FLAG;
  (* Mark Basic space as bad. *)
  BC[30]:=CHR(1);
  UNITWRITE(5,B,2048,2);
  (* Fill up part of the track sector list so *)
  (* that DOS thinks that the tracks are used. *)
  FOR I:=56 TO 123 DO AC[I]:=CHR(0);
  UNITWRITE(5,A,8192,136)
END.
```



APPLE SLUGGER

Apple Slugger

by Robert R. Devine
P.O. Box 10
Adona, AR 72001

I've had my Apple for about 1 1/2 years and try to get in as much programming time as possible. My biggest problem, of course, is trying to get some keyboard time, having to compete not only with my wife, but with my 3 children as well.

My elder daughter, Sondra, loves sports but soon tired of the usual computer fare, because most programs aren't enough like the real thing. Last year I injured my back and had to spend some time in bed, giving me a great opportunity to create my own sports game. The result was Apple Slugger.

The Slugger baseball game makes use of Hi-resolution graphics, a shape table, the buttons and dials on both game paddles, and incorporates the Applesoft tone routine.

SYSTEM NEEDS

Slugger is written in Applesoft and requires a 48K system. Due to the program length it may not run unless your Applesoft is in ROM. The program uses up virtually all of the available space below HGR page 1, so there isn't much room for enhancement. This also made it necessary to remove many of the REMs to conserve space.

PAGE MOD

There is a way to gain space for enhancement for those who don't mind some tinkering with their Apple. Write to Apple and request a copy of the IC number C12 pin modification, dated May 15, 1979. The modification is simple. This modification allows you to select HGR page 2, and still have 4 lines of text at the bottom of the screen. Being able to write over memory areas used by page 1 should allow lots of space for modification. A fancy change might be to have the fielders run off the field, and the new team take the field after every 1/2 inning, or it might be nice to have the base-runners actually run the baselines. Base stealing, player names, and other refinements could also be added.

THE PROGRAM APPROACH

Multiple statements are used heavily; in some cases this is to conserve space, but in most cases it enhances program logic. Since Applesoft ignores any statements on a line after encountering an IF statement that is false, compound statement logic can be handy in eliminating unnecessary branching.

The accompanying shape table defines the players which will be drawn on the screen. Enter the shape table through the monitor, then place it on the same disk with Slugger using **BSAVE PLAYER,A\$9500,L\$61**.

PLAYING SLUGGER

To play Slugger, the players first select and enter the names of their teams. Each player uses a game paddle; the home team uses PDL(0) and the visiting team uses PDL(1).

The person whose team is in the field presses the button on their paddle, which pitches the ball. The computer will select one of four different speed pitches. The person at bat then presses the button on the paddle to try to hit the ball. If the batter thinks it is a bad pitch, he may choose not to swing the bat. The result may be a ball or a call strike. If the batter misses the ball, or swings more than once, the result will be a strike. Your Apple will describe the pitch.

When the batter hits a fair ball, control will go to the dial on the fielding team's game paddle. The ball will take one of 10 different paths, and the fielder must try to catch the ball by turning his paddle dial. Your Apple will decide which fielder will try for the catch, and that fielder will begin to move. One minor detail... the fielder **NEVER** stands still. He is always moving either right or left. The trick is to find that magic spot on your dial where a small turn will change the fielder's direction of movement so that he can intercept the ball. The level of difficulty in catching the ball can be changed in line 7430.

Slugger provides some interesting surprises. Hitting and catching the ball involve skill, however many other things may happen at the random choice of your Apple; home runs, triples, doubles, singles, walks, strikeouts, foul balls, double plays, and (if your fielder has butterfingers) errors!

Slugger makes extensive use of GOSUBS. The main program only runs from lines 720 through 1220, the rest being subroutines called by the main program or other subroutines. If you want to make modifications, simply add the subroutine anywhere that it will fit, and GOSUB to it from the appropriate spot in the program.

THE PROGRAM

Line 100 "seeds" the random table so that every game is different.

Line 550 loads the shape table "PLAYER".

Line 700 draws the playing field, initializes the Applesoft tone routine, and sets the start of game flag.

Lines 720 - 1220 comprise the main program which consists mostly of tests for GOSUBing the subroutines.

Lines 6500 and 6510 are the Applesoft tone routine setup.

Line 7000 selects the pitch speed.

Lines 7005 - 7135 pitch the ball, read the paddle button to see if the batter swung the bat, draw the bat, and test to see if the ball was hit. If the ball was hit, PR=99. If the ball was missed, PR=0. If the batter didn't swing, PR=50.

Lines 7310 and 7325 decide which fielder will try to catch the ball, and select the ball path. The program then steps through line 7435 to move the ball and the fielder, as well as to test for the ball being caught. If the ball is caught, BC=99.

Lines 7500 - 7560 read the appropriate paddle button to prepare to pitch the ball.

Lines 8000 - 8045 determine the random results if the ball is not caught and update the runs scored and status of each base.

Lines 8200 - 8252 determine the random results if the ball was caught and update the outs or errors.

Lines 8400 - 8405 test to see if the ball was swung at, hit, or missed.

Lines 8421 - 8445 are subroutines to describe the batting results, and update the balls and strikes.

Lines 8511 - 8545 specify the coordinates of the players on the field.

Lines 8605 - 8630 draw the playing field and the baselines.

Line 8700 and 8730 draw the shape table "PLAYER" on the field.

Lines 8815 - 8900 do the scoring, baserunner advances, and record keeping.

Lines 8950 - 9210 print the various scoreboards, and input the team names.

Line 9130 serves as a pause, allowing time to read messages on the screen.

Slugger does not check for a winner, or for end of game, so enter Control/C to end the game.

Have fun at the ball game, and don't forget the hot dogs and beer!!!



LIST

```
*****
* APPLE SLUGGER *
* BY ROBERT DEVINE *
* COPYRIGHT (C) 1982 *
* BY MICRO-SPARC INC *
* LINCOLN, MA. 01773 *
*****

100 X = RND ( - PEEK (78) - PEEK (79) * 256): REM
SEED RANDOM TABLE
500 TMAB = 1:X0 = 2:Y0 = 2: DIM B(3): HOME : VTAB 1
2: HTAB 10: PRINT "APPLE SLUGGER": PRINT : HTAB
10: PRINT "BY ROBERT DEVINE": PRINT : HTAB 10:
PRINT "COPYRIGHT (C) 1982": HTAB 10: PRINT "B
Y MICRO-SPARC INC"
520 GOSUB 9130: GOSUB 9130:D% = CHR$(4)
550 HIMEM: 38144: IF PEEK (38250) < 255 THEN POKE
38250,255: PRINT D%"LOAD PLAYER": POKE 232,0:
POKE 233,149
```

```
700 GOSUB 9000: GOSUB 8605: GOSUB 8620: GOSUB 6500
:Y = 99
720 FOR X = 1 TO 8: POKE 768,40: POKE 769,30: CALL
770: ON X GOSUB 8511,8512,8513,8514,8515,8516,
8517,8518: HCOLOR= 3: GOSUB 8700: NEXT X: GOSUB
8525: HCOLOR= 0: GOSUB 8700
1030 IF Y = 99 THEN GOSUB 8850
1040 GOSUB 9200: IF TMAB% = T1% THEN GOSUB 7500
1080 IF TMAB% = T2% THEN GOSUB 7550
1090 POKE 768,50: POKE 769,50: CALL 770: GOSUB 700
0: GOSUB 8400
1120 IF PR = 99 THEN GOSUB 7300: HCOLOR= 1: GOSUB
8700: FOR X = 1 TO 5: ON X GOSUB 8511,8512,851
3,8514,8515: HCOLOR= 3: GOSUB 8700: NEXT X
1140 GOSUB 8620: IF BC = 99 THEN GOSUB 8200
1160 IF BC = 50 THEN FOR X = 1 TO 50:Y = PEEK ( -
16336) - PEEK ( - 16336): NEXT : GOSUB 8000
1170 GOSUB 8815: FOR X = 1 TO 3: IF B(X) = 1 THEN
HCOLOR= 0
1200 IF B(X) = 0 THEN HCOLOR= 1
1220 ON X GOSUB 8535,8540,8545: GOSUB 8700: NEXT :
GOTO 720
```

continued on next page

Apple Slugger (Cont.)

```

6500 FOR X = 770 TO 790: READ Y: POKE X,Y: NEXT : RETURN
6510 DATA 173,48,192,136,208,5,206,1,3,240,9,202,
208,245,174,0,3,76,2,3,96
7000 Y = 0: PR = 50: Z = INT ( RND (1) * 7) - 1: IF
Z < 2 THEN 7000
7005 FOR X = 97 TO 142 STEP Z
7010 HCOLOR= 3: FOR Z = 0 TO 2: HPLLOT 139,X + Z TO
141,X + Z: NEXT Z: HCOLOR= 1: FOR Z = 0 TO 2: HPLLOT
139,X + Z TO 141,X + Z: NEXT Z
7015 GOSUB 7100: NEXT X: RETURN
7100 IF X < 106 THEN RETURN
7105 IF TMAB% = T1% AND PEEK ( - 16287) < 128 THEN
RETURN
7110 IF TMAB% = T2% AND PEEK ( - 16286) < 128 THEN
RETURN
7120 HCOLOR= 0: HPLLOT 135,138 TO 152,138: HPLLOT 13
5,139 TO 152,139: POKE 768,200: POKE 769,50: CALL
770: HCOLOR= 1: HPLLOT 135,138 TO 152,138: HPLLOT
135,139 TO 152,139
7125 Y = Y + 1: IF Y > 1 THEN RETURN
7130 IF X < 136 OR X > 137 THEN PR = 0: RETURN
7135 IF X = 136 OR X = 137 THEN PR = 99: RETURN
7300 FOR X = 1 TO 10: POKE 768,20: POKE 769,40: CALL
770: NEXT
7310 X = 140: Z = INT ( RND (1) * 10) + 1: ON Z GOSUB
8511,8511,8514,8514,8512,8512,8515,8515,8513,8
513
7325 FOR Y = 130 TO 4 STEP - 6: ON Z GOSUB 7341,7
342,7343,7344,7345,7346,7347,7348,7349,7350: GOSUB
7360: IF BC = 99 THEN Y = 0: GOSUB 7365: RETURN
7330 NEXT Y: GOSUB 7365: RETURN
7341 X = X - 5: RETURN
7342 X = X - 4: RETURN
7343 X = X - 3: RETURN
7344 X = X - 2: RETURN
7345 X = X - 1: RETURN
7346 X = X + 1: RETURN
7347 X = X + 2: RETURN
7348 X = X + 3: RETURN
7349 X = X + 4: RETURN
7350 X = X + 5: RETURN
7360 IF NOT XO THEN 7370
7365 HCOLOR= 1: HPLLOT X0 - 1,Y0 - 1 TO X0 + 1,Y0 -
1: HPLLOT X0 - 1,Y0 TO X0 + 1,Y0: HPLLOT X0 - 1,
Y0 + 1 TO X0 + 1,Y0 + 1: IF Y < 4 THEN XO = 0:
RETURN
7370 HCOLOR= 3: HPLLOT X - 1,Y - 1 TO X + 1,Y - 1: HPLLOT
X - 1,Y TO X + 1,Y: HPLLOT X - 1,Y + 1 TO X + 1
,Y + 1
7375 IF TMAB% = T1% THEN 7390
7380 IF PDL (0) > 127 THEN XA = 5
7385 IF PDL (0) < 127 THEN XA = - 5
7386 GOTO 7400
7390 IF PDL (1) > 127 THEN XA = 5
7395 IF PDL (1) < 127 THEN XA = - 5
7400 XO = XN: XN = XN + XA: YO = YN: IF XN < 5 THEN X
N = 5
7406 IF XN > 250 THEN XN = 250
7410 HCOLOR= 1: GOSUB 8730: HCOLOR= 3: GOSUB 8700:
X0 = X: Y0 = Y
7430 IF XN + 8 - X < 4 AND XN + 8 - X > - 4 AND Y
N + 10 - Y < 5 AND YN + 10 - Y > - 5 THEN BC =
99: RETURN
7435 BC = 50: RETURN
7500 Y = PEEK ( - 16286): IF Y > 127 THEN RETURN
7510 GOTO 7500
7550 Y = PEEK ( - 16287): IF Y > 127 THEN RETURN
7560 GOTO 7550
8000 BALL = 0: STRIKE = 0
8005 HOME : VTAB 23: X = INT ( RND (1) * 10) + 1: ON
X GOSUB 8030,8035,8035,8040,8040,8040,8045,804
5,8045,8045: GOTO 9130
8030 IF TMAB% = T1% THEN R1N = R1N + B(1) + B(2) +
B(3) + 1
8031 IF TMAB% = T2% THEN R2N = R2N + B(1) + B(2) +
B(3) + 1
8032 PRINT "THE ";TMAB%;" GET A ";B(1) + B(2) + B(
3) + 1;" RUN HOME RUN": B(1) = 0: B(2) = 0: B(3) =
0: RETURN
8035 IF TMAB% = T1% THEN R1N = R1N + B(1) + B(2) +
B(3)
8036 IF TMAB% = T2% THEN R2N = R2N + B(1) + B(2) +
B(3)
8037 PRINT "THE ";TMAB%;" GET A TRIPLE & SCORE ";B
(1) + B(2) + B(3);" RUNS": B(1) = 0: B(2) = 0: B(
3) = 1: RETURN
8040 HTAB 8: PRINT "THE ";TMAB%;" GET A DOUBLE": B(
2) = B(2) + 1: IF B(1) = 1 THEN B(3) = B(3) +
1
8043 B(1) = 0: RETURN
8045 B(1) = B(1) + 1: HTAB 6: PRINT "THAT HIT IS GO
OD FOR A SINGLE": RETURN
8200 FOR X = 1 TO 10: POKE 768,30: POKE 769,30: CALL
770: NEXT
8205 HOME : VTAB 23: X = INT ( RND (1) * 6) + 1
8210 ON X GOSUB 8225,8230,8235,8240,8245,8250: BALL
= 0: STRIKE = 0: HIT = HIT - 1: GOTO 9130

```

```

8225 OUT = OUT + 1: PRINT "GROUNDER - THE THROW TO
FIRST - OUT ";OUT: IF B(1) = 1 THEN X = INT (
RND (1) * 2) + 1: IF X = 1 AND OUT < 3 THEN 0
UT = OUT + 1: HTAB 9: PRINT "DOUBLE PLAY !!! 0
UT "OUT: B(1) = 0: GOSUB 9130: RETURN
8226 IF B(1) = 1 THEN B(2) = B(2) + 1
8227 B(1) = 0: RETURN
8230 OUTS = OUTS + 1: HTAB 5: PRINT "HIGH FLY BALL
- CAUGHT - OUT ";OUT: RETURN
8235 OUTS = OUTS + 1: PRINT "LINE DRIVE - FANTASTIC
CATCH - OUT ";OUT: RETURN
8240 OUTS = OUTS + 1: HTAB 5: PRINT "HIGH POP UP -
CAUGHT - OUT ";OUT: RETURN
8245 B(1) = B(1) + 1: PRINT "GROUNDER THROW TO FIRS
T - NOT IN TIME": HIT = HIT + 1: RETURN
8250 IF TMAB% = T1% THEN E2R = E2R + 1
8251 IF TMAB% = T2% THEN E1R = E1R + 1
8252 B(1) = B(1) + 1: FOR X = 1 TO 60: Y = PEEK ( -
16336) - PEEK ( - 16336): NEXT : PRINT "BALL
IS DROPPED - ERROR - SAFE ON FIRST": RETURN
8400 HOME : VTAB 23: BC = 0: IF PR = 0 THEN FOR X =
1 TO 45: Y = PEEK ( - 16336) - PEEK ( - 16336
): NEXT X: GOSUB 8426: RETURN
8401 IF PR = 50 THEN X = INT ( RND (1) * 8) + 1: IF
X < 5 THEN POKE 768,100: POKE 769,70: CALL 77
0: ON X GOSUB 8421,8422,8423,8425: RETURN
8402 IF PR = 50 THEN FOR Y = 1 TO 45: Z = PEEK ( -
16336) - PEEK ( - 16336): NEXT Y: GOSUB 8424:
RETURN
8405 IF PR = 99 THEN X = INT ( RND (1) * 4) + 1: ON
X GOSUB 8427,8428,8427,8427: RETURN
8421 BALL = BALL + 1: HTAB 7: PRINT "HIGH AND OUTSI
DE !! - BALL ";BALL: RETURN
8422 BALL = BALL + 1: HTAB 8: PRINT "LOW AND INSIDE
!! - BALL ";BALL: RETURN
8423 BALL = BALL + 1: HTAB 5: PRINT "PITCH IS IN TH
E DIRT ! - BALL ";BALL: RETURN
8424 STRIKE = STRIKE + 1: HTAB 12: PRINT "CALL STRI
KE ! - ";STRIKE: RETURN
8425 BALL = BALL + 1: HTAB 9: PRINT "WILD PITCH !!
- BALL ";BALL: RETURN
8426 STRIKE = STRIKE + 1: X = INT ( RND (1) * 5) +
1: ON X GOSUB 8441,8442,8443,8444,8445: HTAB 1
0: PRINT STRIKE%;" STRIKE - ";STRIKE: RETURN
8427 HTAB 8: PRINT "THAT'S A GOOD SOLID HIT !": HIT
= HIT + 1: RETURN
8428 PR = 0: IF STRIKE < 2 THEN STRIKE = STRIKE + 1
8429 HTAB 3: PRINT "FOUL BALL - OUT OF PLAY - STRI
KE ";STRIKE: RETURN
8441 STRIKE% = "HARD SLIDER": RETURN
8442 STRIKE% = "FAST BALL": RETURN
8443 STRIKE% = "KNUCKLE BALL": RETURN
8444 STRIKE% = "CURVE BALL": RETURN
8445 STRIKE% = "CHANGE-UP": RETURN
8511 XN = 30: YN = 5: RETURN
8512 XN = 125: YN = 5: RETURN
8513 XN = 230: YN = 5: RETURN
8514 XN = 85: YN = 35: RETURN
8515 XN = 150: YN = 30: RETURN
8516 XN = 208: YN = 55: RETURN
8517 XN = 47: YN = 65: RETURN
8518 XN = 133: YN = 75: RETURN
8520 REM PLOT BATTERS XN,YN
8525 XN = 152: YN = 130: RETURN
8530 REM PLOT BASEMENS XN,YN
8535 XN = 219: YN = 70: RETURN
8540 XN = 130: YN = 34: RETURN
8545 XN = 65: YN = 67: RETURN
8605 HGR : HCOLOR= 1: FOR X = 0 TO 159: HPLLOT 0,X TO
279,X: NEXT : RETURN
8620 HCOLOR= 3
8625 HPLLOT 0,67 TO 140,137 TO 279,67: HPLLOT 140,13
6 TO 279,66: HPLLOT 60,95 TO 140,55 TO 220,95: HPLLOT
130,135 TO 130,150 TO 150,150 TO 150,135
8630 HPLLOT 131,135 TO 131,149 TO 149,149 TO 149,13
5: HPLLOT 140,141: HPLLOT 130,142 TO 142,142: HPLLOT
137,143 TO 143,143: HPLLOT 137,144 TO 143,144: HPLLOT
137,145 TO 143,145: RETURN
8700 SCALE= 1: ROT= 0: DRAW 1 AT XN,YN: RETURN
8730 DRAW 1 AT X0,Y0: RETURN
8815 IF STRIKES = 3 THEN OUTS = OUTS + 1: FOR X =
1 TO 50: Y = PEEK ( - 16336) - PEEK ( - 16336
): NEXT
8816 IF STRIKE = 3 THEN STRIKE = 0: BALL = 0: HOME
: VTAB 23: HTAB 6: PRINT "STRIKE OUT !!! NEXT
BATTER": GOSUB 9130
8820 IF BALL = 4 THEN B(1) = B(1) + 1: GOSUB 8535:
HCOLOR= 0: GOSUB 8700
8825 IF BALL = 4 THEN BALL = 0: STRIKE = 0: FOR X =
1 TO 9: POKE 768,75: POKE 769,30: CALL 770: NEXT
8830 IF OUT = 3 THEN TMAB = TMAB + 1: IF TMAB% = T
1% THEN HIT = HIT + 1
8840 IF OUT = 3 AND TMAB% = T2% THEN H2T = H2T + H
IT
8845 IF OUT = 3 THEN INNING = INNING + .5: GOSUB 9
100: FOR X = 1 TO 3: B(X) = 0: ON X GOSUB 8535,
8540,8545: HCOLOR= 1: GOSUB 8700: NEXT X: OUT =
0: HIT = 0
8850 TMAB% = T2%: IF TMAB / 2 = INT (TMAB / 2) THEN
TMAB% = T1%
8851 IF Y = 99 THEN Y = 0: RETURN
8855 IF B(1) = 2 THEN B(2) = B(2) + 1: B(1) = 1
8860 IF B(2) = 2 THEN B(3) = B(3) + 1: B(2) = 1
8865 IF B(3) > 2 AND TMAB% = T1% THEN R1N = R1N
+ B(3) - 1: GOSUB 8950: B(3) = 1
8870 IF B(3) > 2 AND TMAB% = T2% THEN R2N = R2N
+ B(3) - 1: GOSUB 8950: B(3) = 1

```



```

8900 RETURN
8950 HOME : VTAB 23: HTAB 8: PRINT "THE ";TMAB$;"
      SCORE ";B(3) - 1;" RUNS": GOTO 9130
9000 HOME : VTAB 10
9010 INPUT "WHAT IS THE HOME TEAM'S NAME ? ";T1$: PRINT
      : PRINT
9040 INPUT "WHAT IS VISITING TEAM NAME ? ";T2$: RETURN

9100 HOME : VTAB 22: HTAB 6: PRINT "AFTER ";INNING
      : " INNINGS THE SCORE IS...": GOSUB 9130
9110 HOME : VTAB 21: HTAB 20: PRINT "RUNS HITS
      ERRORS": PRINT T1$: PRINT T2$
9120 VTAB 22: HTAB 21: PRINT R1N$;"      ";H1T$;"
      ";E1R: HTAB 21: PRINT R2N$;"      ";H2T$;"
      ";E2R
9130 FOR X = 1 TO 30: FOR Y = 1 TO 30: NEXT Y,X: RETURN

9200 HOME : VTAB 21
9210 HTAB 8: PRINT "THE ";TMAB$;" ARE AT BAT": PRINT
      : HTAB 4: PRINT "BALLS-";BALL$;" STRIKES-"
      : STRIKE$;" OUTS-";OUT$; RETURN

```

REVERSE KEY — A Small Utility for the Apple Computer

by Phillip R. Rowley
2 Woodside Crescent-Clayton
Newcastle-under-Lyme
Staffs ST5 4BW
England

INTRODUCTION

While Apple's INVERSE, FLASH and NORMAL commands are very convenient if a whole printed message is to be in a particular format; they are irritating to use if only a single character is to be printed in FLASH or INVERSE mode.

I like to use this type of display for menus, where the key-letters for the options available are in INVERSE mode. To do this using the conventional method requires something of the form:

```
INVERSE : PRINT "A";NORMAL:PRINT
TAB(10)"ADD DATA"
```

If multiple options are to be given on a single line then the complexity builds up to such an extent that errors become very likely and even unavoidable.

The small machine-code program given below enables NORMAL, INVERSE and FLASH modes to be switched on and off by putting control characters into the PRINT command.

OPERATION

When BRUN from disk, the program modifies the character output switch to point to itself. From this point on, all output characters are tested for special 'format' characters.

If a control-N, control-F or control-I is sent to be printed then it is detected and the output format mask (\$32) is set to \$FF, \$3F or \$7F as appropriate. Characters printed after one of these format characters are then displayed in the appropriate style.

APPLE SLUGGER PLAYER SHAPE

#9500.9560

```

9500- 01 00 04 00 49 49 2D 35
9508- 3F 37 2D 35 3F 37 2D 35
9510- 3B 9F 2D 2D AD 3F 3F 3F
9518- 3F 17 2D 2D 2D 2D 2D 15
9520- 3F 3F 3F 3F 1F 3F 2E 4D
9528- 2D 2D 2D 2D 15 3F DF 3B
9530- 3F DF 77 09 2D 2D 6D 09
9538- F5 DB 3F 3F 3F 17 2D 6D
9540- 29 35 3F DF 3B BF 2D 4D
9548- 29 F5 3F DF 3B BF 2D 4D
9550- 29 F5 FF DB 3F 2E 4D 49
9558- F5 DB DB 3F 00 0B 00 00
9560- 00
*
```

HOW TO USE IT

REVERSE-KEY is activated by either a BRUN from disk or a BLOAD followed by a CALL 768. Once activated, Flash mode is activated by printing a control-F, Inverse by a control-I and Normal by a control-N.

It should be noted that NORMAL mode is not automatically reset following the end of a PRINT command. A control-N may be needed at the end of the PRINT string.

As control characters do not appear when listed, lines containing embedded control characters cannot be edited. If required, the CHR\$ equivalents of the control characters may be used (see BASIC demo program).

Embedded control characters in listings may be revealed, if required, by first running the program on page 151 of the DOS 3.3 manual. Reverse-key may be deactivated by a PR#0 command, and reactivated by a CALL 768.

REVERSE KEY MACHINE LANGUAGE
BY PHILLIP ROWLEY
COPYRIGHT (C) 1982
BY MICROSPARC, INC
LINCOLN, MA. 01773

MEM	ASSY L	MACH L	COMMENT
300:	LDA #0C	A9 0C	RESET OUTPUT POINTER
302:	STA #36	85 36	
304:	LDA #03	A9 03	
306:	STA #37	85 37	
308:	JSR \$3EA	20 EA 03	PASS OUTPUT LOCATION TO DOS
308:	RTS	60	RETURN FOR NEXT CHARACTER
30C:	CMP #86	C9 86	IS CHAR 'CTRL F'?
30E:	BEQ #31B	F0 0B	YES, THEN GOTO #31B
310:	CMP #8E	C9 8E	IS CHAR 'CTRL N'?
312:	BEQ #320	F0 0C	YES, THEN GOTO #320
314:	CMP #89	C9 89	IS CHAR 'CTRL I'?
316:	BEQ #325	F0 0D	YES, THEN GOTO #325
318:	JMP #FDF0	4C F0 FD	PRINT CHAR IN ACCUMULATOR
318:	LDA #7F	A9 7F	FOLLOWING CHARS ARE FLASH
31D:	STA #32	85 32	SO STORE #7F IN #32
31F:	RTS	60	RETURN FOR NEXT CHAR
320:	LDA #FF	A9 FF	FOLLOWING CHARS ARE NORMAL
322:	JMP #31D	4C 1D 03	SO STORE #FF IN #32
325:	LDA #3F	A9 3F	FOLLOWING CHARS ARE INVERSE
327:	JMP #31D	4C 1D 03	SO STORE #3F IN #32

NOTE: FOR NON-DISK SYSTEMS THE JSR \$3EA AT #308 IS NOT NEEDED AND THE THREE EXTRA BYTES #308-30A CAN BE REPLACED BY NOP NOP NOP: I.E. EA EA EA

LIST

```

1 REM *****
2 REM * REVERSE KEY DEMO *
3 REM * BY PHILLIP ROWLEY *
4 REM * COPYRIGHT (C) 1982 *
5 REM * BY MICRO-SPARC INC *
6 REM * LINCOLN, MA. 01773 *
7 REM *****
8 REM <<< NOTE: IN LINES 45 AND 110: 'F'=CTRL F; 'I'=CTR
      L I; 'N'=CTRL N
10 PRINT CHR$ (4)"BRUN REVERSE KEY"
15 HOME : HTAB 10: PRINT "REVERSE KEY DEMO"
20 HTAB 10: PRINT "=====": PRINT
25 I$ = CHR$ (9);N$ = CHR$ (14);F$ = CHR$ (6)
30 LIST 25: LIST 35: PRINT
35 PRINT "AAAA" F$ "AAAA" N$ "AAAA" I$ "AAAA" N$ "AAAA"
40 PRINT : PRINT "OR USING EMBEDDED CONTROL-CHARACTERS": PRINT
      : LIST 45
42 FOR T = 0 TO 9999: NEXT
45 PRINT : PRINT "AAAAFAAANAANAIAAANAANA"
50 PRINT : PRINT "LISTING OF LINES WHEN REVERSE KEY IS ON
      SHOWS HOW LINE WILL PRINT"
55 PRINT : PRINT "LISTING WHEN 'REVERSE KEY' IS OFF SHOWS
      NOTHING UNUSUAL"
60 PRINT : PRINT "PR#0"
65 PR# 0
70 LIST 45
75 FOR T = 0 TO 9999: NEXT
80 PRINT : PRINT "RESTART 'REVERSE KEY' - CALL 768"
85 CALL 768
90 PRINT : LIST 45
100 PRINT : PRINT "REVERSE KEY IS NOW ON FOR YOU TO TRY"
110 PRINT : PRINT "LINES CONTAINING EMBEDDED CONTROL CH
      ARACTERS CANNOT BE EDITED"
120 PRINT : PRINT "LINES USING STRING VARS CAN BE EDITED
      E.G.": PRINT : LIST 35

```


Amper-Find

by Joe Brooks
521 Moody
Athens, TX 75751

Amper-Find is a fast, compact machine language program that will locate variables, strings, commands, and even control characters within an APPLESOFT program, and then print out the corresponding line number(s) using a space efficient format. It resides in page 3 of memory, and can be loaded at any time without altering your current program. Uses include locating GOTO and GOSUB commands to aid in the deletion of REM statements, searching for a variable to determine whether or not it has been used or if it is free for assignment, or perhaps to find a misspelling that you noticed while you were running a program. Also, a feature that I have found quite useful is the ability to locate control characters.

ENTERING AMPER-FIND

To enter Amper-Find into memory, first type CALL -151 to enter the Monitor, then type in the memory listing as shown (except that you should substitute ":" in place of "-"). After the program is entered, type **BSAVE AMPER FIND,A\$300,L\$C2** to save to disk, or type **300.3C1W** to save to tape.

HOW TO USE IT

To use the program from tape, type CALL -151, then 300.3C1R and turn on the recorder. After it has loaded, type 300G, then CTRL-C to return to APPLESOFT. From disk, simply type **BRUN AMPER FIND**. You have just added a new "FIND" command to your Apple!!

To use the new **FIND** command, type **&** followed by the item you wish to locate. This may be a string, variable, command, or control character, individually or in any combination. After pressing the RETURN key, the appropriate line numbers will be printed almost instantly! The **FIND** command can also distinguish between strings and variables by typing **S** or **V** in place of the colon. For instance, typing **&SB** will locate all strings containing the character "B". **&VB** will locate all variables containing "B". If the **&** is not followed by a ":", "S", or "V", then the message "SYNTAX ERROR" will be printed.

NOTE: If the program does not appear to be working correctly, you probably have an APPLESOFT command hidden somewhere. This is because APPLESOFT stores its commands as "tokens", using only one byte of memory (a token table can be found on page 121 of the APPLESOFT manual). For example, **&:LETTER** will actually search for the command "LET" followed by the string or variable "TER". **&SAVENUE** will result with "SYNTAX ERROR" because the token for "SAVE" is stored, rather than "S". Also, spaces will be ignored, unless preceded with a DATA or REM statement, or a quotation mark.

HOW IT WORKS

Here is a detailed explanation of how the program works.

SET &-JUMP VECTOR

This portion of the program is used only for initialization. When you **BRUN AMPER FIND** (300G from tape), it places a jump to the start

of the program (START) at \$3F5 (VECTOR). VECTOR is the location APPLESOFT jumps to when it encounters the "&".

SET POINTERS

Zero page pointers and flags are used to minimize program size (this also increases program execution speed). The HTAB pointer is used in the PRINT ROUTINE. FLGSU is the string-variable flag and is set according to the command: ":"=0, "S"=1, "V"=2. If a command is illegal, the APPLESOFT ROM is used (\$DEC9) to print "SYNTAX ERROR" and return to APPLESOFT. NXTLN begins a loop that when branched to sets the LSTART, LINENO, and FNEXT pointers. To help understand how they are used, I will explain the structure of an APPLESOFT program stored in memory. Reference should be made to the FIND TEST DEMO.

Beginning at \$801, the first two bytes contain an address that I call a "line pointer". The following two bytes contain the line number, stored in hexadecimal. The bytes following this contain the line itself, stored in ASCII code (an ASCII code table can be found on page 138 of the APPLESOFT manual), except for the commands, which are stored as tokens (also hexadecimal). The last byte of the line contains "00". This sequence is then repeated, beginning once again with a line pointer. The line pointers always contains the address of the next line pointer (low byte first, high byte second).

Back to my program, **LINENO** is set to the line number, and **LSTART** is set to the starting address of the line. **FNEXT** is set to the next line pointer. If it is equal to zero, then there are no more lines to search, and a branch is made to the basic warmstart routine in APPLESOFT ROM at \$D43C (BASIC), returning the program to APPLESOFT.

MAIN SEARCH ROUTINE

FLGSTR is used to distinguish between variables and strings. It is toggled between 0 and 1 (0=variable, 1=string) if a quotation mark, DATA token, or REM token is encountered.

It should be noted here that when APPLESOFT converts its commands to tokens, they are stored as such in the keyboard buffer as well as in program memory. Also, the first two bytes of the buffer are not used here, for they contain the "&" and the ":", "S", or "V".

The keyboard buffer is compared, one byte at a time, to the line pointed to by LSTART. If there is no match, then a branch is made to NXTLN, where pointers are updated for the next line. If there is a match, then the string-variable decision is made by comparing FLGSV and FLGSTR. If all is in order, a branch is made to the print routine (PRINT). Otherwise, the search is continued where it left off.

PRINT ROUTINE

The print routine makes use of two APPLESOFT ROM routines: **\$ED24 (LINPRT)** prints the line number, and **\$DAFB (CRDO)** prints a carriage return. At each pass the line number pointed to by LINENO is printed. Then the horizontal position counter (HTAB) is incremented by 6, and checked. If it reaches 36 (\$24), then a carriage return is printed, and HTAB is reset to zero. The horizontal position of the cursor (HPOS) is set equal to HTAB, and a jump is made to NXTLN.

Well, that about wraps it up. I hope that you find Amper Find to be as useful an addition to your Apple utility library as I have!!

Amper-Find
Hex Memory Listing
Copyright (c) 1982
by Micro-SPARC Inc.

JCALL-151

*300.3C1

```
0300- A9 4C 8D F5 03 A9 10 8D
0308- F6 03 A9 03 8D F7 03 8D
0310- A0 00 84 03 84 06 C8 84
0318- 1C A9 08 85 1D AD 01 02
0320- C9 3A F0 10 84 06 C9 53
0328- F0 0A C8 84 06 C9 56 F0
0330- 03 4C C9 DE A5 1C 18 69
0338- 02 85 1A A5 1D 69 00 85
0340- 1B A5 1A 18 69 02 85 18
0348- A5 1B 69 00 85 19 A0 01
0350- B1 1C F0 6B AA 88 B1 1C
0358- 85 1C 86 1D 84 07 88 A2
0360- FF C8 E8 D0 02 84 08 8D
0368- 02 02 C9 00 D0 0F A5 06
0370- F0 2D 38 E5 07 F0 28 C9
0378- 02 F0 24 D0 1D B1 18 F0
0380- B3 C9 22 F0 08 C9 83 F0
0388- 04 C9 82 D0 06 A5 07 49
0390- 01 85 07 B1 18 D0 02 02
0398- F0 C7 A4 08 4C 5F 03 A0
03A0- 00 B1 1A AA C8 B1 1A 20
03A8- 24 ED A5 09 18 69 06 C9
03B0- 24 D0 05 20 FB DA A9 00
03B8- 85 09 85 24 4C 34 03 4C
03C0- 3C D4
```

Amper-Find Test Demo

ILIST

```
10 HOME
20 PRINT "THIS IS A TEST"
30 END
```

0300- 00	07	08	0A	00	HOME	37	00	LINE
\$800	\$801	\$802	\$803	\$804	\$805	\$806	\$807	10
POINTER	LINE	NUMBER	PRINT	"	T	H	I	
0308- 08	14	00	BA	22	54	48	49	
\$808	\$809	\$80A	\$80B	\$80C	\$80D	\$80E	\$80F	
S	I	S	A	T				
0810- 53	20	49	53	20	41	20	54	
\$810	\$811	\$812	\$813	\$814	\$815	\$816	\$817	
E	S	T	"	LINE	POINTER	LINE		
0818- 45	53	54	22	00	23	08	1E	
\$818	\$819	\$81A	\$81B	\$81C	\$81D	\$81E	\$81F	
NUMBER	END							
0820- 00	00	00	00	00	00	00	00	
\$820	\$821	\$822	\$823	\$824	\$825	\$826	\$827	


```

1000 *****
1010 *
1020 *
1030 *
1040 *   Amper-Find
1050 *   by Joe Brooks
1060 *   Copyright (c) 1982
1070 *   by Micro-SPARC, Inc
1080 *
1090 *
1100 *****
1110 .OR      $300
1120 FLGSU   .EQ      $06
1130 FLGSTR  .EQ      $07
1140 YSTO    .EQ      $08
1150 HTAB     .EQ      $09
1160 LSTART   .EQ      $10
1170 LINENO   .EQ      $1A
1180 FNEXT     .EQ      $1C
1190 HPOS      .EQ      $24
1200 KEYBRD   .EQ      $200
1210 VECTOR   .EQ      $3F5
1220 BASIC     .EQ      $043C
1230 CRDO      .EQ      $DAFB
1240 SNTXER    .EQ      $DEC9
1250 LINPR1    .EQ      $ED24
1260 *
1270 * SET %-JUMP VECTOR
1280 *
0300- A9 4C      1290      LDA #4C
0302- 8D F5 03   1300      STA VECTOR
0305- A9 10      1310      LDA #START
0307- 8D F6 03   1320      STA VECTOR+1
030A- A9 03      1330      LDA /START
030C- 8D F7 03   1340      STA VECTOR+2
030F- 60         1350      RTS
1360 *
1370 * SET POINTERS
1380 *
0310- A0 00      1390      LDY #00      ZERO
0312- 84 09      1400      STY HTAB     HTAB POINTER AND
0314- 84 06      1410      STY FLGSU    STR-VAR FLAG
1420 *
0316- C8         1430      INY          SET FNEXT
0317- 84 1C      1440      STY FNEXT   TO STARTING
0319- A9 08      1450      LDA #08     ADDRESS OF
031B- 85 10      1460      STA FNEXT+1 PROGRAM
1470 *
031D- A0 01 02   1480      LDA KEYBRD+1 COMMAND?
0320- C9 3A      1490      CMP #3A     ASCII ':'?
0322- F0 10      1500      BEQ NXTLN   YES!
0324- 84 06      1510      STY FLGSU    SET STR-VAR FLAG=1
0326- C9 53      1520      CMP #53     ASCII 'S'?
0328- F0 0A      1530      BEQ NXTLN   YES!
032A- C8         1540      INY
032B- 84 06      1550      STY FLGSU    SET STR-VAR FLAG=2
032D- C9 56      1560      CMP #56     ASCII 'U'?
032F- F0 03      1570      BEQ NXTLN   YES!
0331- 4C C9 0E   1580      JMP SNTXER  SYNTAX ERROR
1590 *
0334- A5 1C      1600      LDA FNEXT   SET LINENO
0336- 18         1610      CLC          TO ADDRESS
0337- 69 02      1620      ADC #02     OF CURRENT
0339- 85 1A      1630      STA LINENO  LINE NUMBER
033B- A5 10      1640      LDA FNEXT+1
033D- 69 00      1650      ADC #00
033F- 85 18      1660      STA LINENO+1
1670 *
0341- A5 1A      1680      LDA LINENO  SET LSTART
0343- 18         1690      CLC          TO ADDRESS
0344- 69 02      1700      ADC #02     OF CURRENT
0346- 85 18      1710      STA LSTART  LINE START
0348- A5 18      1720      LDA LINENO+1
034A- 69 00      1730      ADC #00
034C- 85 19      1740      STA LSTART+1
1750 *
034E- A0 01      1760      LDY #01     SET FNEXT
0350- B1 1C      1770      LDA (FNEXT),Y TO NEXT
0352- F0 68      1780      BEQ END    LINE POINTER
0354- AA         1790      TAX
0355- 88         1800      DEY
0356- B1 1C      1810      LDA (FNEXT),Y
0358- 85 1C      1820      STA FNEXT
035A- 86 10      1830      STX FNEXT+1
1840 *
1850 * MAIN SEARCH ROUTINE
1860 *
035C- 84 07      1870      STY FLGSTR
035E- 88         1880      DEY
035F- A2 FF      1890      LDX #FF
0361- C8         1900      STEP4     INY
0362- E8         1910      INX
0363- D0 02      1920      BNE STEP5
0365- 84 08      1930      STY YSTO
0367- D0 02 02   1940      STEP5     LDA KEYBRD+2,X
036A- C9 00      1950      CMP #00
036C- D0 0F      1960      BNE STEP6
036E- A5 06      1970      LDA FLGSU
0370- F0 2D      1980      BEQ FPRINT  PRINT ALL?
0372- 38         1990      SEC          YES!
0373- E5 07      2000      SBC FLGSTR  PRINT STRING?

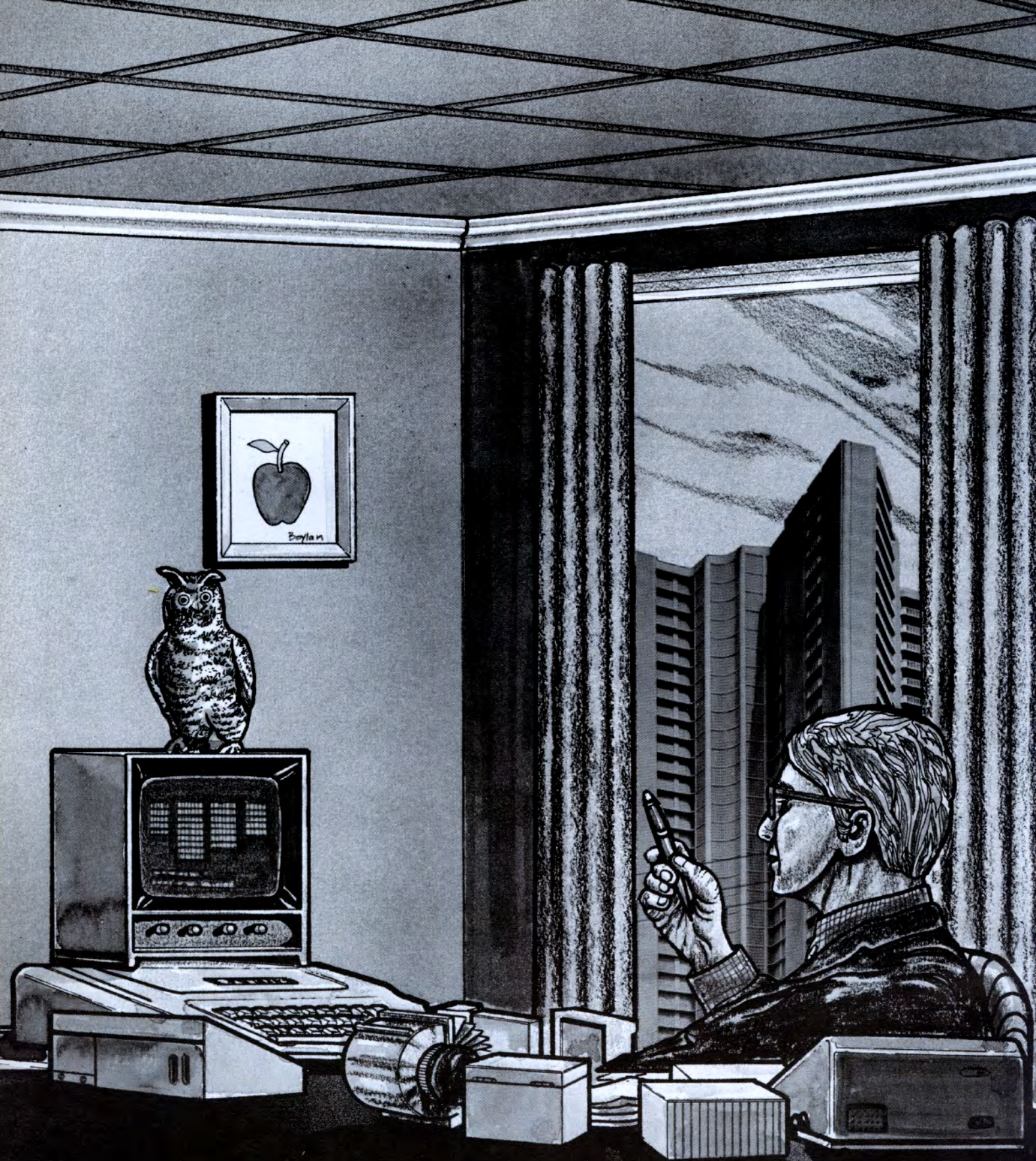
```

```

0375- F0 28      2010      BEQ FPRINT  YES!
0377- C9 02      2020      CMP #02     PRINT VARIABLE?
0379- F0 24      2030      BEQ FPRINT  YES!
037B- D0 1D      2040      BNE STEP8
037D- B1 18      2050      STEP6     LDA (LSTART),Y
037F- F0 B3      2060      BEQ NXTLN
2070      CMP #22
2080      BEQ SHFLG
2090      CMP #83
2100      BEQ SHFLG
2110      CMP #B2
2120      BNE STEP7
2130      SHFLG   LDA FLGSTR
2140      EOR #01
2150      STA FLGSTR
2160      STEP7     LDA (LSTART),Y
2170      CMP KEYBRD+2,X
2180      BEQ STEP4
2190      STEP8     LDY YSTO
2200      JMP STEP3
2210 *
2220 * PRINT ROUTINE
2230 *
033F- A0 00      2240      FPRINT   LDY #00     PRINT LINE
03A1- B1 1A      2250      LDA (LINENO),Y NUMBER
03A3- AA         2260      TAX
03A4- C8         2270      INY
03A5- B1 1A      2280      LDA (LINENO),Y
03A7- 20 24 ED   2290      JSR LINPR1
03AA- A5 09      2300      LDA HTAB     HTAB TO
2310      CLC          NEXT FIELD
03AD- 69 06      2320      ADC #06
03AF- C9 24      2330      CMP #24     TIME FOR CR?
03B1- D0 05      2340      BNE STEP9
03B3- 20 FB DA   2350      JSR CRDO    YES!
03B6- A9 00      2360      LDA #00
03B8- 85 09      2370      STEP9     STA HTAB
03BA- 85 24      2380      STA HPOS
03BC- 4C 34 03   2390      JMP NXTLN
2400 *
2410 * RETURN TO APPLESOF
2420 *
03BF- 4C 3C D4   2430      END      JMP BASIC

```





BOND MANAGER

APPLE BOND MANAGER

Apple Bond Manager

by Doug Sprinkle
4269 Bristol
Troy, MI 48098

You just inherited a modest nest egg from your spinster aunt and have decided to invest this money for a rainy day. The stock market is too risky, and you instead decide to invest in secure corporate bonds. You visit your stockbroker to determine which bonds will provide the maximum return on your money. After asking a number of financial questions, your broker suggests **two different bonds**, one of which has a face interest rate of **eleven percent** and the other a face interest rate of **two percent**. Before you conclude that your broker is a candidate for a padded cell, read on.

All bonds have a **Face Value** which represents the sum the issuer will pay to you at some future date, known as the **bond Due Date**. Likewise, all bonds have a **face interest rate** which represents the periodic interest on the bond that the issuer will pay to you up to the bond Due Date. Thus, a bond having a Face Value of \$2000.00 and an interest rate of 10% will pay \$200.00 per year as interest on the bond. Typically, this interest is paid semi-annually or \$100.00 every six months for this example.

Bonds, however, are rarely (if ever) sold for their Face Value. Thus, if you purchase the \$2000.00 bond for \$1000.00, the issuer will still pay 10% interest on the Face Value (\$2000.00) of the bond and your actual annual return jumps to 20%, i.e. \$200/year on your \$1000.00 investment. Your actual rate of return on your money, however, is still higher since the issuer will pay you \$2000.00 on the bond Due Date on your \$1000.00 investment. The precise increase will vary, of course, on the bond Due Date.

The current interest rate and the overall bond yield are respectively known as the **Current Yield** and **Yield to Maturity** and are readily available to investors in the Wall Street Journal and other financial publications. Unfortunately, there is one further complicating factor to consider when determining your rate of return from a bond. Specifically, the federal Internal Revenue Service treats all **interest income** from your bond as ordinary income and levies a tax based on your individual tax rate. Conversely, any increase between the purchase price of your bond and its Face Value is treated as a **capital gain** (if the bond is held longer than one year) and only one half of the gain is taxed. Consequently, the after tax rate of return is not only difficult to determine but also varies, depending on a number of different factors. The instant program, **Bond Manager**, simplifies this process by calculating not only the current yield for the bond but also the capital gain yield for the bond. In addition, Bond Manager uses these calculated values to determine both the total bond yield and the after tax yield depending, of course, on your individual tax rate. All of this is discussed in greater detail below.

SYSTEM REQUIREMENTS

Bond Manager requires Applesoft in ROM, one disk drive and at least 32K RAM. A printer is optional but, obviously, a requirement to run the printer routine. No special procedure is necessary to run Bond Manager — a simple "RUN BOND MANAGER" will suffice.

HOW THE PROGRAM WORKS

Bond Manager is driven from the main Menu which begins at line 11230 in the program. From the main Menu, eight options are available to the user. The main Menu requests that the user enter a selection number from 1 to 8 and all other entries are ignored (accompanied by a "beep" from the Apple to advise you of the error). Each of these eight options is discussed separately below.

OPTION 1 — DATA INPUT

Option 1 enables the user to enter the **bond data** from the keyboard and transfers control of the program to line 5060. First, the program prompts the user to enter the bond name. Any alphanumeric string can be entered, e.g. "AT&T" (without quotation marks), but **only the first five characters will be significant**.

Secondly, the program prompts the user to enter the **bond purchase price**, i.e., the price actually paid for the bond. Any numeric value greater than .50 will be accepted. Otherwise, the Apple bell will beep and the purchase price must be reentered.

Third, the program prompts the user to enter the **bond Face Value**, i.e., the amount the bond issuer will pay you at the due date of the bond. The bond Face Value is usually a round number, e.g. 25, 1000 or the like. As with the Purchase Price, only numeric values greater than .50 will be accepted.

Fourth, the user must enter the **bond interest rate (in percent)**, i.e. 5, 8.5 or whatever, and the program first displays the appropriate screen prompt. Only a numeric value greater than .50 is acceptable (ALL bonds pay more than 0.5%).

Fifth, the program prompts the user to enter the **month and day Due Date** in the form **MO.DA**. Thus, **February 7** is entered as **2.07** while **February 2** is entered as **2.02 and not 2.2** (the value **2.2 is equivalent to February 20**). Any month value greater than 12 or less than 1 is not accepted, and any day value greater than 31 is not accepted. Bond Manager does not, however, check for 28, 29 or 30 day months so that February 31 will be accepted by the program. As a practical matter, however, this makes very little difference in the bond calculations. As a default condition, if only the month is specified, Bond Manager will automatically supply the first day of the month for the due date.

Lastly, the user must enter the year **Due Date** for the bond, e.g. 1987. The **complete year date must be entered** and (you guessed it) abbreviated entries are rejected and accompanied by a beep from the Apple bell.

After entry of the bond data, four commands are available to the user and the prompts for these commands are displayed at the lower right hand corner of the screen. The letter "R" will return the user to the main Menu, the letter "D" will delete the record, and a carriage return, <CR>, will proceed to the next record for entry of the data for the

next bond. Whenever the "R" key or a carriage return is depressed, a GOSUB to the routine at line 3000 is executed where the calculated bond values (see Option 4 below) are generated. The final option, "E" for Edit, is described below.

Following entry of the data for the first bond and after either "R" or <CR> is depressed, the program prompts the user to **enter the date (self-explanatory) and the user's incremental tax rate** (not so self-explanatory). All of us have a federal incremental tax rate which is dependent on our TAXABLE income rather than GROSS income. In essence, the taxable income is equal to the gross income minus the deductions available to the individual. The Incremental Tax Rate can be determined from Tables X-Z in the instruction booklet for IRS form 1040.

EDITING

The Edit routine, which begins at line 10250, permits the user to edit the data after it has been entered. Once the "E" key is depressed, the **numbers 1 - 6** are displayed in a very narrow window on the left side of the screen so that one number is aligned with each line of data previously entered. **Each number is called A "field number"**.

The Edit routine then sets a very small window at the lower left hand corner of the screen (see program line 10380) and asks the user to specify which field he or she wishes to edit. The Edit routine then prompts the user to enter the new value for the data field and, after the carriage return is depressed, the new data is updated on the screen. As in the data entry procedure, the Edit routine checks the data during input to ensure that it is in proper form.

OPTION 2 — LOAD DATA FILES

The second option from the main Menu enables the user to load previously stored data from the disk drive. From this option, which begins at line 14000, the user can either append the data from the disk data file to the existing data in memory, or replace the existing data in memory, if any. Thereafter, three commands are open to the user.

After prompting from the program, the letter "M" will abort the command and return the program to the main Menu. The letter "C" will display a disk catalog; all data files will begin with "BONDDATA". Lastly, the **user may specify a single character "FILE IDENTIFIER" which is the character following the file name BONDDATA**. Any character other than "M" or "C" will do.

The File Identifier serves a two-fold purpose. **First**, it allows the user to store multiple data files since the "in memory" capacity of the Apple is limited. **Secondly**, the File Identifier enables the user to separate and save bond files with bonds having a different risk factor. Certain bonds have uncertain final values, and thus pay higher yields than the so-called safe or "bluechip" bonds. With high risk bonds, however, the bond issuer may not be able to pay you the bond Face Value on the bond Due Date.

OPTION 3 — SAVE DATA TO DISK

The third option from the main Menu per-

continued on next page

Apple Bond Manager (Cont.)

mits the user to save data in memory to a disk file. This routine, which begins at line 15000, prompts the user to **specify a single character File Identifier** which is appended to the filename "BONDDATA". All data is preserved in memory following this routine.

OPTION 4 — DISPLAY/EDIT DATA

This routine, which begins at line 6000, displays not only the bond data previously entered, but also the calculated bond yields (discussed below). The program prompts the user to enter the starting record number, or <RETURN> for all records.

Four calculated values are displayed during this routine, namely the **CURRENT YIELD**, the **CAPITAL GAIN YIELD**, the **GROSS YIELD** and, perhaps most importantly, the **AFTER TAX YIELD**. The **CURRENT YIELD** is simply the Bond Face Value divided by the Purchase Price and multiplied by the bond Face Interest Value. Thus, a \$1000 bond purchased for \$500 and having a face interest of 4% will have a Current Yield of $(\$1000/\$500 \times 4\%)$ or 8%.

The **CAPITAL GAIN YIELD** is equal to the appreciation of the bond prorated over the life of the bond, provided that the bond is held for over one year, the so-called capital gains holding period. Thus, assuming that the \$1000 bond in the previous example matured or was payable ten years from the purchase date, the **CAPITAL GAIN YIELD** is equal to $100\%/10$ Years or 10% per year.

The **GROSS YIELD** is straightforward and is simply equal to the **CAPITAL GAIN YIELD** plus the **CURRENT YIELD**. The **GROSS YIELD** is significant for tax exempt bonds, such as municipal bonds.

The final calculated value is the **AFTER TAX YIELD** which is the total yield on the bond after taking into account the incremental tax rate of the individual user. Since only one half of the capital gain is taxed as ordinary income, and thus subject to the full tax rate, high income users are financially better off purchasing bonds with high capital gains than high ordinary income. This will become apparent shortly.

After the data and calculated values for the first record are displayed on the screen, the same four commands as in the data entry routine are available, namely "R", "D", "E" and <CR>. These commands, which are displayed in the lower righthand corner, produce the same effect as in the data entry routine but do so in a slightly different fashion.

The **Delete** command immediately replaces the bond name with a string of three asterisks (AS\$), sets the delete flag (variable DL) to 1 and then displays the next record. The Return command first checks to see if the delete flag is set and, if it is, first executes a GOSUB 2000 at line 6600 which deletes all bond data records having the bond name "*****". The Return command then returns the program to the main Menu.

The **Edit** command, as before, permits the bond data to be edited and then immediately updates the screen. The Apple window is set to the right side and spaced upwardly from the bottom of the screen so that only the data is cleared and replaced on the screen. This procedure is much less distracting and somewhat faster than clearing and then rewriting the entire screen.

Lastly, a **carriage return** will display the data and calculated results for the next record. As in the Edit command, only the data and calculated results are cleared and then rewritten to the screen.

OPTION 5 — PRINT RESULTS

The fifth option from the main Menu begins at line 7000 and enables the user to obtain columnar printed output of the bond data as well as the calculated results. This routine, which prompts the user for the starting record as well as the ending record, **will print the bond data output and calculated results for seven bonds before starting a new line**. The printing can be interrupted at any time by pressing the letter "S". As written, the printer output will print the bond results for fourteen bonds on a sixty line page of paper. Customization instructions follow.

OPTION 6 — CHANGE TAX RATE

Option 6 enables the user to change the incremental tax rate. Do not set this tax rate upon entry to the program — when the program needs the incremental tax rate, it will ask for it. After entry of a different tax rate, the program will automatically calculate all of the values for all of the bonds.

The reason for this routine is two-fold. First, it makes the program adaptable to different users. Secondly, this option permits the user to easily recalculate the bond yield in the event that the tax rates change.

OPTION 7 — SORTING THE DATA

With Option 7 the user can sort the bond data by any data field as well as any calculated result. The Sort Menu begins at line 13000 and is self-explanatory. The sort routine itself begins at line 1000 and is placed near the start of the program in order to speed the execution of the sort routine. The Sort routine sorts numerical values in descending order and the bond names in ascending order, i.e. A-Z.

The Sort routine is simple in operation. It starts at Record 1 and simply searches through all of the bond data records for the bond data record having highest value of the specified sort field. The sort then "swaps" the entire first record with the bond record having the highest value for the specified sort field. This procedure is then repeated for the second record and so on throughout all of the bond records. With more than 100 records, the Sort routine will take a minute or so.

OPTION 8 — END

The End routine is (or should be) self-explanatory. It does, however, give the user the option to abort (also self-explanatory).

SAMPLE RUN

As a sample run, we will compare two different bonds in order to determine the best investment strategy. For our purposes, we will assume that the best investment strategy is the one which yields the highest **AFTER TAX YIELD**.

The first bond, "ABC", is a bond which matures or comes **due on May 1, 1989** (most bonds are due on the first of the month). ABC has a **Face Value of \$1000**, a **Purchase Price of \$560** and pays an **annual interest rate of 2%**.

The second bond, "XYZ", matures on **April 1, 1992**. It pays an **annual interest rate of 11%**. XYZ has a **Face Value of \$1000**, and a **Purchase Price of \$840**. Assume that today's date is **January 1, 1981**.

Lastly, assume that your **incremental tax rate is 19%**. After entry of the data for these two bonds (see Option 1 above), on the **DISPLAY/EDIT** mode (Option 4) we find that the **After Tax yield for bond ABC is 11.43%** while the **After Tax yield for bond XYZ is 12.14%**. Consequently, the high interest bond XYZ is the better investment.

*** BOND YIELD CALCULATIONS ***

** TAX RATE 19% **

RECORD NO.	1	2
BOND NAME	ABC	XYZ
PURCHASE PRICE	560	840
FACE VALUE	1000	1000
MONTH DUE DATE	MAY 1	APR 1
YEAR DUE DATE	1989	1992
FACE INTEREST	2.00	11.00
GROSS YIELD	13.00	14.79
CAPITAL GN YLD	9.43	1.69
CURRENT YIELD	3.57	13.10
AFTER TAX YIELD	11.43	12.14

However, you have just gotten a promotion and a raise and your **incremental tax rate has jumped to 49%**. After resetting your tax rate (Option 6), the **After Tax yield for bond ABC drops somewhat to 8.94%** (Uncle Sam takes a bigger chunk of your income) but the **After Tax yield for bond XYZ drops even more dramatically to 7.96%**. Bond ABC, the low interest but high capital gains bond, is now the better buy. Please note that the low interest bond ABC has a better After Tax Yield than bond XYZ even though the Gross Yield for bond XYZ (14.79%) exceeds the Gross Yield for bond ABC (13.00%), all because of the different tax treatment for Capital gains income versus Ordinary income.

*** BOND YIELD CALCULATIONS ***

** TAX RATE 49% **

RECORD NO.	1	2
BOND NAME	ABC	XYZ
PURCHASE PRICE	560	840
FACE VALUE	1000	1000
MONTH DUE DATE	MAY 1	APR 1
YEAR DUE DATE	1989	1992
FACE INTEREST	2.00	11.00
GROSS YIELD	13.00	14.79
CAPITAL GN YLD	9.43	1.69
CURRENT YIELD	3.57	13.10
AFTER TAX YIELD	8.94	7.96

GENERAL PROGRAMMING NOTES

The routine which begins at line 110 is my own version of **PRINT USING** and converts the floating point variable VV in decimal format, e.g., 0.0650, to a two decimal string value VV\$, e.g. 6.50. This conversion routine first takes the integer part of the decimal value multiplied by 10000 and converts this to a string T\$ (line 190). VV\$ is then constructed by inserting PE\$ (".") before the second rightmost digit and then returns to the place from which it was called. Consequently, this conversion routine must be called each time a number is output to the screen in XX.XX format but it really isn't as slow as it looks. In addition, since Applesoft truncates floating point numbers in odd ways, it is necessary to add a correction factor of .00005 (variable C1) to the floating point value before calling this conversion routine.

After executing a GOSUB to this conversion routine, the number 0.0650 is converted into a string 6.50 (VV\$). VV\$, however, can

vary in length from four characters (e.g., 0.56%) up to six characters (e.g., 145.78%). Consequently, in order for the decimal points in the output to be aligned, it is necessary, after returning from the conversion routine, to determine the length of string VV\$ and then insert the proper number of spaces or blanks BEFORE VV\$ is output to the screen. Program lines 6330 and 6340 (among others) accomplish this purpose by establishing an output field that is six characters in length. The use of the ABS function in line 6340 (also among others) prevents the program from bombing in the event that the output field exceeds six characters.

My PRINT USING can be easily transported to your other programs.

BOND CALCULATIONS

The actual calculations of the bond data are performed in lines 3000 to 3170 of the program. Line 3120 determines if the holding period for the bond is greater than twelve months (variable TW) and, if not, all gains are calculated at the ordinary income rate. If the capital gain holding period should change, also change line 3120.

The printer output assumes a 60 line output (e.g., a Paper Tiger with automatic page feed). If your printer does not have automatic page feed, add the appropriate number of print statements at line 8000. In addition, **if your printer requires special control codes in order to operate, add them at the end of line 7170.**

Error trapping throughout the program has been handled in two different ways. First, the data entry is examined at the time of entry to protect against improper and/or ridiculous data values. Secondly, the routine at line

16000 is provided to trap disk errors over which the program has no control. The disk error trap routine preserves data in the event of an error and permits the user to return to the main Menu with his or her data unscathed.

The **data storage requirements** will vary depending on your Apple configuration. **With 48K and Applesoft in ROM, the program will easily handle 200 bonds in active memory (RAM).** If you wish to change the storage capability of the program, change not only the DIM statement in line 11210 but also the constant HH in line 11190.

THE WINDOWS

"Windows" are used extensively throughout the program and particularly in the Edit routine. Windows not only provide a less distracting appearance to the program than continually clearing and rewriting the screen but also prevent unintended scrolling of the top of the screen. This is particularly evident in line 10380 which sets a very small window at the lower left hand corner of the screen for entry of both the field number to be edited, as well as the new data for the field which is edited. After the new data has been entered, the Apple is returned to full screen in line 10570. Personally, I find it very handy to return the Apple to "full screen" when moving from one subroutine to another, since it ensures that the window is preset whenever any subroutine is called.

THE ARRAYS

The bond data as well as the calculated results are stored in the array X(N,Record#) where N equals one of the bond data values or calculated values. The bond name is stored in the string array BN\$(Record#). Those read-

ers with eagle eyes (as well as the critics) will quickly notice that the array value X(5,Record#) is unused. This was intentional, since someday, it may be convenient to have another value handy and variable X(5,Record#) is centered between the input data values and the calculated data values.

THE SUBROUTINES

The actual placement of the program subroutines within the program is also important for rapid and efficient running of the program. Subroutines which are frequently executed, as well as subroutines which are reiterative in nature (e.g. my XX.XX conversion routine and the Sort calculations), are placed near the beginning of the program for more rapid execution. For those who do not know it, whenever Applesoft encounters a "GOTO line number" or "IF Z THEN line number" program line, Applesoft searches for the line number from the beginning of the program. Consequently, if the Sort routine which starts at line 1000 is moved to the end of the program, your sorts will be extremely slow.

My last effort to speed the execution of the program was to **assign numerous constants to variable names** in program lines 11180 - 11200 when the program is first executed. **Applesoft interprets variable names within parentheses much more rapidly than constants.** For example, the program statement **Y = X(N5)** executes much more rapidly than the statement **Y = X(5).** I strongly recommend that you use variable names, rather than numeric values, in your own programs if program execution speed is desired.

That's about it. Before you buy a bond, run Bond Manager, and test the alternatives for your "Best Buy".



```

1
1 REM *****
2 REM * BOND MANAGER *
3 REM * BY DOUG SPRINKLE *
4 REM * COPYRIGHT (C) 1982 *
5 REM * BY MICRO-SPARC INC *
6 REM * LINCOLN, MA. 01773 *
7 REM *****
100 GOTO 11000
110 REM
120 REM
130 REM THIS ROUTINE CONVERTS
140 REM TO XX.XX FORMAT
150 REM
160 REM
170 REM
180 W = INT (W / TT)
190 T$ = STR$ (W) : SL = LEN (T$) - N2
200 IF SL < P5 THEN T$ = Z$ + T$ : SL = SL + N2
210 W$ = LEFT$ (T$,SL) + P$ + RIGHT$ (T$,N2)
220 RETURN
1000 REM
1010 REM START SORT
1020 REM
1030 FOR SN = N1 TO I
1040 MX = SN: REM MAX SUBSCRIPT
1050 T = X(A,SN)
1060 FOR SF = SN TO I
1070 IF X(A,SF) < T THEN 1090
1080 T = X(A,SF):MX = SF
1090 NEXT SF
1100 IF MX = SN THEN 1220
1110 REM
1120 REM SWAP FIELDS
1130 REM
1140 FOR SW = N8 TO N9
1150 T = X(SW,SN)
1160 X(SW,SN) = X(SW,MX)
1170 X(SW,MX) = T
1180 NEXT SW
1190 T$ = BN$(SN)
1200 BN$(SN) = BN$(MX)
1210 BN$(MX) = T$
1220 NEXT SN
1230 RETURN
1240 REM
1250 REM SORT FOR BOND NAME
1260 REM
1270 FOR SN = N1 TO I
1280 MX = SN:T$ = BN$(SN)
1290 FOR SF = SN TO I

```

```

1300 IF BN$(SF) > T$ THEN 1320
1310 T$ = BN$(SF):MX = SF
1320 NEXT SF
1330 GOTO 1100
2000 REM
2010 REM
2020 REM THIS ROUTINE DELETES RECORD
2030 REM
2040 CALL - 936: VTAB TW: HTAB 15: FLASH: PRINT "DELE
TING": NORMAL
2050 IF I > P5 THEN 2070
2060 POP: RETURN
2070 FOR MK = N1 TO I
2080 IF BN$(MK) < ) AS$ THEN 2170
2090 FOR M = MK TO I
2100 X = M + N1
2110 FOR MM = N8 TO N9
2120 X(MM,M) = X(MM,X)
2130 NEXT MM
2140 BN$(M) = BN$(X)
2150 NEXT M
2160 I = I - N1
2165 GOTO 2080
2170 NEXT MK
2180 DL = N8
2190 RETURN
3000 REM
3010 REM CALCULATE VALUES
3020 REM
3030 IF DT < P5 THEN GOSUB 4000
3040 IF DI < P5 THEN 3070
3050 JJ = I: CALL - 936: VTAB TW: HTAB 18: FLASH: PRINT
"WAIT": NORMAL
3060 FOR L = BR TO JJ
3070 X(N6,L) = X(N2,L) / HU
3080 MD = (X(N3,L) - CY) * TW + X(N4,L) - CH
3090 X(N8,L) = ((X(N1,L) - X(N8,L)) / X(N8,L)) / (MD / T
W)
3100 X(N9,L) = (X(N1,L) / X(N8,L)) * X(N6,L)
3110 X(N7,L) = X(N9,L) + X(N8,L)
3120 IF MD > TW THEN 3140
3130 X(N9,L) = X(N9,L) + X(N8,L):X(N8,L) = N8
3140 X(N9,L) = (N1 - TR) * (X(N9,L) + P5 * X(N8,L)) + P5
* X(N8,L)
3150 IF DI < P5 THEN RETURN
3160 NEXT L
3170 RETURN
4000 CALL - 936: VTAB TW
4010 PRINT " ENTER TODAY'S DATE": PRINT
4020 PRINT " BY MONTH, DAY, YR": PRINT: INPUT " (E.G. 1
1,3,1981) " : CH, CD, CY

```

continued on next page

Apple Bond Manager (Cont.)

```

4030 PRINT : PRINT : HTAB TW: PRINT MOD(CM); " "; CD; " ", "
;CY
4040 PRINT : PRINT : PRINT TAB( 12)*CORRECT (Y/N)?;
4050 GET A$: IF A$ = NL$ THEN 4050
4060 IF LEFT$(A$,N1) < > Y$ THEN 4060
4070 CM = CM + CD / F30
4080 CALL - 936
4090 PRINT : PRINT
4100 PRINT " ENTER YOUR MAXIMUM"
4110 PRINT : INPUT " INCREMENTAL TAX RATE ---- " ; A$: T
R = VAL (A$): IF TR < P5 THEN PRINT G$: GOTO 4080

4120 TR = TR / HU: DT = N1: CALL - 936: FLASH : VTAB TW:
HTAB 18: PRINT "WAIT": NORMAL : RETURN
4130 RETURN
5000 REM
5010 REM
5020 REM INPUT DATA FROM
5030 REM KEYBOARD ROUTINE
5040 REM
5050 REM
5060 CALL - 936: JX = I + N1
5070 FOR I = JX TO HH
5080 CALL - 936
5090 INPUT " ENTER BOND NAME " ; BNS(I): PRINT

5100 BNS(I) = LEFT$(BNS(I),N5)
5110 INPUT " ENTER PURCHASE PRICE $ " ; A$: T = VAL
(A$): IF T < P5 THEN HTAB N1: VTAB N2: PRINT G$: GOTO
5110
5120 PRINT : X(N0,I) = T
5130 INPUT " ENTER BOND FACE VALUE $ " ; A$: T = VAL
(A$): IF T < P5 THEN HTAB N1: VTAB N4: PRINT G$: GOTO
5130
5140 PRINT : X(N1,I) = T
5150 INPUT " ENTER INT. RATE ( % ) " ; A$: T = VAL
(A$): IF T < P5 THEN HTAB N1: VTAB N6: PRINT G$: GOTO
5150
5160 PRINT : X(N2,I) = T
5170 INPUT " ENTER DUE DATE ( MO.DAY ) " ; A$: A = VAL
(A$)
5180 IF A = > N4 OR A < N1 THEN PRINT G$: HTAB N1: VTAB
N9: GOTO 5170
5190 T = HU * (A - INT (A)): IF T > ND THEN PRINT G$: HTAB
N1: VTAB N9: GOTO 5170
5200 X(N4,I) = TD * (A - INT (A)) + INT (A): IF X(N4,I)
= VAL (A$) THEN X(N4,I) = X(N4,I) + TD / HU
5210 PRINT
5220 INPUT " ENTER DUE DATE (YR 1982) " ; A$: T = VAL
(A$): IF T < P5 THEN HTAB N1: VTAB N0: PRINT G$: GOTO
5220
5230 IF LEN (A$) < > N4 THEN PRINT G$: HTAB N1: VTAB
11: GOTO 5220
5240 X(N3,I) = T
5250 PRINT
5260 PRINT : PRINT
5270 CC = I:L = CC
5280 GOSUB 10000
5290 IF NS = N1 THEN NS = N0: GOSUB 3030: RETURN
5300 IF NS = N2 THEN NS = N0: CALL - 936: GOTO 5090
5310 IF NS = N4 THEN NS = N0: CALL - 936: GOSUB 3030: GOTO
5400
5320 POKE 33,10: POKE 32,30: VTAB N1: HTAB 35: POKE 35,
18: REM SET WINDOW
5330 CALL - 936: PRINT BNS(I): PRINT : PRINT X(N0,I): PRINT

5340 PRINT X(N1,I): PRINT
5350 PRINT X(N2,I): PRINT
5360 A = INT (X(N4,I)): T = INT (HU * (X(N4,I) - A + CO
) / TD)
5370 A$ = STR$(A): T$ = STR$(T): IF LEN (T$) < N2 THEN
T$ = "0" + T$
5380 PRINT A$ + PE$ + T$: PRINT
5390 PRINT X(N3,I): PRINT : POKE 33,40: POKE 32,0: POKE
35,24: GOTO 5280
5400 NEXT I
5410 RETURN
6000 REM
6010 REM
6020 REM THIS ROUTINE OUTPUTS
6030 REM ANSWERS TO THE SCREEN
6040 REM
6050 REM
6060 CALL - 936: KZ = N1
6070 VTAB 8: PRINT TAB( 5)*ENTER RECORD NO.: PRINT : INPUT
" (RTN) FOR ALL RECORDS " ; A$
6080 KZ = VAL (A$): IF KZ < P5 THEN KZ = N1
6090 IF KZ > I THEN RETURN
6100 CALL - 936
6110 PRINT TAB( N4)*BOND NAME: PRINT : PRINT TAB( N4
)*PURCHASE PRICE $: PRINT : PRINT TAB( N4)*BOND
FACE VALUE $: PRINT : PRINT TAB( N4)*FACE INTERE
ST (%): PRINT
6120 PRINT TAB( N4)*DUE DATE (MO-DAY): PRINT : PRINT
TAB( N4)*DUE DATE ( YEAR ): PRINT : PRINT TAB( N
4)*GROSS YIELD (%): PRINT : PRINT TAB( N4)*CAPITA
L GN YLD (%): PRINT
6130 PRINT TAB( N4)*CURRENT YIELD (%): PRINT : PRINT
TAB( N4)*AFTER TAX YIELD (%): PRINT
6140 FOR K = KZ TO I
6150 REM
6160 REM SET WINDOW
6170 REM
6180 HTAB 38: POKE 33,15: POKE 32,24: POKE 35,B1: CALL
- 936
6190 T = LEN (BNS(K)): S = ABS (N6 - T)
6200 PRINT SPC( S)BNS(K); " RCD "; K
6210 PRINT
6220 T$ = STR$(X(N0,K)): T = LEN (T$): S = ABS (N6 - T
)
6230 PRINT SPC( S)T$: PRINT

```

```

6240 T$ = STR$(X(N1,K)): T = LEN (T$): S = ABS (N6 - T
)
6250 PRINT SPC( S)T$: PRINT
6260 W = X(N6,K) + C1: GOSUB 100
6270 T = LEN (W$): S = ABS (N6 - T)
6280 PRINT SPC( S)W$: PRINT
6290 A = X(N4,K): B = INT (A)
6300 W$ = STR$( INT (HU * (A - B + CO) / TD)): S = ABS
(N3 - LEN (W$))
6310 PRINT SPC( S)MOD( INT (X(N4,K))): W$: PRINT
6320 PRINT SPC( N2)X(N3,K): PRINT
6330 W = X(N7,K) + C1: GOSUB 100
6340 T = LEN (W$): S = ABS (N6 - T)
6350 PRINT SPC( S)W$: PRINT
6360 PRINT
6370 W = X(N8,K) + C1: GOSUB 100
6380 T = LEN (W$): S = ABS (N6 - T)
6390 PRINT SPC( S)W$: PRINT
6400 PRINT
6410 W = X(N9,K) + C1: GOSUB 100
6420 T = LEN (W$): S = ABS (N6 - T)
6430 PRINT SPC( S)W$: PRINT
6440 PRINT
6450 W = X(N0,K) + C1: GOSUB 100
6460 T = LEN (W$): S = ABS (N6 - T)
6470 PRINT SPC( S)W$: PRINT
6480 REM RESET WINDOW
6490 POKE 32,0: POKE 33,40
6500 CC = K: GOSUB 10000
6510 B1 = 20
6520 IF NS = N1 THEN NS = N0: GOTO 6000
6530 IF NS = N4 THEN NS = N0: GOTO 6590
6540 IF NS = N3 THEN 6570
6550 BNS(K) = A$: NS = N0: DL = N1: REM SET
DELETE FLAG
6560 GOTO 6590
6570 NS = N0: N = CC: L = CC
6580 GOSUB 3070: SC = N0: GOTO 6180
6590 NEXT K
6600 B1 = 24: IF DL > P5 THEN GOSUB 2000: DL = N0
6610 RETURN
7000 REM
7010 REM PRINTER OUTPUT
7020 REM
7030 CALL - 936
7040 INPUT "PRINT RESULTS? (Y/N) (RET) " ; A$
7050 PRINT
7060 IF LEFT$(A$,N1) < > Y$ THEN RETURN
7070 PRINT "TURN ON PRINTER AND": PRINT
7080 INPUT "ENTER PRINTER SLOT NUMBER (RET) " ; A$: N = VAL
(A$)
7090 IF N < N1 OR N > N7 THEN PRINT G$: GOTO 7030
7100 CALL - 936: VTAB 7: PRINT "ENTER FIRST RECORD NO.
OR": INPUT "PUSH (RET) FOR FIRST RECORD - " ; A$: PRINT

7110 IF A$ = NL$ THEN KK = 1: GOTO 7130
7120 KK = VAL (A$)
7130 IF KK < P5 THEN KK = 0N
7140 PRINT "ENTER LAST RECORD NO. OR": INPUT "PUSH (RET
) FOR LAST RECORD - " ; A$
7150 IF A$ = NL$ THEN LR = 1: GOTO 7170
7160 LR = VAL (A$)
7170 CALL - 936: PRINT TAB( 6)*PUSH ' S ' TO HALT PRI
NTING*: VTAB 4: HTAB 1: POKE 34,3: PR# N
7180 REM THIS ROUTINE OUTPUTS TO PRINTER
7190 KN = KK + 6: IF KN > LR THEN KN = LR
7200 PRINT CHR$( N4): PRINT SPC( 15)*BOND YIELD C
ALCULATIONS ***: PRINT
7210 T = INT (TR * HU + .5)
7220 PRINT SPC( 20)*TAX RATE " ; T: " % ***: PRINT
7230 PRINT "RECORD NO. " ;
7240 FOR K = KK TO KN
7250 T$ = STR$(K): T = LEN (T$): S = N8 - T
7260 PRINT SPC( S)T$: PRINT
7270 NEXT K
7280 PRINT CHR$( N4)
7290 GOSUB 8030
7300 PRINT "BOND NAME " ;
7310 FOR K = KK TO KN
7320 IF LEN (BNS(K)) > N8 THEN BNS(K) = LEFT$(BNS(K)
,N8)
7330 T = LEN (BNS(K)): S = ABS (N8 - T)
7340 PRINT SPC( S)BNS(K);
7350 NEXT K
7360 PRINT CHR$( N4)
7370 GOSUB 8030
7380 PRINT "PURCHASE PRICE " ;
7390 FOR K = KK TO KN
7400 T$ = STR$(X(0,K)): T = LEN (T$): S = N8 - T
7410 PRINT SPC( S)T$: PRINT
7420 NEXT K: PRINT CHR$( N4)
7430 GOSUB 8030
7440 PRINT "FACE VALUE " ;
7450 FOR K = KK TO KN
7460 T$ = STR$(X(N1,K)): T = LEN (T$): S = N8 - T: PRINT
SPC( S)T$: NEXT K
7470 PRINT CHR$( N4)
7480 GOSUB 8030
7490 PRINT "MONTH DUE DATE " ;
7500 FOR K = KK TO KN
7510 T = X(N4,K): A = INT (T): T$ = MOD(A)
7520 W$ = STR$( INT (HU * (T - A + CO) / TD)): T = LEN
(W$): S = N5 - T
7530 PRINT SPC( S)T$: W$: PRINT
7540 NEXT K
7550 PRINT CHR$( N4)
7560 GOSUB 8030
7570 PRINT "YEAR DUE DATE " ;
7580 FOR K = KK TO KN
7590 T$ = STR$(X(3,K)): S = 4
7600 PRINT SPC( S)T$: PRINT
7610 NEXT K
7620 PRINT CHR$( N4)
7630 GOSUB 8030
7640 PRINT "FACE INTEREST " ;

```



```

7650 FOR K = KK TO KN
7660 W = X(N6,K) + C1: GOSUB 180: T = LEN (W%): S = ABS
(N8 - T)
7670 PRINT SPC( S)W%;
7680 NEXT K
7690 PRINT CHR$( NJ)
7700 GOSUB 8030
7710 PRINT "GROSS YIELD ";
7720 FOR K = KK TO KN
7730 W = X(N7,K) + C1: GOSUB 180: T = LEN (W%): S = ABS
(N8 - T)
7740 PRINT SPC( S)W%;
7750 NEXT K
7760 PRINT CHR$( NJ)
7770 GOSUB 8030
7780 PRINT "CAPITAL GN YLD ";
7790 FOR K = KK TO KN
7800 W = X(N8,K) + C1: GOSUB 180: T = LEN (W%): S = ABS
(N8 - T)
7810 PRINT SPC( S)W%;
7820 NEXT K
7830 PRINT CHR$( NJ)
7840 GOSUB 8030
7850 PRINT "CURRENT YIELD ";
7860 FOR K = KK TO KN
7870 W = X(N9,K) + C1: GOSUB 180: T = LEN (W%): S = ABS
(N8 - T)
7880 PRINT SPC( S)W%;
7890 NEXT K
7900 PRINT CHR$( NJ)
7910 PRINT "AFTER TAX YIELD ";
7920 FOR K = KK TO KN
7930 W = X(N0,K) + C1: GOSUB 180: T = LEN (W%): S = ABS
(N8 - T)
7940 PRINT SPC( S)W%;
7950 NEXT K
7960 PRINT CHR$( NJ)
7970 IF KN >= LR THEN 8070
7980 KK = KK + 7: KN = KK + 6
7990 IF KN > LR THEN KN = LR
8000 PRINT : PRINT
8010 GOTO 7200
8020 REM KEY DEPRESS SENSE
8030 B = PEEK ( - 16384): POKE - 16368,0: IF B < 127 THEN
RETURN
8040 IF B = 211 THEN POP : GOTO 8070
8050 RETURN
8060 REM RESET WINDOW AND RECONNECT DOS
8070 POKE 34,8: CALL - 935: CALL - 365: CALL 1802: RETURN

10000 REM
10010 REM
10020 REM EDIT ROUTINE
10030 REM
10040 NS = 0: POKE 35,24: REM RESET BOT MAR
10050 VTAB 21: HTAB 25: PRINT "R RTN TO MENU"
10060 VTAB 22: HTAB 25: PRINT "D DELETE RCD"
10070 VTAB 23: HTAB 25: PRINT "E EDIT RCD"
10080 VTAB 24: HTAB 24: PRINT "RTN NEXT RCD";
10090 HTAB 1
10100 POKE 33,20: POKE 32,0: POKE 34,22
10110 GET A$: IF A$ = NL$ THEN 10110
10120 IF A$ < > "R" THEN 10150
10130 POKE 33,40: POKE 32,0: POKE 34,0
10140 NS = 1: RETURN
10150 IF A$ < > "D" THEN 10180
10160 POKE 33,40: POKE 32,0: POKE 34,0
10170 NS = 2: RETURN
10180 IF A$ < > "E" THEN 10210
10190 POKE 33,40: POKE 32,0: POKE 34,0
10200 NS = 3: GOTO 10250
10210 IF A$ = CHR$( NJ) THEN 10230
10220 PRINT G$: GOTO 10110
10230 POKE 33,40: POKE 32,0: POKE 34,0
10240 NS = 4: RETURN
10250 REM
10260 REM
10270 REM EDIT ROUTINE
10280 REM
10290 REM SET WINDOW
10300 POKE 33,3
10310 CALL - 936: PRINT "1 -"
10320 PRINT "2 -"
10330 PRINT "3 -"
10340 PRINT "4 -"
10350 PRINT "5 -"
10360 PRINT "6 -": VTAB 23: HTAB 1
10370 POKE 33,40: REM RESET WINDOW
10380 POKE 33,20: POKE 32,0: POKE 34,21
10390 REM
10400 CALL - 936: PRINT "ENTER FIELD NO."
10410 GET A$: IF VAL (A$) < P5 OR VAL (A$) > 6.5 THEN
PRINT G$: GOTO 10400
10420 A = VAL (A$)
10430 CALL - 936
10440 PRINT "ENTER NEW " : PRINT EF$(A); " " : INPUT
A$
10450 IF A = 1 THEN B$(CC) = LEFT$( A$,N5): GOTO 1057
8
10460 IF VAL (A$) = N0 THEN PRINT G$: GOTO 10390
10470 IF A = N2 THEN X(N0,CC) = VAL (A$): GOTO 10570
10480 IF A = N3 THEN X(N1,CC) = VAL (A$): GOTO 10570
10490 IF A = N4 THEN X(N2,CC) = VAL (A$): GOTO 10570
10500 IF A < > N5 THEN 10550
10510 A = VAL (A$): IF A < P5 OR A >= NJ THEN PRINT
G$: GOTO 10390
10520 T = HU * (A - INT (A)): IF T > N0 THEN PRINT G$:
GOTO 10390
10530 X(N4,CC) = TD * (A - INT (A)) + INT (A): IF X(N4
,CC) = VAL (A$) THEN X(N4,CC) = X(N4,CC) + TD / HU
8
10540 GOTO 10570
10550 IF A = N6 AND LEN (A$) < > N4 THEN PRINT G$: GOTO
10390

```

```

10560 X(N3,CC) = VAL (A$): GOTO 10570
10570 CALL - 936: POKE 33,3: POKE 34,0: CALL - 936: POKE
33,40: RETURN
10000 REM
10010 EF$(1) = "BOND NAME"
10020 EF$(2) = "PUR PRICE"
10030 EF$(3) = "FACE VALUE"
10040 EF$(4) = "INTER RATE"
10050 EF$(5) = "DUE DATE(MO)"
10060 EF$(6) = "DUE DATE(YR)"
10070 REM B$ = BOND NAME
10080 REM X(6,X)=BOND INTEREST RATE
10090 REM X(7,X)=GROSS YIELD
10100 REM X(8,X)=CAPITAL GAIN YIELD
10110 REM X(9,X)=CURRENT YIELD
10120 REM X(10,X)=REAL YIELD
10130 REM X(0,X) = BOND PRICE
10140 REM X(N1,X) = BOND FACE VALUE
10150 REM X(2,X) = BOND INTEREST
10160 REM X(3,X) = DUE DATE (YR)
10170 REM X(4,X) = DUE DATE (MO)
10180 TT = 10000: HU = 100: Z$ = "00": P$ = " ": TW = 12: P
5 = .4: A$ = "": N0 = 0: N1 = 1: Y$ = "Y": ST$ = "S"
": NL$ = "": ND = 31: REM P5 HAS BEEN CHANGED TO .4
- CAPITAL GAIN RATE IS NOW 40%
11190 N5 = 5: N2 = 2: N3 = 3: N4 = 4: F30 = 30: HH = 200: N6 =
6: N7 = 7: N8 = 8: N9 = 9: N0 = 10: NJ = 13: EM = 24: TD =
3.3333: CO = .005: C1 = .00005
11200 G$ = " ": REM G$ = (CONTROL) G
11210 DIM B$(200), M$(12), X(10,200), EC$(15)
11220 GOSUB 17000
11230 REM
11240 REM
11250 REM
11260 REM MENUE
11270 REM
11280 REM
11290 CALL - 936
11300 PRINT SPC( 14)*** MENUE ***: PRINT : PRINT
11310 PRINT TAB( N5)*1 - ENTER DATA FROM KBD: PRINT
11320 PRINT TAB( N5)*2 - LOAD DATA FROM DISC: PRINT
11330 PRINT TAB( N5)*3 - SAVE DATA TO DISC: PRINT
11340 PRINT TAB( N5)*4 - DISPLAY/EDIT DATA: PRINT
11350 PRINT TAB( N5)*5 - PRINT RESULTS: PRINT
11360 PRINT TAB( N5)*6 - CHANGE TAX RATE: PRINT
11370 PRINT TAB( N5)*7 - SORT: PRINT
11380 POKE 36,N5 - 1: PRINT "8 - END": HTAB 27: PRINT
I; " ": INVERSE : PRINT "RECORDS": NORMAL : PRINT :
PRINT
11390 PRINT TAB( N8)"ENTER SELECTION ";
11400 HTAB 1: VTAB 24: POKE 34,23
11410 GET A$: IF A$ = NL$ THEN 11410
11420 A = VAL (A$)
11430 IF A > P5 AND A < 8.5 THEN 11450
11440 PRINT G$: GOTO 11410
11450 POKE 34,0
11460 ON A GOSUB 5060,14000,15000,6060,7030,12000,13000
,11400
11470 GOTO 11230
11480 PRINT : PRINT " ARE YOU SURE? (Y/N)"
11490 GET A$: IF A$ = NL$ THEN 11490
11500 IF A$ < > "Y" THEN 11230
11510 END
12000 REM
12010 REM THIS ROUTINE RESETS
12020 REM THE TAX RATE AND
12030 REM RECALCULATES
12040 CALL - 936: VTAB 12: MD = 1
12050 GOSUB 4090: DI = 1: BR = N1: GOSUB 3030
12060 DI = 0: RETURN
13000 REM
13010 REM
13020 REM SORT ROUTINE
13030 REM
13040 REM
13050 CALL - 936
13060 VS(N0) = N0: VS(N1) = N6: VS(N2) = N7: VS(N3) = N8: VS
(N4) = N9: VS(N5) = N0: VS(N6) = N0: VS(N7) = N1: VS(N8
) = N3: VS(N9) = N4
13070 PRINT TAB( 10)*** SORT ROUTINE ***: PRINT : PRINT
13080 PRINT TAB( N5)*0 - BOND NAME"
13090 PRINT TAB( N5)*1 - BOND INTEREST"
13100 PRINT TAB( N5)*2 - GROSS YIELD"
13110 PRINT TAB( N5)*3 - CAPITAL GAIN YIELD"
13120 PRINT TAB( N5)*4 - CURRENT YIELD"
13130 PRINT TAB( N5)*5 - AFTER TAX YIELD"
13140 PRINT TAB( N5)*6 - BOND PURCHASE PRICE"
13150 PRINT TAB( N5)*7 - BOND FACE VALUE"
13160 PRINT TAB( N5)*8 - BOND DUE DATE(YR)"
13170 PRINT TAB( N5)*9 - BOND DUE DATE(MO)": PRINT
13180 PRINT TAB( N3)"(RET) - RETURN TO MENUE": PRINT :
PRINT
13190 PRINT TAB( 5)"ENTER SORT FIELD";
13200 GET A$: IF A$ = NL$ THEN 13200
13210 IF A$ = CHR$( NJ) THEN RETURN
13220 IF A$ = "0" THEN A = 0: GOTO 13240
13230 A = VAL (A$): IF A < P5 OR A > 9.5 THEN PRINT G$
: GOTO 13190
13240 CALL - 936: VTAB TW: HTAB 16: FLASH : PRINT "SOR
TING": NORMAL
13250 IF A = N0 THEN 1270
13260 A = VS(A)
13270 GOTO 1000
14000 ONERR GOTO 16000
14010 REM
14020 REM THIS ROUTINE LOADS DATA
14030 REM FROM FILE "BONDDATA"
14040 CALL - 936: FOR J = N1 TO N8: PRINT G$: NEXT J
14050 VTAB 11: PRINT TAB( 12)"INSERT DATA DISK"
14060 VTAB 13: HTAB 12: INPUT "AND PRESS RETURN": A$
14070 D$ = CHR$( 4): BR = N1: JJ = N0: CALL - 936: VTAB
TW: IF I < P5 THEN 14130

```

continued on next page

Apple Bond Manager (Cont.)

```

14000 PRINT "APPEND TO EXISTING FILE? (Y/N) ";
14090 GET A$: IF A$ = NL$ THEN 14090
14100 IF LEFT$(A$,N1) < > Y$ THEN I = N0:BR = N1: GOTO
14120
14110 JJ = 1:BR = 1
14120 PRINT : PRINT
14130 PRINT "ENTER FILE IDENTIFIER "
14140 PRINT : PRINT "OR PRESS 'C' FOR CATALOG"
14150 PRINT : PRINT "OR 'M' TO RETURN TO MENU " ;
14160 GET FF$: IF FF$ = NL$ THEN 14160
14170 IF FF$ = "M" THEN RETURN
14180 IF FF$ < > "C" THEN 14240
14190 PRINT " " : CALL - 936
14200 PRINT D$:"CATALOG"
14210 PRINT : PRINT "PRESS ANY KEY TO CONTINUE";
14220 GET A$: IF A$ = NL$ THEN 14220
14230 GOTO 14000
14240 CALL - 936: VTAB TW: HTAB TW: NORMAL
14250 PRINT "READING BOND DATA";FF$
14260 PRINT D$:"OPEN BONDDATA";FF$
14270 PRINT D$:"READ BONDDATA";FF$
14280 INPUT KK
14290 I = I + KK
14300 IF I > HH THEN I = HH
14310 FOR J = JJ + N1 TO I
14320 INPUT BN$(J)
14330 INPUT X(N0,J)
14340 INPUT X(N1,J)
14350 INPUT X(N2,J)
14360 INPUT X(N4,J)
14370 INPUT X(N3,J)
14380 NEXT J
14390 PRINT D$:"CLOSE BONDDATA";FF$
14400 DI = N1: GOSUB 3030:DI = N0: RETURN : REM CA
LCULATE RESULTS
15000 ONERR GOTO 16000
15010 REM
15020 REM THIS ROUTINE SAVES DATA
15030 REM TO FILE 'BONDDATA'
15040 REM
15050 IF I > P5 THEN 15090
15060 CALL - 936: VTAB 11: PRINT TAB( 12) "NO DATA IN
MEMORY"
15070 VTAB 13: HTAB 14: INPUT "PRESS RETURN";A$
15080 RETURN
15090 CALL - 936: FOR J = N1 TO N0: PRINT G$: NEXT J
15100 VTAB 11: PRINT TAB( 12) "INSERT DATA DISK"
15110 VTAB 13: HTAB 12: INPUT "AND PRESS RETURN";A$
15120 CALL - 936: VTAB TW: HTAB TW
15130 PRINT "ENTER FILE NUMBER ";
15140 GET FF$: IF FF$ = NL$ THEN 15140
15150 REM
15160 CALL - 936: VTAB TW: HTAB TW: NORMAL
15170 PRINT "WRITING BOND DATA";FF$
15180 D$ = CHR$( 4)
15190 PRINT D$:"OPEN BONDDATA";FF$
15200 PRINT D$:"DELETE BONDDATA";FF$
15210 PRINT D$:"OPEN BONDDATA";FF$
15220 PRINT D$:"WRITE BONDDATA";FF$
15230 PRINT I
15240 FOR J = N1 TO I
15250 PRINT BN$(J)
15260 PRINT X(N0,J)
15270 PRINT X(N1,J)
15280 PRINT X(N2,J)
15290 PRINT X(N4,J)
15300 PRINT X(N3,J)
15310 NEXT J
15320 PRINT D$:"CLOSE BONDDATA";FF$
15330 NORMAL
15340 RETURN
16000 REM
16010 REM
16020 REM ERROR ROUTINE
16030 REM
16040 REM
16050 EC = PEEK (222)
16060 GOTO 16160
16070 CALL - 936: FOR T = N1 TO N0: PRINT G$: NEXT T
16090 VTAB 12: PRINT " *** DISK ERROR ***"
16130 PRINT : PRINT EC$(EC)
16140 PRINT : INPUT "PUSH RETURN TO CONTINUE";A$
16150 GOTO 11230
16160 EC$(1) = "LANGUAGE NOT AVAILABLE"
16170 EC$(4) = "WRITE PROTECTED"
16180 EC$(5) = "END OF DATA"
16190 EC$(6) = "FILE NOT FOUND"
16200 EC$(8) = "I/O ERROR"
16210 EC$(9) = "DISK FULL"
16220 EC$(10) = "FILE LOCKED"
16230 EC$(11) = "SYNTAX ERROR"
16240 GOTO 16070
17000 T$ = "JANFEBMARAPRMAJUNJLYAUGSEPCTOBNVDEC"
17010 T = - N2
17020 FOR J = N1 TO TW
17030 T = T + N3
17040 MO$(J) = MID$( T$,T,N3)
17050 NEXT J: RETURN

```

**DOS
3+2**



APPLE DOS 3 + 2

Apple DOS 3+2

by Joel Buckley
1212 Broadway
Hanover, PA 17331

INTRODUCTION

There are three main advantages to having the new 16-sector DOS 3.3. There's a 23% increase in disk storage space, the speed of disk I/O is substantially increased, and it is the new standard for Apple computer software. The one big disadvantage is its incompatibility with the earlier 13-sector versions of Apple DOS. This article presents a program, called **DOS 3+2**, that **provides co-resident 13 and 16 sector DOS** and goes a long way towards solving problems associated with having two incompatible disk formats.

PROBLEMS OF TWO DOS's

What are some of the problems caused by the incompatibility of DOS 3.3 with DOS 3.2?

To use the 13-sector formatted disks now requires a cumbersome two disk booting process. You no longer have the nice autorun feature with DOS 3.2 software. But why would anyone still be using 13-sector disks? Well, if you've accumulated a lot of programs and data on the old disks, you just may not feel up to converting them all to DOS 3.3. Or maybe you trade disks with some old diehards who haven't gotten DOS 3.3 yet. More importantly, much of the Apple software available today still comes only in the 13-sector format. It's easier to produce only one format, and while DOS 3.3 owners can run DOS 3.2, the reverse is not true. Also, a lot of commercial software is in a copy-protected format that can't even be transferred to 16-sector DOS. And finally, if you do decide to transfer to DOS 3.3 you have to deal with the notorious "MUFFIN" program.

FILE TRANSFER

The advertising describing DOS 3.3 is deceptive. It claims to include a utility program to convert 13-sector disks to 16-sector. Not quite! The program "MUFFIN" **doesn't convert** disks, but rather **transfers** files from a 13-sector to a 16-sector disk. Picky, Picky, you say. Transfer, not convert — what's the big deal? The big deal is that MUFFIN **transfers ONE FILE AT A TIME**. So if you have fifty files on a disk and only one disk drive, MUFFIN'ing that one disk requires inserting and removing disks **ONE HUNDRED TIMES!** Multiply that by thirty or forty disks and you're ready to go back to cassette tape.

Before DOS 3.3, Apple didn't provide any program at all for copying a disk with a single disk drive. Even "FID" uses the same excruciatingly slow, one file at a time technique as "MUFFIN". And try programming in Pascal with one drive — it is an exercise in futility!

Fortunately, single drive disk copy programs have been written that can transfer all the files on a disk in four passes or less. And single drive, whole disk "MUFFIN" programs have been written as well. In fact, it was to provide an environment for such a program that I wrote my first version of DOS 3+2 (for **DOS 3.3 and DOS 3.2**).

FEATURES OF DOS 3+2

DOS 3+2 is an assembly language program that relocates the 13-sector disk I/O routines from "MUFFIN" and appends them to 48K 16-sector DOS 3.3.

The version of DOS to be used is selectable from within any program via a single POKE command.

The DOS version can also be changed at any time (even while running an unmodified program) via a keyboard intercept switching routine.

Additionally, DOS 3+2 features error-driven automatic switching. **Any disk read error will lead to the other DOS being tried automatically!!**

Any disks INIT'ed under 16-sector DOS will contain the entire DOS 3+2. (INIT under DOS 3.2.1 is disabled.)

BACKGROUND

To understand how DOS 3+2 works requires some knowledge of the structure of DOS. A breakdown of 48K DOS 3.3 into its major functional components is presented in TABLE 1.

TABLE 1. MAJOR PARTS OF DOS

Address	Description
BFFF-B600	RWTS — (Read or Write a Track and Sector) low level disk access routines
B5FF-AAC1	FILE MANAGER — disk file command routines (calls RWTS)
AAC0-9D00	OPERATING SYSTEM — BASIC interface, command interpreter (calls File Manager)
9CFF-9600	FILE BUFFERS (MAXFILES3)

Comparing DOS 3.2.1 and DOS 3.3 reveals that the **operating system** and **file manager** portions of the two DOS's are almost identical, but the **RWTS** sections are substantially different. So we can use the same operating system and file manager sections for both DOS's and just have different RWTS sections.

The program "MUFFIN" from the DOS 3.3 System Master serves as the source for the 13-sector RWTS routines. These routines will be relocated to just below DOS 3.3. A comparison of the locations of these routines is given in TABLE 2 for normal 48K DOS 3.2.1, for MUFFIN, and for DOS 3+2.

TABLE 2. DOS 3.2.1 RWTS LOCATION

48K DOS	MUF-FIN	DOS 3+2	
BF	—	—	(used only by INIT)
BE	1F	9C	
BD	1E	9B	
BC	1D	9A	
BB	1C	99	
BA	1B	98	
B9	1A	97	
B8	19	96	

B7	—	—	(used only by INIT)
B6	—	—	(used only during boot)

PROGRAM LOGIC

Refer to LISTING 1.

The program was assembled on the **Apple Pascal 6502 assembler** and transferred to Apple DOS using a special Pascal<=>DOS File Transfer Program I wrote. A few notes on some non-standard features of this assembler are in order. The assembler uses the unconventional "@" instead of parentheses to indicate the indirect addressing mode. The percent symbol "%" is used to indicate the MODULO operation (remainder division). Oddly, the assembler also uses the "%" symbol as the external parameter token in .MACRO's. Since this prevents use of MODULO in .MACRO's, I patched the assembler to use the question mark "?" as the external parameter token in .MACRO's. Another bug in the assembler gives an error when numbers greater than \$8000 are divided (or moduloed) by \$100 and used as immediate addressing mode arguments. The unusual coding in the .MACRO labeled PATCH compensates for this bug.

Though the assembly listing is pretty well commented, some more detailed explanation will be useful in understanding how DOS 3+2 works and in applying these techniques to other programs. This explanation will be divided into two main parts: 1) explaining how DOS 3+2 is created by the program; and 2) understanding how the modifications to normal DOS work. You can enter the Hex Code directly into your Apple (see **Entering the Program** later in this article).

CREATING DOS 3+2

The action starts at \$323C with a check to see that 48K DOS 3.3 is present. Then the operator is prompted to insert a disk containing the program "MUFFIN". (The PRINT subroutine (at \$3225-\$323B) doesn't have the string length restriction of 127 or 255 characters that many others do, and can easily be used in other assembly language programs.)

After BLOADing "MUFFIN", the DOS 3.2.1 RWTS routines are relocated to just under DOS 3.3. Since these routines occupy \$700 bytes, when relocated they will occupy the space from **\$9600-\$9CFF**. Relocating machine language code involves more than just moving the code to a new address. All address references in the code must be changed to reflect the new address range. DOS also contains segments which are composed of data (rather than instructions). This data may represent a lot of different things, some of which will be the same value at any address, and other items (like tables of addresses) which have to be modified for the new location. To correctly modify this requires detailed information on all the data segments. (More discussion of code relocation can be found in the manual for Apple's Programmer's Aid #1 ROM, which contains a code relocation program.)

Fortunately, a DOS relocation routine already is available on any DOS 3.2.1 Master

continued on next page

Apple DOS 3 + 2 (Cont.)

Disk (one bootable on any memory size machine). I have simply adapted it to work here. This routine disassembles (using the monitor) the code in a range of addresses (found in a table at \$3003-\$300A). Any three byte instructions found are checked to see if the addresses are in the range being relocated, and if so they are adjusted by the offset between the source and destination addresses. Finally, the relocated code is actually moved to its destination.

The rest of the program moves three additional sections of code into DOS, makes some other patches to DOS, and jumps to the DOS coldstart routine. The purpose of these patches and the extra code is explained in the next section.

HOW DOS 3+2 WORKS

THE RWTS VECTOR

Conveniently, all calls to the RWTS routines in DOS are vectored through one subroutine (at \$B7B5-\$B7C1). The various command parameters are stored in a table pointed to by the 6502 A and Y registers. This routine is patched to jump to \$BCE0 (a normally unused section of DOS), where the short section of code labeled RWTSVEC (\$31BB-\$31C4) has been moved. This patch resets a flag for use in the error-driven switching routine. Changing a single byte in the address field of the final JMP instruction here is all that it takes to select between DOS 3.3 and DOS 3.2.1!

THE KEYBOARD INTERCEPT

The DOS keyboard input processing routine is intercepted at \$9EAF and sent to \$9CB1 (in the middle of another unused section of DOS where the DOS switching routines from \$31C5-\$3224 are now located). If the key is control-A then DOS 3.2.1 is selected, if it's control-B then DOS 3.3 is picked. This process is otherwise invisible to DOS and to any program that might be running at the time.

This keyboard intercept routine is somewhat unusual in that it is **not** done by changing the vector at \$38-\$39 (that is too easily disconnected). Also, note that it is **only a keyboard intercept** — a control-A [or CHR\$(1)] in a PRINT statement will not cause any DOS switching.

THE ERROR-DRIVEN SWITCH

To compensate for programs which disconnect the DOS keyboard intercept, or for instances when an operator fails to select the proper DOS, I added a routine which causes the computer to automatically try the other DOS if it encounters a disk read error using the currently active version of DOS. (Now that's what you call a user-friendly feature!)

Disk read errors from both RWTS sections are vectored to \$9CC1 (where the routine labeled ERRSWX at \$31E6 is moved to). This causes the other DOS to be tried in case of a disk read error. If a disk read error recurs, it switches back to the DOS originally in use and exits through the regular DOS error routine.

OTHER ROUTINES

The FLIPX routine (or a portion of it) is called whenever DOS is switched either through the keyboard or the error switch. This routine does two things besides changing the RWTS vector. It rings twice for DOS 3.2.1 and three times for DOS 3.3 to provide the operator positive assurance that the switch has been made. It also updates the DOS version number (3.2 or 3.3), which is displayed whenever a CATALOG command is given. (The string "DISK VOLUME" is changed to "DOS 3.x VOL".) So any application program using DOS 3+2 which changes DOS by POKEing the appropriate value into the RWTS vector at \$BCE9, should change this byte (\$B3B4), too.

Another patch disables the INIT command under 13-sector DOS (gives an I/O ERROR). Other patches tell DOS that it has grown in size, either for saving itself to disk during 16-sector INIT, or so it knows where to start building file buffers without writing over the new sections that have been appended to it.

ENTERING THE PROGRAM

DOS 3+2 can be entered directly into your Apple by first, entering the monitor (type CALL-151 <RETURN>). Then begin entering the Hex Code at memory location \$3000. The initial entries are: 3000: 4C 3C 32 00 19 8F 1B etc. If you are unfamiliar with how to enter Hex Code, consult your Apple II Reference Manual pp 43-44. After entry, it would be a good idea to check your list against the listing by using the disassembler. Since this program makes several patches to DOS, even a single byte error could have disastrous results. When you're sure it's correct, you can save it with BSAVE DOS 3+2, A\$3000, L\$351.

Ready to go? Start the program with 3000G from the monitor or CALL 12288 from BASIC. Insert a disk containing "MUFFIN" when you're told to, and two seconds later it's all over. (It would be wise to only use disks that you can afford to lose until you've verified that DOS 3+2 is working properly.)

Usage Notes

Pardon me for patting myself on the back, but I must say that DOS 3+2 works like a dream! (Almost any application program (like "FILE CABINET") can now read or write data on 13 and 16-sector formatted disks interchangeably, without any program modification! (It's best to use control-A and control-B to switch between the two DOS's. You could leave the automatic error-driven switch do it for you, but this means listening to the unsettling grinding sound that comes with any disk read error.)

You can now use "FID" to transfer files between the two DOS versions. Unlike "MUFFIN", you can see the disk catalog or check the available disk memory space before transferring any program. Also, you can transfer from 16 to 13-sector instead of just the other way. And while "FID" still requires the same effort (one file at a time) as "MUFFIN" to transfer files, with DOS 3+2 the need to make the transfer is in many cases eliminated.

One of the nicest features about DOS 3+2 is that the program only needs to be used once. You can INIT disks under DOS 3+2 and when

```
; DOS 3+2
; BY JOEL BUCKLEY
;
; COPYRIGHT (1982)
; MICRO-SPARC, INC.
```

```
; Program relocates DOS 3.2.1 RWTS routines from MUFFIN and
; appends them to DOS 3.3. A keyboard intercept routine
; controls switching between DOS 3.2.1 and DOS 3.3.
; Additionally, a disk read error will automatically cause
; the other DOS to be tried. Other changes enable slave
; diskettes to be INIT'ed in 16 sector format with entire
; DOS3+2. (DOS 3.2.1 INIT command is disabled.)
```

```
; Code Segments:
; 3003-3011 Misc Data
; 3012-31BA String Data
; 31BB-31C4 RWTS Vector - moved to $BCE0-$BCE9
; 31C5-31E5 Keyboard Intercept - moved to $9CA0-$9CC0
; 31E6-3224 Error-driven switch - moved to $9CC1-$9CFE
; 3225-323B Print Subroutine
; 323C-3350 Main Routine
```

```
.ABSOLUTE
.PROC DDOS, 0
.ORG 3000
```

```
; Misc equates.
SWITCH .EQU 26 ; 9B XOR BD
DISINDX .EQU 2F
ZF1 .EQU 40
ZF2 .EQU 42
DOSCOLD .EQU 3D3
ENTER2 .EQU 9B04 ; DOS 3.2.1 RWTS reentry point
ERROR2 .EQU 9C39 ; DOS 3.2.1 disk read error exit
DOS3 .EQU 9CA0 ; relocated address
ENTRY .EQU 9CB1 ; " "
ERRSW .EQU 9CC1 ; " "
ERRFLG .EQU 9CDB ; " "
FLIP .EQU 9CE1 ; " "
DOSNUM .EQU 0B3B4 ; * in CATSTR
```

```
DOSVER .EQU 0B3BE
RWTSVEC .EQU 0BCE0 ; relocated address
DOSSW .EQU 0BCE9 ; RWTS vector hi-byte
ENTER3 .EQU 0BD04 ; DOS 3.3 RWTS reentry point
ERROR3 .EQU 0BE4B ; DOS 3.3 disk read error exit
DISASS .EQU 0F8BE
HOME .EQU 0FC5B
GETLN .EQU 0FD6F
COUT .EQU 0FDED
BELL .EQU 0FF3A
3000: 4C 3C 32 JMP START
```

```
; Misc Data
3003: 00 19 PROGBEG .WORD 1900 ; code
3005: 8F 1B PROGEND .WORD 1BBF ; code
3007: 00 1E .WORD 1E00 ; data
3009: FF 1F .WORD 1FFF ; code
```

```
300B: 00 TEMP .BYTE
PROGNO .EQU XTEMP-PROGBEG
; No. of sections of code to be relocated.
300C: 1F SRCTOP .BYTE 1F ; Top page of source.
300D: 9C DESTOP .BYTE 9C ; Top page of destination.
300E: 00 OFFSET .BYTE ; DESTOP-SRCTOP goes here.
300F: 07 COUNT .BYTE 7 ; No. of pages to be moved.
3010: 19 RNLGLO .BYTE 19 ; Relocation range low.
3011: 20 RNLGHI .BYTE 20 ; Relocation range hi.
```

```
; String Data
STR1 .EQU 3012
3012: 44 4F 53 20 33 20 2B .ASCII "DOS 3 + 2 - BY JOEL BUCKLEY"
3019: 20 32 20 20 2D 20 20 20
3020: 42 59 20 4A 4F 45 4C
3027: 20 42 55 43 4B 4C 45
302E: 59
302F: 0D .BYTE 0D
3030: 2D 2D 2D 2D 2D 2D 2D .ASCII "-----"
3037: 2D 2D 2D 2D 2D 2D 2D
```


booted they'll still contain all of DOS 3+2. The technique used for expanding DOS can also be applied to other frequently used routines that you would like to have permanently attached to DOS (like a lower-case driver).

LIMITATIONS

DOS 3+2 does place a few limitations on programs being used with it. Other programs that patch into the same unused parts of DOS that DOS 3+2 uses obviously won't work. And some programs that directly access parts of DOS may not perform as expected. But there shouldn't be any problems if only the standard page 3 jump vectors are used to access DOS.

It is now impossible to enter a control-A or control-B into a program or string while DOS 3+2 is in effect. While this is unlikely to be much of a problem, if it is, there are two approaches to dealing with it. If the applica-

tion permits, you can temporarily disconnect DOS (with **TEXT : PR #0 : IN #0**), and reconnect it later with a call 1002. Or you can change the control codes that select which DOS is in effect (at **\$31DA** and **\$31E2** before DOS 3+2 is run, or **\$9CB5** and **\$9CBD** after it's been run).

RENUMBER FIX

Other programs that won't work without modification are those that expect the DOS starting address to be at **\$9D00** (or the DOS file buffer bottom to be at **\$9600**). One such program is "RENUMBER", which has its own slightly deficient code relocation routine. Since it's a very commonly used program, I will present a fix for its use with DOS 3+2 here.

Please note that **this fix will work only with "RENUMBER" from the DOS 3.3 System Master Disk. "RENUMBER" from the DOS 3.2.1 Master Disk is NOT the same.**

LOAD RENUMBER. Enter the monitor with

CALL -151. Type **B62: 4C D0 0B <return> BD0: C9 D6 D0 03 4C 70 0B 18 69 10 4C 65 0B <return> 1085: D6 <return>**. (This patch was provided by C. Bongers in the July/August 1981 Call -A.P.P.L.E.) And while you're at it, you might as well fix another "RENUMBER" bug, one that prevents line numbers after LIST from being renumbered, and instead treats a number after an asterisk* (multiplication symbol) as if it was a line number. Type **12B5:AC AB**. (This patch from Alan and Valerie Floeter in the October 1981 Call -A.P.P.L.E.) Now reenter BASIC and SAVE RENUMBER before RUNNING it. The patched "RENUMBER" will still work properly under regular DOS.

SUMMARY

DOS 3+2 can be a very effective remedy in attempting to deal with the two DOS blues.

```

303E: 2D 2D 2D 2D 2D 2D 2D
3045: 2D 2D 2D 2D 2D 2D 2D
304C: 2D
304D: 0D
304E: 28 43 4F 50 59 52 49
3055: 47 48 54 20 31 39 38
305C: 31 20 42 59 20 4A 4F
3063: 45 4C 20 42 55 43 4B
306A: 4C 45 59 29
306E: 0D 0D
3070: 20 20 20 54 48 49 53
3077: 20 50 52 4F 47 52 41
307E: 4D 20 41 50 50 45 4E
3085: 44 53 20 44 4F 53 20
308C: 33 2E 32 2E 31
3091: 0D 0D
3093: 44 49 53 4B 20 49 2F
309A: 4F 20 52 4F 55 54 49
30A1: 4E 45 53 20 54 4F 20
30AB: 34 38 4B 20 44 4F 53
30AF: 20 33 2E 33 2E
30B4: 0D 0D
30B6: 20 20 20 49 4E 53 45
30BD: 52 54 20 44 49 53 4B
30C4: 20 43 4F 4E 54 41 49
30CB: 4E 49 4E 47 20 27 4D
30D2: 55 46 46 49 4E 27
30D8: 0D 0D
30DA: 41 4E 44 20 50 52 45
30E1: 53 53 20 3C 52 45 54
30E8: 55 52 4E 3E 2E
30ED: 00
30EE: 0D 04
30F0: 42 4C 4F 41 44 20 4D
30F7: 55 46 46 49 4E
30FC: 0D
30FD: 0D
30FE: 44 4F 53 20 33 20 2B
3105: 20 32 20 49 53 20 49
310C: 4E 53 54 41 4C 4C 45
3113: 44 2E
3115: 07 07 0D 0D
3119: 20 20 20 43 54 52 4C
3120: 2D 41 20 53 45 4C 45
3127: 43 54 53 20 44 4F 53
312E: 20 33 2E 32 2E 31 2E
3135: 0D 0D
3137: 20 20 20 43 54 52 4C
313E: 20 42 20 53 45 4C 45
3145: 43 54 53 20 44 4F 53
314C: 20 33 2E 33 2E
3151: 0D 0D
3153: 28 41 20 44 49 53 4B
315A: 20 52 45 41 44 20 45
3161: 52 52 4F 52 20 43 41
3168: 55 53 45 53 20 4F 54
316F: 48 45 52
3172: 0D 0D
3174: 44 4F 53 20 54 4F 20
317B: 42 45 20 54 52 49 45
3182: 44 20 41 55 54 4F 4D
3189: 41 54 49 43 41 4C 4C
3190: 59 2E 29
3193: 00
3194: 34 38 4B 20 44 4F 53
319B: 20 33 2E 33 20 4E 4F
31A2: 54 20 46 4F 55 4E 44
31A9: 2E
31AA: 07 07 0D 0D 00
31AF: A0 CC CF D6 A0 B3
31B5: AE B3 A0 D3 CF C4
31BB: 48
31BC: A9 00
31BE: BD DB 9C
31C1: 68
31C2: 4C 00 BD
31C5: A9 BD
31C7: 20 E6 9C
31CA: A4 24
31CC: B1 2B
31CE: 48
31CF: 29 3F
31D1: 09 40
31D3: 91 2B
31D5: 68
31D6: 20 BA 9E
31D9: C9 81
31DB: D0 04
31DD: A9 9B
31DF: D0 E6
31E1: C9 82
31E3: F0 E0
31E5: 60
31E6: 2C DB 9C
31E9: 70 0E
31EB: BD DB 9C
31EE: 20 E1 9C
31F1: B0 03
31F3: 4C 04 9B
31F6: 4C 04 BD
31F9: 20 E1 9C
31FC: A9 00
31FE: B0 03
3200: 4C 39 9C
3203: 4C 4B BE
3206: AD E9 BC
3209: 49 26
320B: BD E9 BC
320E: 6A
320F: 29 01
3211: 49 B3
3213: BD B4 B3
3216: 6A
3217: 08
3218: 90 03
321A: 20 3A FF
321D: 20 3A FF
3220: 20 3A FF
3223: 2B
3224: 60
3226: AD E9 BC
3229: 49 26
322B: BD E9 BC
322E: 6A
322F: 29 01
3231: 49 B3
3233: BD B4 B3
3236: 6A
3237: 08
3238: 90 03
323A: 20 3A FF
323D: 20 3A FF
3240: 20 3A FF
3243: 2B
3244: 60
3246: AD E9 BC
3249: 49 26
324B: BD E9 BC
324E: 6A
324F: 29 01
3251: 49 B3
3253: BD B4 B3
3256: 6A
3257: 08
3258: 90 03
325A: 20 3A FF
325D: 20 3A FF
3260: 20 3A FF
3263: 2B
3264: 60
3266: AD E9 BC
3269: 49 26
326B: BD E9 BC
326E: 6A
326F: 29 01
3271: 49 B3
3273: BD B4 B3
3276: 6A
3277: 08
3278: 90 03
327A: 20 3A FF
327D: 20 3A FF
3280: 20 3A FF
3283: 2B
3284: 60
3286: AD E9 BC
3289: 49 26
328B: BD E9 BC
328E: 6A
328F: 29 01
3291: 49 B3
3293: BD B4 B3
3296: 6A
3297: 08
3298: 90 03
329A: 20 3A FF
329D: 20 3A FF
3300: 20 3A FF
3303: 2B
3304: 60
3306: AD E9 BC
3309: 49 26
330B: BD E9 BC
330E: 6A
330F: 29 01
3311: 49 B3
3313: BD B4 B3
3316: 6A
3317: 08
3318: 90 03
331A: 20 3A FF
331D: 20 3A FF
3320: 20 3A FF
3323: 2B
3324: 60
3326: AD E9 BC
3329: 49 26
332B: BD E9 BC
332E: 6A
332F: 29 01
3331: 49 B3
3333: BD B4 B3
3336: 6A
3337: 08
3338: 90 03
333A: 20 3A FF
333D: 20 3A FF
3340: 20 3A FF
3343: 2B
3344: 60
3346: AD E9 BC
3349: 49 26
334B: BD E9 BC
334E: 6A
334F: 29 01
3351: 49 B3
3353: BD B4 B3
3356: 6A
3357: 08
3358: 90 03
335A: 20 3A FF
335D: 20 3A FF
3360: 20 3A FF
3363: 2B
3364: 60
3366: AD E9 BC
3369: 49 26
336B: BD E9 BC
336E: 6A
336F: 29 01
3371: 49 B3
3373: BD B4 B3
3376: 6A
3377: 08
3378: 90 03
337A: 20 3A FF
337D: 20 3A FF
3380: 20 3A FF
3383: 2B
3384: 60
3386: AD E9 BC
3389: 49 26
338B: BD E9 BC
338E: 6A
338F: 29 01
3391: 49 B3
3393: BD B4 B3
3396: 6A
3397: 08
3398: 90 03
339A: 20 3A FF
339D: 20 3A FF
3400: 20 3A FF
3403: 2B
3404: 60
3406: AD E9 BC
3409: 49 26
340B: BD E9 BC
340E: 6A
340F: 29 01
3411: 49 B3
3413: BD B4 B3
3416: 6A
3417: 08
3418: 90 03
341A: 20 3A FF
341D: 20 3A FF
3420: 20 3A FF
3423: 2B
3424: 60
3426: AD E9 BC
3429: 49 26
342B: BD E9 BC
342E: 6A
342F: 29 01
3431: 49 B3
3433: BD B4 B3
3436: 6A
3437: 08
3438: 90 03
343A: 20 3A FF
343D: 20 3A FF
3440: 20 3A FF
3443: 2B
3444: 60
3446: AD E9 BC
3449: 49 26
344B: BD E9 BC
344E: 6A
344F: 29 01
3451: 49 B3
3453: BD B4 B3
3456: 6A
3457: 08
3458: 90 03
345A: 20 3A FF
345D: 20 3A FF
3460: 20 3A FF
3463: 2B
3464: 60
3466: AD E9 BC
3469: 49 26
346B: BD E9 BC
346E: 6A
346F: 29 01
3471: 49 B3
3473: BD B4 B3
3476: 6A
3477: 08
3478: 90 03
347A: 20 3A FF
347D: 20 3A FF
3480: 20 3A FF
3483: 2B
3484: 60
3486: AD E9 BC
3489: 49 26
348B: BD E9 BC
348E: 6A
348F: 29 01
3491: 49 B3
3493: BD B4 B3
3496: 6A
3497: 08
3498: 90 03
349A: 20 3A FF
349D: 20 3A FF
3500: 20 3A FF
3503: 2B
3504: 60
3506: AD E9 BC
3509: 49 26
350B: BD E9 BC
350E: 6A
350F: 29 01
3511: 49 B3
3513: BD B4 B3
3516: 6A
3517: 08
3518: 90 03
351A: 20 3A FF
351D: 20 3A FF
3520: 20 3A FF
3523: 2B
3524: 60
3526: AD E9 BC
3529: 49 26
352B: BD E9 BC
352E: 6A
352F: 29 01
3531: 49 B3
3533: BD B4 B3
3536: 6A
3537: 08
3538: 90 03
353A: 20 3A FF
353D: 20 3A FF
3540: 20 3A FF
3543: 2B
3544: 60
3546: AD E9 BC
3549: 49 26
354B: BD E9 BC
354E: 6A
354F: 29 01
3551: 49 B3
3553: BD B4 B3
3556: 6A
3557: 08
3558: 90 03
355A: 20 3A FF
355D: 20 3A FF
3560: 20 3A FF
3563: 2B
3564: 60
3566: AD E9 BC
3569: 49 26
356B: BD E9 BC
356E: 6A
356F: 29 01
3571: 49 B3
3573: BD B4 B3
3576: 6A
3577: 08
3578: 90 03
357A: 20 3A FF
357D: 20 3A FF
3580: 20 3A FF
3583: 2B
3584: 60
3586: AD E9 BC
3589: 49 26
358B: BD E9 BC
358E: 6A
358F: 29 01
3591: 49 B3
3593: BD B4 B3
3596: 6A
3597: 08
3598: 90 03
359A: 20 3A FF
359D: 20 3A FF
3600: 20 3A FF
3603: 2B
3604: 60
3606: AD E9 BC
3609: 49 26
360B: BD E9 BC
360E: 6A
360F: 29 01
3611: 49 B3
3613: BD B4 B3
3616: 6A
3617: 08
3618: 90 03
361A: 20 3A FF
361D: 20 3A FF
3620: 20 3A FF
3623: 2B
3624: 60
3626: AD E9 BC
3629: 49 26
362B: BD E9 BC
362E: 6A
362F: 29 01
3631: 49 B3
3633: BD B4 B3
3636: 6A
3637: 08
3638: 90 03
363A: 20 3A FF
363D: 20 3A FF
3640: 20 3A FF
3643: 2B
3644: 60
3646: AD E9 BC
3649: 49 26
364B: BD E9 BC
364E: 6A
364F: 29 01
3651: 49 B3
3653: BD B4 B3
3656: 6A
3657: 08
3658: 90 03
365A: 20 3A FF
365D: 20 3A FF
3660: 20 3A FF
3663: 2B
3664: 60
3666: AD E9 BC
3669: 49 26
366B: BD E9 BC
366E: 6A
366F: 29 01
3671: 49 B3
3673: BD B4 B3
3676: 6A
3677: 08
3678: 90 03
367A: 20 3A FF
367D: 20 3A FF
3680: 20 3A FF
3683: 2B
3684: 60
3686: AD E9 BC
3689: 49 26
368B: BD E9 BC
368E: 6A
368F: 29 01
3691: 49 B3
3693: BD B4 B3
3696: 6A
3697: 08
3698: 90 03
369A: 20 3A FF
369D: 20 3A FF
3700: 20 3A FF
3703: 2B
3704: 60
3706: AD E9 BC
3709: 49 26
370B: BD E9 BC
370E: 6A
370F: 29 01
3711: 49 B3
3713: BD B4 B3
3716: 6A
3717: 08
3718: 90 03
371A: 20 3A FF
371D: 20 3A FF
3720: 20 3A FF
3723: 2B
3724: 60
3726: AD E9 BC
3729: 49 26
372B: BD E9 BC
372E: 6A
372F: 29 01
3731: 49 B3
3733: BD B4 B3
3736: 6A
3737: 08
3738: 90 03
373A: 20 3A FF
373D: 20 3A FF
3740: 20 3A FF
3743: 2B
3744: 60
3746: AD E9 BC
3749: 49 26
374B: BD E9 BC
374E: 6A
374F: 29 01
3751: 49 B3
3753: BD B4 B3
3756: 6A
3757: 08
3758: 90 03
375A: 20 3A FF
375D: 20 3A FF
3760: 20 3A FF
3763: 2B
3764: 60
3766: AD E9 BC
3769: 49 26
376B: BD E9 BC
376E: 6A
376F: 29 01
3771: 49 B3
3773: BD B4 B3
3776: 6A
3777: 08
3778: 90 03
377A: 20 3A FF
377D: 20 3A FF
3780: 20 3A FF
3783: 2B
3784: 60
3786: AD E9 BC
3789: 49 26
378B: BD E9 BC
378E: 6A
378F: 29 01
3791: 49 B3
3793: BD B4 B3
3796: 6A
3797: 08
3798: 90 03
379A: 20 3A FF
379D: 20 3A FF
3800: 20 3A FF
3803: 2B
3804: 60
3806: AD E9 BC
3809: 49 26
380B: BD E9 BC
380E: 6A
380F: 29 01
3811: 49 B3
3813: BD B4 B3
3816: 6A
3817: 08
3818: 90 03
381A: 20 3A FF
381D: 20 3A FF
3820: 20 3A FF
3823: 2B
3824: 60
3826: AD E9 BC
3829: 49 26
382B: BD E9 BC
382E: 6A
382F: 29 01
3831: 49 B3
3833: BD B4 B3
3836: 6A
3837: 08
3838: 90 03
383A: 20 3A FF
383D: 20 3A FF
3840: 20 3A FF
3843: 2B
3844: 60
3846: AD E9 BC
3849: 49 26
384B: BD E9 BC
384E: 6A
384F: 29 01
3851: 49 B3
3853: BD B4 B3
3856: 6A
3857: 08
3858: 90 03
385A: 20 3A FF
385D: 20 3A FF
3860: 20 3A FF
3863: 2B
3864: 60
3866: AD E9 BC
3869: 49 26
386B: BD E9 BC
386E: 6A
386F: 29 01
3871: 49 B3
3873: BD B4 B3
3876: 6A
3877: 08
3878: 90 03
387A: 20 3A FF
387D: 20 3A FF
3880: 20 3A FF
3883: 2B
3884: 60
3886: AD E9 BC
3889: 49 26
388B: BD E9 BC
388E: 6A
388F: 29 01
3891: 49 B3
3893: BD B4 B3
3896: 6A
3897: 08
3898: 90 03
389A: 20 3A FF
389D: 20 3A FF
3900: 20 3A FF
3903: 2B
3904: 60
3906: AD E9 BC
3909: 49 26
390B: BD E9 BC
390E: 6A
390F: 29 01
3911: 49 B3
3913: BD B4 B3
3916: 6A
3917: 08
3918: 90 03
391A: 20 3A FF
391D: 20 3A FF
3920: 20 3A FF
3923: 2B
3924: 60
3926: AD E9 BC
3929: 49 26
392B: BD E9 BC
392E: 6A
392F: 29 01
3931: 49 B3
3933: BD B4 B3
3936: 6A
3937: 08
3938: 90 03
393A: 20 3A FF
393D: 20 3A FF
3940: 20 3A FF
3943: 2B
3944: 60
3946: AD E9 BC
3949: 49 26
394B: BD E9 BC
394E: 6A
394F: 29 01
3951: 49 B3
3953: BD B4 B3
3956: 6A
3957: 08
3958: 90 03
395A: 20 3A FF
395D: 20 3A FF
3960: 20 3A FF
3963: 2B
3964: 60
3966: AD E9 BC
3969: 49 26
396B: BD E9 BC
396E: 6A
396F: 29 01
3971: 49 B3
3973: BD B4 B3
3976: 6A
3977: 08
3978: 90 03
397A: 20 3A FF
397D: 20 3A FF
3980: 20 3A FF
3983: 2B
3984: 60
3986: AD E9 BC
3989: 49 26
398B: BD E9 BC
398E: 6A
398F: 29 01
3991: 49 B3
3993: BD B4 B3
3996: 6A
3997: 08
3998: 90 03
399A: 20 3A FF
399D: 20 3A FF
4000: 20 3A FF
4003: 2B
4004: 60
4006: AD E9 BC
4009: 49 26
400B: BD E9 BC
400E: 6A
400F: 29 01
4011: 49 B3
4013: BD B4 B3
4016: 6A
4017: 08
4018: 90 03
401A: 20 3A FF
401D: 20 3A FF
4020: 20 3A FF
4023: 2B
4024: 60
4026: AD E9 BC
4029: 49 26
402B: BD E9 BC
402E: 6A
402F: 29 01
4031: 49 B3
4033: BD B4 B3
4036: 6A
4037: 08
4038: 90 03
403A: 20 3A FF
403D: 20 3A FF
4040: 20 3A FF
4043: 2B
4044: 60
4046: AD E9 BC
4049: 49 26
404B: BD E9 BC
404E: 6A
404F: 29 01
4051: 49 B3
4053: BD B4 B3
4056: 6A
4057: 08
4058: 90 03
405A: 20 3A FF
405D: 20 3A FF
4060: 20 3A FF
4063: 2B
4064: 60
4066: AD E9 BC
4069: 49 26
406B: BD E9 BC
406E: 6A
406F: 29 01
4071: 49 B3
4073: BD B4 B3
4076: 6A
4077: 08
4078: 90 03
407A: 20 3A FF
407D: 20 3A FF
4080: 20 3A FF
4083: 2B
4084: 60
4086: AD E9 BC
4089: 49 26
408B: BD E9 BC
408E: 6A
408F: 29 01
4091: 49 B3
4093: BD B4 B3
4096: 6A
4097: 08
4098: 90 03
409A: 20 3A FF
409D: 20 3A FF
4100: 20 3A FF
4103: 2B
4104: 60
4106: AD E9 BC
4109: 49 26
410B: BD E9 BC
410E: 6A
410F: 29 01
4111: 49 B3
4113: BD B4 B3
4116: 6A
4117: 08
4118: 90 03
411A: 20 3A FF
411D: 20 3A FF
4120: 20 3A FF
4123: 2B
4124: 60
4126: AD E9 BC
4129: 49 26
412B: BD E9 BC
412E: 6A
412F: 29 01
4131: 49 B3
4133: BD B4 B3
4136: 6A
4137: 08
4138: 90 03
413A: 20 3A FF
413D: 20 3A FF
4140: 20 3A FF
4143: 2B
4144: 60
4146: AD E9 BC
4149: 49 26
414B: BD E9 BC
414E: 6A
414F: 29 01
4151: 49 B3
4153: BD B4 B3
4156: 6A
4157: 08
4158: 90 03
415A: 20 3A FF
415D: 20 3A FF
4160: 20 3A FF
4163: 2B
4164: 60
4166: AD E9 BC
4169: 49 26
416B: BD E9 BC
416E: 6A
416F: 29 01
4171: 49 B3
4173: BD B4 B3
4176: 6A
4177: 08
4178: 90 03
417A: 20 3A FF
417D: 20 3A FF
4180: 20 3A FF
4183: 2B
4184: 60
4186: AD E9 BC
4189: 49 26
418B: BD E9 BC
418E: 6A
418F: 29 01
4191: 49 B3
4193: BD B4 B3
4196: 6A
4197: 08
4198: 90 03
419A: 20 3A FF
419D: 20 3A FF
4200: 20 3A FF
4203: 2B
4204: 60
4206: AD E9 BC
4209: 49 26
420B: BD E9 BC
420E: 6A
420F: 29 01
4211: 49 B3
4213: BD B4 B3
4216: 6A
4217: 08
4218: 90 03
421A: 20 3A FF
421D: 20 3A FF
4220: 20 3A FF
4223: 2B
4224: 60
4226: AD E9 BC
4229: 49 26
422B: BD E9 BC
422E: 6A
422F: 29 01
4231: 49 B3
4233: BD B4 B3
4236: 6A
4237: 08
4238: 90 03
423A: 20 3A FF
423D: 20 3A FF
4240: 20 3A FF
4243: 2B
4244: 60
4246: AD E9 BC
4249: 49 26
424B: BD E9 BC
424E: 6A
424F: 29 01
4251: 49 B3
4253: BD B4 B3
4256: 6A
4257: 08
4258: 90 03
425A: 20 3A FF
425D: 20 3A FF
4260: 20 3A FF
4263: 2B
4264: 60
4266: AD E9 BC
4269: 49 26
426B: BD E9 BC
426E: 6A
426F: 29 01
4271: 49 B3
4273: BD B4 B3
4276: 6A
4277: 08
4278: 90 03
427A: 20 3A FF
427D: 20 3A FF
4280: 20 3A FF
4283: 2B
4284: 60
4286: AD E9 BC
4289: 49 26
428B: BD E9 BC
428E: 6A
428F: 29 01
4291: 49 B3
4293: BD B4 B3
4296: 6A
4297: 08
4298: 90 03
429A: 20 3A FF
429D: 20 3A FF
4300: 20 3A FF
4303: 2B
4304: 60
4306: AD E9 BC
4309: 49 26
430B: BD E9 BC
430E: 6A
430F: 29 01
4311: 49 B3
4313: BD B4 B3
4316: 6A
4317: 08
4318: 90 03
431A: 20 3A FF
431D: 20 3A FF
4320: 20 3A FF
4323: 2B
4324: 60
4326: AD E9 BC
4329: 49 26
432B: BD E9 BC
432E: 6A
432F: 29 01
4331: 49 B3
4333: BD B4 B3
4336: 6A
4337: 08
4338: 90 03
433A: 20 3A FF
433D: 20 3A FF
4340: 20 3A FF
4343: 2B
4344: 60
4346: AD E9 BC
4349: 49 26
434B: BD E9 BC
434E: 6A
434F: 29 01
```


Apple DOS 3 + 2 (Cont.)

```

SWEND
SWLEN .EQU SWEND-DOS3X
; Print Subroutine
; This routine prints a string of ASCII
; characters terminated with hex 00.
; Call with address lo-byte in A,
; address hi-byte in Y. Uses A, X, Y,
; and two zero page locations.

3225: B5 40
3227: B4 41
3229: A0 00
322B: F0 0A
322D: 09 80
322F: 20 ED FD
3232: C8
3233: D0 02
3235: E6 41
3237: B1 40
3239: D0 F2
323B: 60

PRINT STA ZP1
      STY ZP1+1
      LDY # 0
      BEQ PR3
PR2   ORA # 80
      JSR COUT
      INY
      BNE PR3
      INC ZP1+1
PR3   LDA @ ZP1, Y
      BNE PR2
      RTS

      .MACRO DISPLAY,1
      LDA # ?1*100
      LDY # ?1/100
      JSR PRINT
      .ENDM

; Main Routine
START LDA DOSVER
      CMP # 3 ; 49K DOS 3.3?
      BEQ YEP
      DISPLAY STR4
      LDA # STR4*100
      LDY # STR4/100
      JSR PRINT
      RTS

YEP   JSR HOME
      DISPLAY STR1
      LDA # STR1*100
      LDY # STR1/100
      JSR PRINT
      JSR GETLN
      DISPLAY STR2
      LDA # STR2*100
      LDY # STR2/100
      JSR PRINT
      DISPLAY STR3
      LDA # STR3*100
      LDY # STR3/100
      JSR PRINT

; Relocation routine adapted from
; DOS 3.2.1 Master relocation code.
; MUFFIN DOS 3.2.1 RWTS code
; will be relocated to $9600-$9CFF.
RELOC SEC
      LDA DESTOP
      SBC SRC TOP
      STA OFFSET
      LDA # 0
      TAX
      TAY
      STA ZP1
      STA ZP2

; Get starting address of code to
; be relocated.
PRGREL STX XTEMP
      LDA PRGBEG,X
      STA ZP1
      LDA PRGBEG+1,X
      STA ZP1+1

; Adjust any instructions in range
; to RNLGO to RNGLI by adding OFFSET.
INSDIS LDX # 0
      LDA @ ZP1, X
      JSR DISASS
      LDY DISINDX
      CPY # 2 ; 3-byte instruction
      BNE NXTINS
      LDA @ ZP1, Y
      CMP RNLGO
      BCC NXTINS
      CMP RNGLI
      BCS NXTINS
      ADC OFFSET
      STA @ ZP1, Y

; Adjust starting address for next
; disassembly. Check for out of range.
NXTINS SEC
      LDA DISINDX
      ADC ZP1
      STA ZP1
      LDA # 0
      ADC ZP1+1
      STA ZP1+1
      LDX XTEMP
      CMP PROGEND+1, X
      BCC INSDIS
      LDA ZP1
      CHF PROGEND, X
      BCC INSDIS

; Check for last code range.
      TAX
      CLC
      ADC # 4
      TAX
      CPX # PROGNO
      BCC PRGREL

```

```

32C8: AD 0C 30
32CB: B5 41
32CD: AC 0D 30
32D0: B4 43
32D2: A9 00
32D4: B5 40
32D6: B5 42
32D8: A8

32D9: B1 40
32DB: 91 42
32DD: C8
32DE: D0 F9

32E0: C6 41
32E2: C6 43
32E4: CE 0F 30
32E7: D0 F0

```

```

; Move COUNT memory pages from
; SRC TOP to DESTOP working down.
MOVE  LDA SRC TOP
      STA ZP1+1
      LDY DESTOP
      STY ZP2+1
      LDA # 0
      STA ZP1
      STA ZP2
      TAY

```

```

NXTADR LDA @ ZP1, Y
        STA @ ZP2, Y
        INY
        BNE NXTADR

        DEC ZP1+1
        DEC ZP2+1
        DEC COUNT
        BNE NXTADR

```

; Patch DOS and exit.

```

      .MACRO MOVE,3
; Moves up to 256 bytes.
      LDX # ?1
      LDA ?2-1, X
      STA ?3-1, X
      DEX
      BNE $!
      .ENDM

```

```

      .MACRO PATCH,2
; (-8000,+80 compensate for assembler
; bug with arguments > $8000.)
      LDA # ?1-8000*100
      LDY # ?1-8000/100+80
      STA ?2
      STY ?2+1
      .ENDM

```

```

; Move CATSTR to $B3AF.
MOVE CATLEN, CATSTR, 0B3AF
      LDX # CATLEN
      LDA CATSTR-1, X
      STA 0B3AF-1, X
      DEX
      BNE $!

```

```

; Move RWTS vector to $BCE0.
MOVE RWLEN, RWTSVEC, RWTSVEC
      LDX # RWLEN
      LDA RWTSVEC-8000*100
      STA RWTSVEC-1, X
      DEX
      BNE $!

```

```

; Patch in RWTS vector.
; (JSR $BCE0 replaces JSR $BD00.)
PATCH RWTSVEC, 0B7BB
      LDA # RWTSVEC-8000*100
      LDY # RWTSVEC-8000/100+80
      STA 0B7BB
      STY 0B7BB+1

```

```

; Move keyboard intercept and
; disk error switch routines to $9CA0.
MOVE SWLEN, DOS3X, DOS3
      LDX # SWLEN
      LDP DOS3X-1, X
      STY DOS3-1, X
      DEX
      BNE $!

```

```

; Patch in DOS keyboard intercept.
PATCH ENTRY, 9EAF
      LDA # ENTRY-8000*100
      LDY # ENTRY-8000/100+80
      STA 9EAF
      STY 9EAF+1

```

```

; Change disk read error vectors to $9C01.
PATCH ERRSW, 9BFC
      LDA # ERRSW-8000*100
      LDY # ERRSW-8000/100+80
      STA 9BFC
      STY 9BFC+1
      PATCH ERRSW, 0BE09
      LDA # ERRSW-8000*100
      LDY # ERRSW-8000/100+80
      STA 0BE09
      STY 0BE09+1

```

```

; Disable DOS 3.2.1 INIT command.
      LDA # 39
      STA 9C07

```

```

; Extend DOS for DOS 3.3 INIT.
      LDA # 96
      STA 9D0D ; DOS bottom
      LDA # 0B
      STA 0B71A ; last DOS sector
      STA 0B763 ; last DOS sector
      LDA # 22
      STA 0B7E0 ; # DOS pages - $A

```

```

; Extend DOS for HIMEM, MAXFILES.
      LDA # 95
      STA 9D01 ; file buffer ptr

```

```

; Cold start DOS (to adjust pointers).
JMP DOSCOLD

```

.END ; of assembly



Amper-Speed

AMPER-SPEED: Fast Read And Write For Text Files

by William Reynolds III
1733 N. Ford St.
McMinnville, OR 97128

While converting Alan Hill's Amper-Reader (Nibble Vol. 2 #1) for use with my Memory Management System 2.0, I was asked to produce a routine that would write to a text file as quickly as Amper-Reader reads a file. With the information that I had learned from the excellent book **Beneath Apple DOS**, I decided to rewrite Amper-Reader to include a **writing function**. The result is AMPER-SPEED. It is called by using nearly the same format as Amper-Reader. It uses about the same amount of memory for code, and uses the standard DOS buffers (not a dedicated buffer area).

Since DOS buffers are used, the calls to Read & Write may use different filenames, and the pointers for the files that are open will not be disturbed by the call for the specified file.

THE PROGRAM BUFFERS

The routine resides in .75K of memory, at \$9A00-9CFF. The problem with using that area is that it is part of the memory that DOS uses for its buffers. Any attempt to store a routine in an area that is being used for DOS I/O may be disastrous. For this reason, **the DOS buffers must be moved down in memory by .75K**. The routine **AMPERSPEED START** should be used. All that you need to do is include this line at the beginning of your program: **PRINT CHR\$(4); "BRUN AMPER-SPEED START"**.

NEW DOS BUFFERS

AMPERSPEED START will set up a new address for the part of memory that may be used for buffers, call the DOS routine that builds them, and set the & jump vector to point to AMPERSPEED. Then it will tell DOS to **BLOAD AMPERSPEED, A\$9A00**. At this point, the routine is installed in memory, the & vector is set, and HIMEM is set to the bottom of the last buffer.

WHAT IT DOES

Now that the routine is in place, how do you use it? The purpose of this routine is to **speed up disk Input/Output (I/O) when using sequential text files**. It will read or write using DOS, without going through the DOS command interpreter. The I/O will occur directly between the disk file and the Applesoft string variable storage area in memory. **The advantage is an increase in I/O speed of nearly 3 to 1!**

THE AMPERSAND

Applesoft has three ways to jump to a machine language routine: **CALL**, **USER(X)**, and the **&**. The first two can call a routine anywhere in the Apple. The third, the **&**, will force a jump to the address **\$3F5** whenever it is included in an Applesoft program. This is true for program-running mode, and direct keyboard mode as well. You can see that by setting the proper jump command at **\$3F5**, we can direct it to any location in memory.

Unless set by the programmer, **\$3F5** will cause a jump to an address in the monitor that merely tells it to **RETURN** to Applesoft. This keeps you from bombing your normal programs with the **&**.

THE SYNTAX

The AMPERSPEED command syntax is as follows: **&X(FIS, A\$(Y), NR%)**. The **'X'** is not a command letter, but rather, represents these commands: **R**ead, **C**ontinue to read, **W**rite, **A**ppend, and **D**elete variable. The items in the parentheses are parameters for the command.

In the example, **FIS** is the name of the string variable that contains the name of the file to be used, **A\$(Y)** is a dimensioned string variable which the string data will be input from or written to the disk. **Y** is the starting subscript to be used. **Y** may be predefined as a variable, or specified implicitly in the call to AMPER-SPEED. The variable **NR%** in the call must be an integer variable that tells the routine how many strings to read, or how many subscripts of **A\$** to write.

A CAUTION

One note of caution. Although you may use whatever names that you wish for your variables, they **MUST** be set **before** AMPERSPEED is called. Also, do not exceed the array size of **A\$** with **NR%** or **Y**, as this is not verified by the program, and the results will be unpredictable.

The single exception to the above is for the command **D**. To delete a dimensioned string array, the syntax is: **& D (A\$(0))** (NOTE—you must specify the 0 element). When this is executed, the array will no longer exist, and all the memory that was used by its elements will be made available.

FILE ACCESS

When the routine is called for file access, AMPERSPEED will locate a buffer for the file to use. If a buffer is already assigned to the file (that is, if the file is already Open), it will use that one. If a buffer is not assigned it will **OPEN** one, and that buffer will remain **OPEN** until **CLOSED** by a DOS command.

The heart of the routine is the use of internal DOS routines and the DOS File Manager. All parameters are set for the DOS File Manager, and it does the work of finding the file sectors.

In the case of a **Read**, a record length of \$100 is simulated. In this way, each time we run out of the data on a sector when looking for text strings, we have only to increment the record number to read, and DOS will get the next sector. The data from it is scanned for an **\$8D**. When found, the bytes between the previous **\$8D** and the one just found are stored in string memory area, and the length byte and address of the beginning of the string are stored in the variable area.

During the scan, the text characters are stored in the keyboard input buffer, which is not in use at that time. If AMPERSPEED is called directly from the keyboard, and not from a program, it will perform properly, but you will get a 'SYNTAX ERROR' message. This occurs because AMPERSPEED uses page zero pointers for access to the file manager parameter list, the buffer name section, the buffer data section, and the buffer work area. The file buffer's work area for the file

manager has 4 bytes that are not used by DOS. I use these to save the current sector number, and the pointer to the sector for scanning text.

THE COMMANDS

Now, an explanation of the commands: **R** will Open the file at the beginning, and then read the number of strings specified by **NR%** into array **A\$**, starting at subscript **Y**.

The **C** (continue) command will read in the same manner, but uses the ending point of the last **R** or **C** for its starting point.

The **W** (Write) will Open the file, making a new one if necessary, then write **NR%** strings from **A\$**, starting at **Y**, into the file.

The **A** (Append) will write to the file, but will use the pointers left by the last **W** or **A**.

In all four cases, **NR%** will contain the number of items to be read or written. Upon return, **NR%** will be set to the number of times that were actually read or written. In the case of an error, you will be able to tell how far you got when the error occurred.

ERRORS

Ah yes, about errors. AMPERSPEED is set up to send any errors that occur, to the Applesoft error handler. You may use the **ONERRGOTO** statement to check and process the errors. If you decide not to trap errors that occur when AMPERSPEED is running a fatal error will occur to the Applesoft program upon re-entry, and no error message will be printed. All errors use the standard DOS and Applesoft numbers when they are checked in the error handling routine.

THE POINTERS

The pointers must be set up by AMPERSPEED itself, so do NOT use **&A** without first using a **&W** or **&C** before using the **&R** command for EACH file to be accessed.

Since AMPERSPEED uses the DOS file buffers for I/O, if you do not set Maxfiles to a large enough number to handle the Open files, you will run out of them, and be unable to complete your task. **When you have finished with a file make certain that you remember to CLOSE it.**

Any file that has been Opened by AMPERSPEED may be **CLOSED** by the standard DOS syntax. If the file has been **OPENed** for a **WRITE**, it is very important to remember to **CLOSE** it. If it is not **CLOSED**, the last of the text (or data) will not be written from the buffer, and it will be lost. Also, the VTOC will not be updated to protect the sectors used by the file, and the directory sector count for the file entry will not be correct when **CATALOGed**.

HOW AMPERSPEED WORKS

When the **&** is encountered by Applesoft, it will **JUMP** to **\$3F5**. From there it is sent to **\$9A00**. At **\$9A00**, the next character in the line is checked to see if it is valid command for this routine. If it is, the address is retrieved, and the function is performed.

If no match is detected, a branch to **'NOT-NOW'** is performed. This will cause a **JUMP** to an RTS instruction, which will return to Applesoft. The **JUMP** instruction may be patched

continued on next page

AMPER-SPEED: Fast Read And Write For Text Files (Cont.)

by the user to interface with other routines. Also, in the address table and character table, there is room for six more functions to be detected. If the user desires to detect other commands and to direct control to other routines, the ASCII equivalent for the command letter should be stored in the first byte(s) of the character table, and the **address-1** of the routine(s) should be stored in the first byte pair(s) of the address table. The value of the X-register (at the beginning of the command detect sequence) must be increased for each new command which is added.

THE SYSTEM

As written, the routine will use drive 1, slot 6, and it will set volume match to 0. If desired, these parameters may be changed from your Applesoft program by using **POKE 39657, DRIVE:POKE 39662, SLOT:POKE 39715, VOLUME**. The assembly listing is sufficiently commented to help the reader understand the purpose of each of the functions and major subroutines. **AMPERSPEED** was written for DOS 3.3, and the Applesoft page 0 pointers are saved into an unused area in the DOS 3.3 RWTS.

If you use DOS 3.2, you must change the loops that store and restore page 0 to store to another area that will be protected. If you are not using page three, **\$300 — \$318** could be used. Also, many times the lower part of the stack (**\$100 — \$118**) can be safely used.

Since heavy use is made of routines that are imbedded in ROM Applesoft, the old Ram version will not work.

ENTERING AMPERSPEED

The AMPERSPEED START and AMPERSPEED programs can be entered into your Apple using a LISA 2.5 Assembly Language Program (using the Source Language listings).

Alternatively, they can be entered **directly into memory** using the Hex Machine Language from the listings. For more detail on enter Hex code directly into your Apple Memory, consult the Apple **Reference Manual** pp. 43-44.

Here are the steps for entering the programs using the Hex code!

1. Type **CALL-151** to enter the Monitor
2. Begin entering AMPERSPEED START at memory location **\$300** as follows:
***300:A9 9A 8D F7 03 A9 99 etc.**
3. When you have finished entering AMPERSPEED START save it to Disk with:
BSAVE AMPERSPEED START, A\$300, L\$40

4. Now begin entering AMPERSPEED at memory location **\$800** as follows:
***800:A2 05 CA 30 32 DD DF 9C etc.**
5. When you have finished entering AMPERSPEED, save it to Disk with:
BSAVE AMPERSPEED, A\$800, L\$2F4
6. The Programs are now stored on Disk.

Note that **both** AMPERSPEED START and AMPERSPEED must be on the same disk for correct operation. This is necessary because AMPERSPEED START is the **Loader** for AMPERSPEED and **relocates** AMPERSPEED to a beginning memory location of **\$9A00** upon loading it.

A FINAL NOTE

There you have it. Remember, AMPERSPEED is a fast writer/reader for sequential text files only. But it will produce a speed improvement of up to three times the normal Apple DOS.



AMPERSPEED START LISTING

```

;ASM
**END OF PASS 1

**END OF PASS 2

0000      1  ;AMPERSPEED START
0000      2  ;WILLIAM REYNOLDS III
0000      3  ;
0000      4  ;COPYRIGHT (C) 1982
0000      5  ;MICRO-SPARC, INC.
0000      6  ;ASSEMBLED WITH LISA 2.5
0000      7  ;
0000      8  ;
03F5      9  VECTAD      EQU $3F5
9A00     10  RT4BAD      EQU $9A00
9D01     11  BUFFAD      EQU $9D01
A7D4     12  BUILD       EQU $A7D4
FD00     13  PRINT       EQU $FD00
0000     14  ;
0000     15  ;
0300     16              ORG $300
0300     17              OBJ $B00
0300     18  ;
0300     19  ;
0300 A9 9A 20  START     LDA /RT4BAD
0302 BD F7 03 21         STA VECTAD+*02
0305 A9 99 22           LDA $*99
0307 BD 01 9D 23         STA BUFFAD
030A 20 D4 A7 24         JSR BUILD
030D A9 4C 25  LOAD      LDA $*4C
030F BD F5 03 26         STA VECTAD
0312 A9 00 27           LDA $*00
0314 BD F6 03 28         STA VECTAD+*01
0317 A0 00 29           LDY $*00
0319 B9 25 03 30  LOOP   LDA MESS,Y
031C F0 06 31           BEQ END
031E 20 ED FD 32         JSR PRINT
0321 C8 33             INY
0322 D0 F5 34           BNE LOOP
0324 60 35  END          RTS
0325 BD 84 36  MESS      HEX BD84
0327 C2 CC CF 37         ASC "BLOAD AMPERSPEED, A$9A00"
032A C1 C4 A0
032D C1 CD D0
0330 C5 D2 D3
0333 D0 C5 C5
0336 C4 AC C1
0339 A4 B9 C1
033C B0 B0
033E BD 00 38           HEX BD00
0340 39  END

***** END OF ASSEMBLY

```

AMPERSPEED HEX DUMP

```

$B00.AFB
0000- A2 05 CA 30 32 DD E3 9C
0008- D0 F8 BA 0A 48 A2 00 B5
0010- D0 9D 69 BA E8 E0 18 D0
0018- F6 68 AA 20 B1 00 F0 1A
0020- C9 28 D0 F7 20 B1 00 BD
0028- EF 9C 48 BD EE 9C 48 20
0030- DC 03 B4 E2 B5 E3 60 4C
0038- 36 9A A2 10 BA 48 20 46
0040- 9A 68 AA 4C 65 D8 A0 01
0048- A5 DA 91 D4 88 A5 D8 91
0050- D4 A2 00 BD 69 BA 95 D0
0058- E8 E0 18 D0 F6 60 20 DC
0060- 03 20 D6 03 90 07 A0 0A
0068- B1 E2 AA D0 CF 60 A2 00
0070- B6 E4 20 E3 DF A6 E4 95
0078- D0 94 D1 E8 E8 E0 06 F0
0080- 06 20 BE DE 4C 70 9A 20
0088- B1 00 A9 00 A0 01 B1 D4
0090- 85 DC B8 B1 D4 85 DD D0
0098- 08 A5 DC D0 04 C6 DC C6
00A0- DD B4 D9 84 DA B4 DB B1
00A8- D0 85 D8 C8 B1 D0 48 C8
00B0- B1 D0 85 D1 68 85 D0 60
00B8- 20 95 A0 A4 D8 88 30 09
00C0- B1 D0 09 80 99 75 AA D0
00C8- F4 20 64 A7 90 0F A5 45
00D0- D0 05 A2 0C 4C 3C 9A B5
00D8- 41 A5 44 85 40 20 43 A7
00E0- 20 4E A7 20 2C A7 A0 05
00E8- A9 01 91 E2 C8 A9 06 91
00F0- E2 A9 00 C8 91 E2 A0 10
00FB- B1 E2 B5 D6 C8 B1 E2 B5
00FD- D7 A0 C0 B1 E2 B5 DE C8
00FE- B1 E2 B5 DF 60 48 20 B8
0100- 9A A0 00 A9 01 91 E2 98
0108- C8 C8 91 E2 C8 68 AA 91
0110- E2 C8 A9 00 91 E2 4C 5E
0118- 9A 20 6E 9A A9 01 20 D0
0120- 2B B1 DE 48 B8 B1 DE A0
0128- 85 E4 91 DE C8 91 DE A0
0130- 2B B1 DE 48 B8 B1 DE A0
0138- A0 00 A9 03 91 E2 C8 91
0140- E2 68 C8 91 E2 C8 68 91
0148- E2 A9 00 C8 91 E2 C8 91
0150- E2 20 5E 9A A0 00 A6 E4
0158- 84 E5 B1 D6 F0 6F C9 BD
0160- F0 20 E6 D9 29 7F 9D 00
0170- 02 E8 C8 D0 EB A0 2A B1
0178- DE 18 69 01 91 DE C8 B1
0180- DE 69 00 91 DE B6 E4 4C
0188- 3F 9B B4 E5 98 A0 2C 91
0190- DE A6 D9 CA A0 00 A5 6F
0198- D0 02 C6 70 C6 6F BD 00
01A0- 02 91 6F CA E0 FF D0 EE
01A8- A0 00 A5 D9 91 D2 A5 6F
01B0- C8 91 D2 A5 70 C8 91 D2
01B8- 20 B5 9C A5 6E 69 01 C5
01C0- 70 90 05 A2 2D 4C 3C 9A
01C8- A2 00 B6 D9 B6 E4 A4 E5
01D0- C8 F0 A2 D0 B8 A2 05 D0
01D8- EC 20 6E 9A 20 B8 9A A0
01E0- 2C B1 DE A8 84 E5 4C D0
01E8- 9B 20 E3 DF 38 A5 83 E9
01F0- 07 B5 DE A5 84 E9 00 B5
01F8- DF A0 02 B1 DE 85 E0 C8
0200- B1 DE 85 E1 18 A5 E0 65
0208- DE 85 00 A5 E1 65 DF 85
0210- 41 A0 00 B1 40 91 DE E6
0218- DE D0 02 E6 DF A5 40 C5
0220- 6D A5 41 E5 6E E6 40 D0
0228- 02 E6 41 90 E6 38 A5 6D
0230- E5 E0 48 A5 6E E5 E1 B5
0238- 6E 68 85 6D 20 B1 00 D0
0240- A8 4C 51 9A 20 6E 9A A9
0248- 00 20 D0 9B A0 00 A9 0A
0250- 91 E2 A9 00 A0 02 91 E2
0258- C8 C0 06 D0 F9 20 5E 9A
0260- A0 00 A9 04 91 E2 4A C8
0268- 91 E2 A0 00 B1 D2 A0 06
0270- 91 E2 AA A9 00 C8 91 E2
0278- A9 02 A0 09 91 E2 88 A9
0280- 00 91 E2 A0 01 B1 D2 B5
0288- DE C8 B1 D2 B5 DF A0 00
0290- CA E0 FF D0 0A B1 DE 09
0298- 80 99 00 02 C8 D0 F1 A9
02A0- 8D 99 00 02 20 5E 9A 20
02A8- B5 9C 4C 72 9C 18 A5 D2
02B0- 69 03 B5 D2 A5 D3 69 00
02B8- 85 D3 E6 DA D0 02 E6 D8
02C0- A5 D8 C5 DD 90 08 A5 DA
02C8- C5 DC 90 08 20 46 9A 68
02D0- 68 60 20 6E 9A 20 B8 9A
02D8- 4C 68 9C 52 43 44 57 41
02E0- 09 A5 2B 18 69 07 28 9B
02E8- E0 9B F0 9B 4B 9C D9 9C

```



```

1 ;AMPERSPEED
2 ;WILLIAM REYNOLDS III
3 ;COPYRIGHT (C) 1982
4 ;MICROSPARC, INC.
5 ;ASSEMBLED WITH LISA
6 ;48K LEVEL VERSION
7 ;
8 ;
9 A1 EPZ $40
10 A2 EPZ $44
11 ENDSTP EPZ $6D
12 FRETOP EPZ $6F
13 LVAR EPZ $B3
14 CHRGET EPZ $B1
15 FNAME EPZ $D0
16 NAL EPZ $D2
17 NR EPZ $D4
18 BUFF1 EPZ $D6
19 LENFN EPZ $D8
20 NCHAR EPZ $D9
21 NREAD EPZ $DA
22 VNR EPZ $DC
23 TEMP1 EPZ $DE
24 OFFSET EPZ $E0
25 PARMAD EPZ $E2
26 XSAVE EPZ $E4
27 YSAVE EPZ $E5
28 BUFF2 EQU $200
29 FILMAN EQU $3D6
30 GTPMAD EQU $3DC
31 NONAME EQU $A095
32 NAMADD EQU $A72C
33 MOVNAM EQU $A743
34 DOSPAR EQU $A74E
35 GETBUF EQU $A764
36 NMBUFF EQU $AA75
37 HOLD EQU $BA69
38 BASERR EQU $DB65
39 GETNXT EQU $DEBE
40 GETADD EQU $DFE3
41 ;
42 ;
43 ORG $9A00
44 OBJ $B00
45 ;
46 ;
47 ;& IS VECTORED HERE
48 ;
49 ;CHECK FOR COMMAND LETTER
50 ;IF NOT A MATCH, EXIT WITH
51 ;ASCII OF COMMAND IN A-REG
52 ;
53 ;IF FOUND, CHECK FOR ' ' TO
54 ;BE START OF PASSED VALUE
55 ;DEFINITIONS, IF NOT FOUND
56 ;EXIT VIA BASIC 'SYNTAX ERROR'
57 ;
58 ;IF FOUND, SET ADDRESS TO
59 ;RETURN TO, ADDRESS TO GO TO,
60 ;AND ADDRESS OF DOS FILE MANAGER
61 ;PARAMETER LIST
62 ;
63 ;
64 START LDX #$05
65 LOOP DEX
66 BMI NOTNOW
67 CMP CHRTBL,X
68 BNE LOOP
69 TXA
70 ASL
71 PHA
72 LDX #$00
73 SVMORE LDA FNAME,X
74 STA HOLD,X
75 INX
76 CPX #$18
77 BNE SVMORE
78 PLA
79 TAX
80 LOOP01 JSR CHRGET
81 BEQ ERRSYN
82 CMP #' '
83 BNE LOOP01
84 JSR CHRGET
85 LDA ADDTBL+$01,X
86 PHA
87 LDA ADDTBL,X
88 PHA
89 JSR GTPMAD
90 STY PARMAD
91 STA PARMAD+$01
92 BYE RTS
93 ;
94 ;
95 ;EXIT HERE IS FOR COMMAND NOT FOR
96 ;THIS ROUTINE. MAY BE PATCHED
97 ;BY USER FOR OTHER ROUTINE CALLS
98 ;
99 ;
100 NOTNOW JMP BYE
101 ;
102 ;
103 ;EXIT WITH ERROR
104 ;
105 ;
106 ERRSYN LDX #$10
107 ERROR TXA
108 PHA
109 JSR EXIT
110 PLA
111 TAX
112 JMP BASERR
113 ;
114 ;
115 ;SAVE NUMBER OF ITEMS HANDLED
116 ;RESTORE ZERO PAGE DATA
117 ;
118 ;
119 EXIT LDX #$01
120 LDA NREAD
121 STA (NR),Y

```

```

122 DEV
123 LDA NREAD+$01
124 STA (NR),Y
125 EXIT1 LDX #$00
126 RDMRZP LDA HOLD,X
127 STA FNAME,X
128 INX
129 CPX #$18
130 BNE RDMRZP
131 RTS
132 ;
133 ;
134 ;GO TO FILE MANAGER
135 ;ERROR TRAP IF CARRY SET
136 ;
137 ;
138 GOFLMN JSR GTPMAD
139 JSR FILMAN
140 BCC OKOUT
141 LDY #$0A
142 LDA (PARMAD),Y
143 TAX
144 BNE ERROR
145 OKOUT RTS
146 ;
147 ;
148 ;GET PASSED PARMS NOW
149 ;FIRST SECTION LOOPS 3 TIMES
150 ;GETTING ADDRESS OF FILENAME
151 ;MATRIX STRING VARIABLE TO BE
152 ;USED, AND NUMBER OF ITEMS
153 ;
154 ;REST OF ROUTINE INITIALIZES
155 ;VARIABLES FOR COUNT, AND
156 ;SETS FILENAME LENGTH AND
157 ;ADDRESS OF THE NAME ITSELF
158 ;
159 BASPRM LDX #$00
160 LOOP1 STX XSAVE
161 JSR GETADD
162 LDX XSAVE
163 STA FNAME,X
164 STY FNAME+$01,X
165 INX
166 INX
167 CPX #$06
168 BEQ CONT
169 JSR GETNXT
170 JMP LOOP1
171 CONT JSR CHRGET
172 LDA #$00
173 LDY #$01
174 LDA (NR),Y
175 STA VNR
176 DEV
177 LDA (NR),Y
178 STA VNR+$01
179 BNE CONT1
180 LDA VNR
181 BNE CONT1
182 DEC VNR
183 DEC VNR+$01
184 CONT1 STY NCHAR
185 STY NREAD
186 STY NREAD+$01
187 LDA (FNAME),Y
188 STA LENFN
189 INY
190 LDA (FNAME),Y
191 PHA
192 INY
193 LDA (FNAME),Y
194 STA FNAME+1
195 PLA
196 STA FNAME
197 RTS
198 ;
199 ;
200 ;GET A THE DOS BUFFER IN USE
201 ;OR A NEW ONE IF NOT IN USE
202 ;
203 ;FIRST BLANK THE NAME BUFFER TO
204 ;ALL SPACES, THEN COPY NAME
205 ;FROM MEMORY INTO BUFFER
206 ;
207 ;NEXT, CHECK ALL BUFFERS TO SEE
208 ;IF THERE IS A BUFFER ALREADY IN
209 ;USE FOR THIS FILE, IF NOT GET
210 ;ADDRESS OF THE FIRST FREE BUFFER
211 ;
212 ;IF A1+1 = 0 THEN NO BUFFER, ERROR
213 ;
214 ;MOVE NAME INTO BUFFER NAME FIELD
215 ;TO HOLD IT OPEN, SET FILE MANAGER
216 ;PARAMETER LIST TO HAVE ADDRESSES
217 ;OF THE SECTIONS FO THE DOS BUFFER
218 ;AND THE ADDRESS OF THE FILENAME
219 ;
220 ;SET PARM LIST FOR DRIVE, SLOT
221 ;AND TEXT FILE TYPE
222 ;
223 ;SET BUFF01 TO POINT TO DATA
224 ;SECTION FOR DOS BUFFER, AND
225 ;TEMP1 TO THE WORKAREA OF BUFFER
226 ;
227 FNDDBF JSR NONAME
228 LDY LENFN
229 FNDDBF1 DEV
230 BMI NAMMOV
231 LDA (FNAME),Y
232 ORA #$00
233 STA NMBUFF,Y
234 BNE FNDDBF1
235 NAMMOV JSR GETBUF
236 BCC BUFFND
237 LDA A2+$01
238 BNE BUFFRE
239 LDX #$0C
240 JMP ERROR
241 BUFFRE STA A1+$01

```

```

242 LDA A2
243 STA A1
244 BUFFND JSR MOVNAM
245 JSR DOSPAR
246 JSR NAMADD
247 LDY #$05
248 ;
249 ;
250 ;DRIVE # IS LOADED HERE
251 ;
252 ;
253 LDA #$01
254 STA (PARMAD),Y
255 INY
256 ;
257 ;
258 ;SLOT # IS LOADED HERE
259 ;
260 ;
261 LDA #$06
262 STA (PARMAD),Y
263 LDA #$00
264 INY
265 STA (PARMAD),Y
266 LDY #$10
267 LDA (PARMAD),Y
268 STA BUFF1
269 INY
270 LDA (PARMAD),Y
271 STA BUFF1+$01
272 LDY #$0C
273 LDA (PARMAD),Y
274 STA TEMP1
275 INY
276 LDA (PARMAD),Y
277 STA TEMP1+$01
278 RTS
279 ;
280 ;
281 ;SETS PARM LIST TO OPEN
282 ;RECORD LENGTH IS $100 IF A-REG
283 ;=1, OR $00 IF A=0. ALSO, A NEW
284 ;FILE MAY BE CREATED IF A=0
285 ;AND MAY NOT WHEN A#0
286 ;
287 ;SAVE A-REG TO X-REG FOR FILE
288 ;MANAGER USE, SET VOLUME #,
289 ;GOTO FILE MANAGER
290 ;
291 ;
292 OPEN PHA
293 JSR FNDDBF
294 LDY #$00
295 LDA #$01
296 STA (PARMAD),Y
297 TYA
298 INY
299 INY
300 STA (PARMAD),Y
301 INY
302 PLA
303 TAX
304 STA (PARMAD),Y
305 INY
306 ;
307 ;
308 ;VOLUME # IS LOADED HERE
309 ;
310 ;
311 LDA #$00
312 STA (PARMAD),Y
313 JMP GOFLMN
314 ;
315 ;
316 ;READ FILE STARTS HERE
317 ;
318 ;SET A=1 (RECORD LENGTH=$100),
319 ;CREATE NO FILE. SET STORED RECORD
320 ;NUMBER AND POINTER IN SECTOR TO 0
321 ;
322 ;AT 'READFN', GET RECORD NUMBER
323 ;SET PARM LIST TO READ THAT REC #
324 ;GOT TO FILE MANAGER TO READ
325 ;
326 ;AT 'READ2', LOOP, COPYING ASCII
327 ;DOS BUFFER INTO KEYBOARD BUFFER
328 ;TO HOLD UNTIL A 'CR' IS HIT.
329 ;
330 ;IF Y=0, THEN WE ARE OUT OF DATA
331 ;IN THIS SECTOR, ADD COUNTER, GO
332 ;TO 'READFN'. IF A 'CR' IS HIT
333 ;THEN END OF STRING, SAVE LENGTH
334 ;OF STRING TO VARIABLE, MOVE
335 ;STRING INTO STRING MEMORY, AND
336 ;SAVE ADDRESS OF START OF STRING
337 ;INTO VARIABLE AREA.
338 ;
339 ;GOTO SUBROUTINE TO GET ADDRESS OF
340 ;NEXT VARIABLE SUBSCRIPT, AND
341 ;CHECK COUNTER TO SEE IF WE ARE
342 ;DONE WITH THE COMMAND YET
343 ;
344 ;ERROR TRAP FOR OUT OF MEMORY
345 ;AND END OF FILE IN HERE ALSO
346 ;
347 ;
348 READ JSR BASPRM
349 LDA #$01
350 JSR OPEN
351 LDY #$2A
352 LDA #$00
353 STA (TEMP1),Y
354 INY
355 STA XSAVE
356 STA (TEMP1),Y
357 INY
358 STA (TEMP1),Y

```

continued on next page

AMPER-SPEED: Fast Read And Write For Text Files (Cont.)

```

359 READFN LDY #02B
360 LDA (TEMP1),Y
361 PHA
362 DEY
363 LDA (TEMP1),Y
364 PHA
365 LDY #000
366 LDA #003
367 STA (PARMAD),Y
368 INY
369 STA (PARMAD),Y
370 PLA
371 INY
372 STA (PARMAD),Y
373 INY
374 PLA
375 STA (PARMAD),Y
376 LDA #000
377 INY
378 STA (PARMAD),Y
379 INY
380 STA (PARMAD),Y
381 JSR GOFLMN
382 LDY #000
383 LDX XSAVE
384 STY YSAVE
385 LDA (BUFF1),Y
386 BEQ EOF
387 CMP #0BD
388 BEQ ENSTRG
389 INC NCHAR
390 AND #07F
391 STA BUFF2,X
392 INX
393 INY
394 BNE READ2
395 NEWSEC LDY #02A
396 LDA (TEMP1),Y
397 CLC
398 ADC #001
399 STA (TEMP1),Y
400 INY
401 LDA (TEMP1),Y
402 ADC #000
403 STA (TEMP1),Y
404 STX XSAVE
405 JMP READFN
406 ENSTRG STY YSAVE
407 TYA
408 LDY #02C
409 STA (TEMP1),Y
410 LDX NCHAR
411 DEX
412 LDY #000
413 READ3 LDA FRETOP
414 BNE READ4
415 DEC FRETOP+001
416 READ4 DEC FRETOP
417 LDA BUFF2,X
418 STA (FRETOP),Y
419 DEX
420 CPX #0FF
421 BNE READ3
422 LDY #000
423 LDA NCHAR
424 STA (NAL),Y
425 LDA FRETOP
426 INY
427 STA (NAL),Y
428 LDA FRETOP+001
429 INY
430 STA (NAL),Y
431 JSR ADD
432 READ6 LDA ENDSTP+001
433 ADC #001
434 CMP FRETOP+001
435 BLT READ7
436 LDX #02D
437 JMPERR JMP ERROR
438 READ7 LDX #000
439 STX NCHAR
440 STX XSAVE
441 LDY YSAVE
442 INY
443 BEQ NEWSEC
444 BNE READ2
445 EOF LDX #005
446 BNE JMPERR
447 ;
448 ;
449 ;CONTINUE STARTS HERE
450 ;GET PASSED PARAMETERS, GET BUFFER
451 ;ADDRESSES SET UP, GET CURRENT
452 ;POSITION IN SECTOR, RECORD #
453 ;IS ALREADY SAVED FROM LAST CALL
454 ;
455 ;
456 CONTRD JSR BASPRM
457 JSR FNDBUF
458 LDY #02C
459 LDA (TEMP1),Y
460 TAY
461 STY YSAVE
462 JMP READ7
463 ;
464 ;
465 ;DE-ALLOCATE ROUTINE,NO CHANGE
466 ;DELETES THE VARIABLES PASSED
467 ;AND MOVES ALL OF REST DOWN TO
468 ;CLEARED SPACE

```

```

469 ;
470 ;
471 DELETE JSR GETADD
472 SEC
473 LDA LVAR
474 SBC #007
475 STA TEMP1
476 LDA LVAR+001
477 SBC #000
478 STA TEMP1+001
479 LDY #002
480 LDA (TEMP1),Y
481 STA OFFSET
482 INY
483 LDA (TEMP1),Y
484 STA OFFSET+001
485 CLC
486 LDA OFFSET
487 ADC TEMP1
488 STA A1
489 LDA OFFSET+001
490 ADC TEMP1+001
491 STA A1+001
492 LDY #000
493 MOVE LDA (A1),Y
494 STA (TEMP1),Y
495 INC TEMP1
496 BNE NEXTA1
497 INC TEMP1+001
498 NEXTA1 LDA A1
499 CMP ENDSTP
500 LDA A1+001
501 SBC ENDSTP+001
502 INC A1
503 BNE MOVE1
504 INC A1+001
505 MOVE1 BCC MOVE
506 SEC
507 LDA ENDSTP
508 SBC OFFSET
509 PHA
510 LDA ENDSTP+001
511 SBC OFFSET+001
512 STA ENDSTP+001
513 PLA
514 STA ENDSTP
515 JSR CHRGET
516 BNE DELETE
517 JMP EXIT1
518 ;
519 ;
520 ;WRITE STARTS HERE
521 ;
522 ;GET BUFFER ADDRESSES, OPEN FILE,
523 ;CREATING IF NOT FOUND,POSITION AT
524 ;START OF FILE
525 ;
526 ;COPY STRING LENGTH INTO PARM LIST
527 ;COPY STRING INTO KEYBOARD BUFFER,
528 ;THEN ADD A 'CR' TO IT. CALL FILE
529 ;MANAGER TO WRITE AT CURRENT
530 ;POSITION.
531 ;
532 ;GOTO SUBROUTINE TO GET NEXT
533 ;ADDRESS, AND CHECK TO SEE IF DONE
534 ;
535 ;
536 WRITE JSR BASPRM
537 LDA #000
538 JSR OPEN
539 LDY #000
540 LDA #00A
541 STA (PARMAD),Y
542 LDA #000
543 LDY #002
544 WRITE1 STA (PARMAD),Y
545 INY
546 CPY #006
547 BNE WRITE1
548 JSR GOFLMN
549 CONVRT LDY #000
550 LDA #004
551 STA (PARMAD),Y
552 LSR
553 INY
554 STA (PARMAD),Y
555 WRITE2 LDY #000
556 LDA (NAL),Y
557 LDY #006
558 STA (PARMAD),Y
559 TAX
560 LDA #000
561 INY
562 STA (PARMAD),Y
563 LDA #002
564 LDY #009
565 STA (PARMAD),Y
566 DEY
567 LDA #000
568 STA (PARMAD),Y
569 LDY #001
570 LDA (NAL),Y
571 STA TEMP1
572 INY
573 LDA (NAL),Y
574 STA TEMP1+001
575 LDY #000
576 MOVSTG DEX
577 CPX #0FF
578 BEQ STORCR
579 LDA (TEMP1),Y
580 ORA #0B0
581 STA BUFF2,Y
582 INY
583 BNE MOVSTG
584 STORCR LDA #0BD
585 STA BUFF2,Y
586 JSR GOFLMN
587 JSR ADD
588 JMP WRITE2

```

```

589 ;
590 ;
591 ;SUBROUTINE TO GET ADDRESS OF NEXT
592 ;STRING SUBSCRIPT, AND ADD TO
593 ;COUNT TO SEE IF DONE YET.
594 ;
595 ;IF DONE, EXIT ROUTINE, ELSE, RTS
596 ;
597 ;
598 ADD CLC
599 LDA NAL
600 ADC #003
601 STA NAL
602 LDA NAL+001
603 ADC #000
604 STA NAL+001
605 INC NREAD
606 BNE ADDS
607 INC NREAD+001
608 ADDS LDA NREAD+001
609 CMP VNR+001
610 BLT ADDEND
611 LDA NREAD
612 CMP VNR
613 BLT ADDEND
614 JSR EXIT
615 PLA
616 PLA
617 ADDEND RTS
618 ;
619 ;
620 ;APPEND STARTS HERE
621 ;GET PASS PARMS, GET BUFFER
622 ;GOTO WRITE
623 ;
624 ;
625 APPEND JSR BASPRM
626 JSR FNDBUF
627 JMP CONVRT
628 ;
629 ;
630 ;STORAGE FOR ZERO PAGE SAVE
631 ;CHARACTER TABLE FOR COMMANDS
632 ;AND ROUTINE ADDRESSES HERE
633 ;
634 ;
635 CHRTBL ASC 'R'
636 ASC 'C'
637 ASC 'D'
638 ASC 'W'
639 ASC 'A'
640 DFS #06
641 ADDTBL ADR READ-001
642 ADR CONTRD-001
643 ADR DELETE-001
644 ADR WRITE-001
645 ADR APPEND-001
646 DFS #0C

```


APPLE DARTS

Apple Darts

by Robert R. Devine
Small Computer Service
P.O. Box 10
Adona, Arkansas 72001

When you consider the relatively short period of time that personal computers have been around, it's really amazing how many good game programs have been written. Even now, it's not easy to sit down and create a program that someone else hasn't already written!! I think you'll find that APPLE DARTS takes an old and popular game, and makes it fun (as well as a bit safer) for all ages to play. No longer will Johnny need to worry about getting grounded for a month because he stuck a dart into dad's new paneling, mom's pretty lampshade, or his sister's backside!!

HOW TO ENTER THE PROGRAM:

To enter the shape table, you should first boot DOS, and then enter the monitor using **CALL-151**. You're now ready to enter the shape table which consists of **13 shapes**: the numbers 0 thru 9, a small square, a dart, and a ¼ circle. After it's all entered, return to Applesoft by entering **3D0G**, and save your shape table to disk using **BSAVE DB SHAPES A\$9400/L185, A\$9400,L185**. I always like to include the start address and length in any shape table or assembly routine names, as it makes it easier to access them for future use.

You're now ready to enter the Applesoft program. There are no special tricks involved, so you can simply enter it as listed.

HOW THE GAME WORKS

When you run the program you'll first be asked if you would like instructions, after which you'll be able to select one of two different game variations and difficulty levels. The dart board does take a little time to be drawn on the screen; however, it's rather detailed, and well worth the wait.

Once the board is drawn you can play as many games as you wish, without having to redraw the board. The reason the program starts drawing each circle very fast, and then seems to slow down, is because of the peculiarities of the way your Apple draws lines. When the lines are close to vertical, a simple increment of 1 in the **Y=SQR(Radius*2-X*2)** formula will fill in all the points between the lines. However, as the lines become more and more horizontal, it is necessary to keep reducing the formula's X increment, all the way down to .03, in order to be sure that all points are filled in. Once the board is drawn, the game is ready to play. Simply press the button on your game paddle to "throw" your dart.

GAME VARIATIONS

You'll note that the **scoring** for games 1 and 2 start in lines **350** and **400** respectively. **C(X)** is the value of the segment that the dart hit. **PS(PL)** is the score holder for player **PL**. The value of **F (1-3)** indicates whether the hit was in the bulls eye, inner ring, or outer ring respectively. **Testing to see if X** is an integer will tell whether the segment hit was **black or white**, depending on if it was in the inner or outer ring. With this information in hand, **you can easily add as many different game rule variations as you wish**. To do so, simply

change the **ON G GOSUB 350,400** in line **300** to **ON G GOSUB 350,400,GAME3,GAME4, GAME-N**, and write the new scoring routines.

HOW THE PROGRAM WORKS

The program first initializes the most frequently used variables at the beginning of the variable table, and DIMensions the needed array in line 1. It then jumps to line **15000**, which prints the title, and handles the instructions routine. Line **16000** then allows game selection, and difficulty level, after which it tests (**IF F>0** then we've played before) and resets previous scores to zero. Falling through to line **16010** indicates that this is the first game since a **RUN** command, whereupon the shape table is read from the disk (line **10000**), the color is set, and execution jumps to line **1000**, which begins execution of the graphic routines to draw the dart board.

Lines **1000** thru **1035** draw the first segmented circle. Rather than using a nice neat **FOR . . . NEXT** loop, it was necessary to **INCRement** a counter to bump the **X** value, and as the lines become closer to horizontal, slowly make the **INCRement** smaller and smaller. Line **1030** and **1035** test for which quadrants are to be drawn and **GOSUBs** the routines at **200** thru **230** which draw the lines.

Lines **1040** thru **1075** repeat the routine for the inner segmented circle, using the same logic, except that this time, some color switching is needed because we must erase parts of the lines that were drawn by the previous routine.

Lines **1080** thru **1090** draw the innermost black circle, again erasing parts of the lines drawn by the last routine.

Finally we get to lines **1095** thru **1100** which draw the **bulls eye**.

Line **1145** reads the DATA tables, getting the **X** and **Y** coordinates (**XC** and **YC**) of the number circles for each segment, then **GOSUBs** line **1200** which draws the ¼ circle in **4 ROTations**, and finally **GOSUBs** the routine at line **1210** thru **1225** which draws the numbers inside each circle. At this point the dart board is complete.

Line **1230** reads in all of the outer circle coordinates **OX(X)** and **OY(X)**, and then does the same for the inner circle coordinates **IX(X)** and **IY(X)**, as well as reading and **POKEing** the sound routine into memory.

As you've probably noticed by now, the major part of the program involves the initial setup, and is used only once!!

THE MAIN PROGRAM

The main program resides in lines **240** thru **280**. The program jumps from segment to segment, drawing the cursor, erasing it, and jumping to line **300** to see if the button was pushed. In difficulty level 1 the cursor always moves counterclockwise, however in level 2 . . . who knows? I originally planned to use the sound routine to make a sound as the cursor passed each segment, however it slowed things down too much.

The **F** flag tells which circle, inner or outer, we came from, or if the bulls eye was hit. **When the button is pressed**, the program jumps to lines **310** thru **320** to draw the dart, bumps the throw counter **T**, and **GOSUBs** the appropriate game scoring routine at **350** or **400**.

After updating the player's score **PS(PL)**, the routines test for a win. If none is found, execution jumps to line **610** which tests for change of player, prints the current scores, makes the sound (one beep or two) for the next player who throws a dart, jumps to line **100** or **110** to set color, and then to lines **310,315** or **320** which erase the dart.

Note: The **RETURNS** on lines **310** thru **320** (in this instance only) actually cause a jump to line **305**, which is another **RETURN**. This completes all **unRETURNed GOSUBs**, and puts us back at the very next segment on the board so that we can complete the same **FOR . . . NEXT** loop that we started.

If a winner is found in line **360**, game 1, the dart is first erased from the bulls eye in line **310**, and the program jumps to line **600** which declares the winner, **POPs** two **unRETURNed GOSUBs** off the stack, and goes into the new game routine.

If a winner is found in line **430**, game 2, execution jumps to line **620** which finds out where we came from, sets color, and then jumps to line **310,315** or **320** which again erase the dart. This time however, they **RETURN** to line **430**, which promptly sends the program to line **600** for the end of game routine.

I've tried to give a slightly clearer idea of what happens with each of the game scoring routines for those of you who might like to change the rules, or add a few game variations of your own.

THE POWER OF ON (X) GO . . .

You'll note the **mixed use of IF . . . THEN and ON (exp) GOSUB or ON (exp) GOTO statements** in the listing. While the **IF . . . THEN** is a powerful statement, the relatively infrequent use of the **ON (exp) GOSUB** and **ON (exp) GOTO** in most program listings, seems to indicate a lack of recognition that these can be even more powerful statements. Statements of this form allow, in only one statement, and almost limitless amount of testing and branching. They also have the benefit of allowing what amounts to multiple **IF . . . THENs** on the same line, without the need to worry about execution failure if all tests evaluate as false. I've also heard people complain about Applesoft not having the **IF . . . THEN . . . ELSE** branching statement, which is available with some other Basics. Did you know that the Basic statement **IF X=3 THEN 100 ELSE 300**, is exactly the same as the Applesoft **ON (X=3)+1 GOTO 300, 100 statement?**

VARIABLE TABLE:

XC,YC	X and Y coordinates for drawing number circles
OX(X), OY(X)	X and Y coordinates for outer ring darts
IX(X), IY(X)	X and Y coordinates for inner ring darts
C	Color of number circles (0 or 3)
F	Flag to indicate bulls eye, inner, or outer ring (1-3)
D	Difficulty level (1-2)
PL	Player number (1-2)

continued on next page

Apple Darts (Cont.)

PS(PL)	Score holder for player PL
T	Throw counter
G	Game selected (1-2)
Y	Vertical endpoint for drawing segmented circles
C(X)	Counterclockwise board order segment value
X	Horizontal endpoint for drawing segmented circles
X	Used in loops to sweep cursor around board
INC	Counter to bump X value when drawing segmented circles

89400-9489

```

9400- 0D 00 1C 00 2A 00 32 00
940B- 3C 00 48 00 52 00 5E 00
9410- 69 00 72 00 7D 00 88 00
941B- 8E 00 99 00 0C 25 1C 3F
9420- 17 36 2E 1E 0E 2D 05 24
942B- 04 00 24 BC 96 12 2D 1C
9430- 24 00 65 E4 3F 17 96 F1
943B- 2E 2D 35 00 0C 0C 3C 3F
9440- B7 92 15 2D 0C E4 07 00
944B- 3A 27 0C 0C 0C 36 AE 37
9450- 3E 00 38 27 2C 2D F5 AA
945B- 36 17 3F 1C 04 00 75 F6
9460- 3F 1C 24 E5 0C 0C 2D 06
946B- 00 0C 0C 3C 3F B7 52 1E
9470- 2E 00 E7 64 2D 15 F6 0E
947B- F6 3F 1C 24 00 E7 64 2D
9480- 15 36 77 1E 1E 3F 04 00
948B- 25 3F 36 2D 04 00 2D F5
9490- 3B 36 E4 3C 67 29 E5 24
949B- 00 24 24 24 2C 36 36 36
94A0- 2E 24 24 24 2C 32 36 36
94AB- 2E 24 24 24 36 36 25 24
94B0- 15 36 07 00 00 00 00 00
94BB- 00 D0

```

1

```

1 REM *****
2 REM * APPLE DARTS *
3 REM * BY ROBERT DEVINE *
4 REM * COPYRIGHT (C) 1982 *
5 REM * BY MICRO-SPARC INC *
6 REM * LINCOLN, MA. 01773 *
7 REM *****
10 X = 0: F = 0: PL = 0: D = 0: Y = 0: INC = 0: DIM DX(20), DY
(20), IX(20), IY(20), C(20): GOTO 15000
100 HCOLOR = 0: IF X / 2 = INT (X / 2) THEN HCOLOR = 3
102 FOR I = 1 TO SP: NEXT I
105 RETURN
110 HCOLOR = 3: IF X / 2 = INT (X / 2) THEN HCOLOR = 0
112 FOR I = 1 TO SP: NEXT I
115 RETURN
200 HPLLOT 140 + X, 79 + Y TO 140, 79 TO 140 - X, 79 - Y: RETURN
210 HPLLOT 140 + X, 79 - Y TO 140, 79 TO 140 - X, 79 + Y: RETURN
220 HPLLOT 140 + X, 79 + Y: HPLLOT 140 - X, 79 - Y: RETURN
230 HPLLOT 140 + X, 79 - Y: HPLLOT 140 - X, 79 + Y: RETURN
240 F = 3: ON (D = 1) GOTO 245: ON INT (RND (1) * 2) GOTO
245: FOR X = 20 TO 1 STEP - 1: GOTO 250
245 FOR X = 1 TO 20
250 GOSUB 100: DRAW 11 AT DX(X), DY(X): GOSUB 110: DRAW
11 AT DX(X), DY(X): GOSUB 300: NEXT
260 F = 2: ON (D = 1) GOTO 265: ON INT (RND (1) * 2) GOTO
265: FOR X = 20 TO 1 STEP - 1: GOTO 270
265 FOR X = 1 TO 20
270 GOSUB 110: DRAW 11 AT IX(X), IY(X): GOSUB 100: DRAW
11 AT IX(X), IY(X): GOSUB 300: NEXT :X = 0
280 F = 1: HCOLOR = 0: DRAW 11 AT 140, 79: FOR I = 1 TO SP
: NEXT I: HCOLOR = 3: DRAW 11 AT 140, 79: GOSUB 300: GOTO
240
300 IF PEEK ( - 16287 + PL - 1) > 127 THEN HCOLOR = 5:
ON F GOSUB 310, 315, 320: T = T + 1: ON G GOSUB 350, 4
00
305 RETURN
310 DRAW 12 AT 140, 79: RETURN
315 DRAW 12 AT IX(X), IY(X): RETURN
320 DRAW 12 AT DX(X), DY(X): RETURN
350 REM GAME 1 (SCORE 1-20)
355 IF C(X) = PS(PL) + 1 THEN PS(PL) = C(X)
360 IF PS(PL) = 20 AND F = 1 THEN HCOLOR = 3: GOSUB 310
: GOTO 600
370 GOTO 610
400 REM GAME 2 (SCORE 500)
405 IF F = 3 AND (X / 2 = INT (X / 2)) THEN PS(PL) = P
S(PL) - C(X): GOTO 610
410 IF F = 2 AND (X / 2 < ) INT (X / 2) THEN PS(PL) =
PS(PL) - C(X): GOTO 610
415 IF F = 1 THEN PS(PL) = PS(PL) + 50: GOTO 430
420 PS(PL) = PS(PL) + C(X)
430 IF PS(PL) > = 500 THEN GOSUB 620: GOTO 600

```

```

440 GOTO 610
600 HOME : VTAB 22: HTAB 10: FLASH : PRINT " PLAYER " PL
" WINS !!! ": NORMAL : POP : POP : FOR X = 1 TO 100
0: NEXT X: HOME : VTAB 21: PRINT "WANT ANOTHER GAME
?": GET A$: PRINT : IF A$ = "Y" THEN 16000
605 TEXT : HOME : PRINT CHR$(4)"CATALOG": END
610 FOR Y = 1 TO 100: NEXT Y: ON (G = 2) GOTO 612: ON (
T < 3) GOTO 615: T = 0
612 PL = PL + 1: IF PL = 3 THEN PL = 1
615 VTAB 21: PRINT "PLAYER 1 SCORE="PS(1)" PLAYER 2 SC
ORE="PS(2)" : VTAB 23: FOR Y = 1 TO PL: POKE 76
8, 100 * PL: POKE 769, 50: CALL 770: NEXT Y: PRINT TAB(
4)"PLAYER "PL"'S TURN --- THROW DART "T + 1
620 ON (F - 1) GOSUB 100, 110: ON (F - 1) GOTO 315, 320: HCOLOR = 3: GOTO 310
1000 INC = 1: X = 0
1005 IF X > 50 THEN INC = .50
1010 IF X > 71 THEN INC = .10
1015 IF X > 78.11 THEN INC = .03
1020 X = X + INC: IF X > 79 THEN 1040
1025 Y = SQR (6241 - X ^ 2)
1030 IF X < = 24 OR (X > = 46 AND X < = 63.2) OR X >
= 75 THEN GOSUB 200: GOSUB 230: GOTO 1005
1035 IF (X > 24 AND X < 46) OR (X > = 63.2 AND X < 75)
THEN GOSUB 210: GOSUB 220: GOTO 1005
1040 INC = 1: X = 0
1045 IF X > 33 THEN INC = .50
1050 IF X > 40.5 THEN INC = .10
1055 IF X > 44.5 THEN INC = .03
1060 X = X + INC: IF X > 45 THEN 1080
1065 Y = SQR (2025 - X ^ 2)
1070 IF X < = 13.5 OR (X > = 26 AND X < = 36) OR X >
42.75 THEN HCOLOR = 3: GOSUB 210: HCOLOR = 0: GOSUB
200: GOTO 1045
1075 IF (X > 13.5 AND X < 26) OR (X > 36 AND X < 42.75)
THEN HCOLOR = 3: GOSUB 200: HCOLOR = 0: GOSUB 210: GOTO
1045
1080 INC = 1: X = 0
1085 HCOLOR = 0: IF X > 11 THEN INC = .03
1090 X = X + INC: IF X > 15 THEN 1095
1092 Y = SQR (225 - X ^ 2): GOSUB 200: GOSUB 210: HCOLOR =
3: GOSUB 220: GOSUB 230: GOTO 1085
1095 HCOLOR = 3: FOR X = 0 TO 6 STEP .10
1100 Y = SQR (36 - X ^ 2): GOSUB 200: GOSUB 210: NEXT X
: GOTO 1145
1145 FOR A = 1 TO 20: READ XC, YC, C: GOSUB 1200: GOSUB 1
210: NEXT A: GOTO 1230
1200 HCOLOR = C: SCALE = 1: FOR X = 0 TO 40 STEP 16: ROT =
X: DRAW 13 AT XC, YC: NEXT X: RETURN
1210 IF C = 0 THEN C = 3: GOTO 1220
1215 C = 0
1220 HCOLOR = C: SCALE = 1: ROT = 0: N$ = STR$(A): IF LEN
(N$) = 1 THEN DRAW VAL (N$) + 1 AT XC, YC: RETURN
1225 DRAW VAL ( LEFT$(N$, 1)) + 1 AT XC - 4, YC: DRAW VAL
( RIGHT$(N$, 1)) + 1 AT XC + 3, YC: RETURN
1230 FOR X = 1 TO 20: READ DX(X), DY(X): NEXT X: FOR X =
1 TO 20: READ IX(X), IY(X): NEXT X: FOR X = 1 TO 20:
READ CX(X): NEXT X: FOR X = 770 TO 790: READ Y: POKE
X, Y: NEXT X
1235 VTAB 21: PRINT "PLAYER 1 SCORE="PS(1)" PLAYER 2 S
CORE="PS(2)" : VTAB 23: FOR Y = 1 TO PL: POKE 7
68, 100 * PL: POKE 769, 50: CALL 770: NEXT Y: PRINT TAB(
4)"PLAYER "PL"'S TURN --- THROW DART "T + 1
1240 GOTO 240
10000 PRINT CHR$(4)"BLOAD DB SHAPES A09400/L105": HIMEM:
37000: POKE 232, 0: POKE 233, 140: RETURN
10039 REM NUMBER COORDINATES AND CIRCLE COLORS XC, YC,
C
10040 DATA 171, 18, 0, 171, 141, 3, 129, 147, 3, 201, 48, 0, 129
, 11, 0, 207, 90, 0, 92, 120, 3, 72, 88, 3, 93, 38, 0, 201, 118, 3, 7
2, 67, 0, 118, 18, 3, 207, 69, 3, 79, 47, 3, 188, 129, 0, 79, 118, 0
, 131, 147, 0, 188, 38, 3, 118, 141, 0, 151, 11, 3
10049 REM OUTER TRACK COORDINATES DX(X), DY(X)
10050 DATA 130, 26, 117, 31, 184, 41, 93, 54, 88, 78, 89, 86, 94,
102, 103, 116, 116, 127, 131, 132, 150, 132, 165, 127, 178, 116
, 185, 182, 190, 87, 191, 70, 186, 55, 177, 41, 164, 31, 149, 26
10054 REM INNER TRACK COORDINATES IX(X), IY(X)
10055 DATA 135, 40, 124, 43, 113, 50, 106, 61, 103, 73, 104, 85
, 107, 96, 114, 107, 123, 114, 136, 117, 147, 117, 156, 114, 167
, 107, 174, 97, 177, 85, 176, 72, 173, 68, 166, 50, 157, 43, 146,
40
10059 REM COUNTERCLOCKWISE BOARD ORDER C(X)
10060 DATA 5, 12, 9, 14, 11, 8, 16, 7, 19, 3, 17, 2, 15, 10, 6, 13, 4,
18, 1, 20
10069 REM SOUND DATA
10070 DATA 173, 48, 192, 136, 208, 5, 206, 1, 3, 240, 9, 202, 208,
245, 174, 0, 3, 76, 2, 3, 96
15000 HOME : VTAB 8: PRINT TAB( 17)"DARTS": PRINT : PRINT
TAB( 13)"COPYRIGHT 1981": PRINT : PRINT TAB( 12)"
ROBERT R. DEVINE": PRINT : PRINT TAB( 13)"ADDNA AR
KANSAS": PRINT : PRINT
15010 PRINT TAB( 10)"WANT INSTRUCTIONS?": GET A$: PRINT
: PRINT : PRINT : IF A$ = "N" THEN 16000
15020 HOME : VTAB 6: PRINT "TO PLAY DARTS EACH PLAYER U
SES A GAME": PRINT : PRINT "PADDLE, PLAYER 1 USES P
DL(0) AND PLAYER": PRINT : PRINT "2 USES PL(1). SI
MPLY PRESS YOUR GAME": PRINT : PRINT "BUTTON WHEN T
HE FLASHING SQUARE IS ON": PRINT : PRINT "THE SPOT
YOU WANT."
15030 PRINT : PRINT "HIT ANY KEY TO CONTINUE": GET A$
: HOME : INVERSE : PRINT " GAME 1 ": NORMAL : PRINT
: PRINT "TO WIN YOU MUST HIT EACH NUMBER (1-20)": PRINT
: PRINT "IN ORDER, AND THEN YOU MUST HIT THE": PRINT
: PRINT "BULLS EYE.": PRINT
15040 INVERSE : PRINT " GAME 2 ": NORMAL : PRINT : PRINT
"THE FIRST PLAYER TO GET 500": PRINT : PRINT "POINT
S WINS. EACH HIT IS WORTH THE": PRINT : PRINT "SECT
ION VALUE. A BULLS EYE IS WORTH 50": PRINT : PRINT
"POINTS. TO MAKE THINGS INTERESTING.": PRINT
15050 PRINT "ANY BLACK SQUARE YOU HIT WILL": PRINT : PRINT
"BE DEDUCTED FROM YOUR SCORE.": PRINT : PRINT
16000 INPUT "WHICH GAME (1 OR 2) ": G: ON (G < 1 OR G >
2) GOTO 16000: PRINT : INPUT "DIFFICULTY (1=EASY, 2
=HARD) ": D: INPUT "ENTER SPEED OF PLAY (1=FAST, 255
=SLOW) ": SP: HOME : IF F > 0 THEN PL = 1: PS(1) = 0:
PS(2) = 0: T = 0: GOTO 1235
16010 GOSUB 10000: HOME : HGR : HCOLOR = 3: PL = 1: GOTO
10000

```



MICROCALC

Micro-Calc

by Stephen Bohn
6863 Emerald Ave.
Enon, OH 45323

Personal Accounting can be a real chore, and just about everyone has to deal with some. Micro-Calc is a program designed to handle your accounting and personal finance problems. Whether you have a simple task, or a complex series of calculations to perform, Micro-Calc can handle the job. The program was designed with flexibility in mind, because everyone's personal finance problems are different.

Micro-Calc is useful because it writes customized accounting programs. It will allow you to utilize up to **twenty rows by twenty columns** of information in any format you desire. **You merely set up the format, and Micro-Calc writes the program for you.** You can set up a new program for any situation and then save it by any name you choose. The new program will allow you to save, read, change or print whatever data you enter, as well as perform calculations on that data. With Micro-Calc you decide how you wish to tackle your bookkeeping problems.

PLANNING AHEAD

The key to your success with Micro-Calc is planning ahead. Your first step is to determine your format on paper. Suppose you want to calculate monthly balances for three accounts over the next three months. You need your present balance, finance charge, and monthly payment for each account to figure this out. Let's assume that you decide on this format:

	: BALANCE :	% :	MO. PMNT :	JUNE :	JULY :	AUG :
AUTO :	:	:	:	:	:	:
HOME :	:	:	:	:	:	:
STORE :	:	:	:	:	:	:
TOTAL :	:	:	:	:	:	:

THE FUNCTIONS

At this point, you must decide which one of three possible functions you wish to utilize at each intersection of a row and column, or **CELL** of your program. Your choices are **D** for data entry, **C** for Comment, and **F** for Formula or calculation. Also, at this time you must decide on your **column widths**. These are determined by the length of the column names, or by how much space will be required to display the information in each column. You can choose from one to nine characters in cells which contain comments, and from four to nine characters in cells which hold data or perform calculations. **Each row name is automatically allotted a ten character space.** After assigning functions to the cells, your model might now look like this:

	(8)	(3)	(7)	(8)	(8)	(8)
: BALANCE :	% :	MO. PMNT :	JUNE :	JULY :	AUG :	
AUTO :	D	:C:	D	: F :	F :	F :
HOME :	D	:C:	D	: F :	F :	F :
STORE :	D	:C:	D	: F :	F :	F :
:	C	:C:	C	: C :	C :	C :
TOTAL :	F	:C:	F	: F :	F :	F :

READY TO RUN

After charting out your model on paper, you are ready to run Micro-Calc. You are asked to enter the column widths, the column names, the row names, and which function (**C**, **D** or **F**) you wish to use at each **CELL**. It is imperative that you correct any errors now, because this information becomes the framework upon which your bookkeeping program is built.

After your format is complete, you are asked to enter the **comments**. These can be letters, numbers, or symbols and are primarily used for clarification within your program. There is one exception however: **a dollar sign (\$) cannot be the first character of a comment.** When comments are assigned to cells which are next to each other on the same row, you can write over the boundaries of the columns. By doing so, you can insert anything from sentences to lines between rows. All in all, it makes for a much neater display.

Once you've finished entering your comments, Micro-Calc will store all the information you've entered directly into memory. This process can take up to five minutes for a large program. After the information is stored, your personal accounting program will be displayed in a semi-finished state. Our example would look like this:

	: BALANCE :	% :	MO. PMNT :	JUNE :
AUTO :\$	.00 :	12% :	\$	.00 :
HOME :\$	.00 :	6% :	\$	.00 :
STORE :\$	.00 :	18% :	\$	.00 :
: ===== : *** : =====				
TOTAL :	:	*** :	:	:

Wherever :\$.00 : appears, it means that you will later be **entering data** at that location. The **blank spaces indicate where the results of calculations will be displayed**, or where a comment wasn't entered. Notice that only the first thirty-nine characters in memory are displayed, but that **by using the arrow keys, you can view the entire format.** By pressing either the right or left arrow keys, **the file will scroll ten characters either to the right or left.** The **comma** and **period** keys will also move right or left, but the file will only move **one character** at a time.

At this point you may change the column and row names or the comments by pressing **C**. Do this only when necessary because the computer will recompile all the data when you are finished making changes. If everything is correct, it is time to name your program, and then insert the disk you want your program saved to. Once Micro-Calc has finished saving the program format, you will be asked to reinsert the disk containing Micro-Calc.

ENTERING FORMULAS

The next procedure is to enter the formulas required for the calculations. It is important when entering the formulas to remember the **order of calculation**. All calculations are performed from **left to right within a row**, and the **columns are calculated from top to bottom**. This is especially important if any division is performed. To enter data into any of the **CELLS**, simply press the **ESCape** key and

enter the row number and column number of the cell you wish to use. The symbol **[R,C]** will appear on the screen, with **R** representing the row number and **C** representing the column number you have chosen. This feature will also allow you to utilize a previously calculated result in another calculation. The formula for calculation for the first row, fourth column in our example would be: **(BALANCE X %) - MO. PMNT + [1,1]*1.01 - [1,3]**. To enter this, press the **ESCape** key, **1** then **RETURN**, **1** then **RETURN**, *****, **1**, **.**, **0**, **1**, **.**, **ESC**, **1**, **RETURN**, **3**, **RETURN**, **RETURN**, and then **Y**. The rest of the calculations used in the example are as follows:

COLUMN 1

FIFTH ROW-[1,1]+[2,1]+[3,1]

COLUMN 3

[1,3]+[2,3]+[3,3]

COLUMN 4

FIRST ROW-[1,1]*1.01-[1,3]
SECOND ROW-[2,1]*1.005-[2,3]
THIRD ROW-[3,1]*1.015-[3,3]
FIFTH ROW-[1,4]+[2,4]+[3,4]

COLUMN 5

[1,4]*1.01-[1,3]
[2,4]*1.005-[2,3]
[3,4]*1.015-[3,3]
[1,5]+[2,5]+[3,5]

COLUMN 6

[1,5]*1.01-[1,3]
[2,5]*1.005-[2,3]
[3,5]*1.015-[3,3]
[1,6]+[2,6]+[3,6]

As you enter the formulas, you may **view the model you are constructing by pressing CTRL-V**. Here again, you may use the arrow keys to view the entire file. When finished, press the **R** key and the program will return to entering formulas. After the formulas have been entered you may review them; note however that the symbol **[R,C]** has been replaced by **D(R,C)** which the computer understands as data at that location. At this time you may also correct any formulas that you have entered incorrectly.

CUSTOM DATA ENTRY

When you are satisfied with all the formulas, you will be asked if you want to enter a customized data entry format. You do not have to use this feature, but it can be very helpful, especially if the data you will be entering is not in the same sequence as it is presented in your model. An example might be entering data from a cash register tape. **You could set up your customized data entry format so that you could run down the register tape and type in the numbers you need.** This would save hunting down the appropriate **CELL** in your model. To use this feature, just **enter the row and column numbers of the data locations you will be using in the**

Micro-Calc (Cont.)

order you want. You can also view the model at any time by pressing **CTRL-V**. After you have entered the entire sequence, you may view the model and also make any corrections if you need to. Pressing the **S** key will **save** the model. If you do not wish to use the customized data entry feature, pressing the **Q** key will allow you to continue.

Micro-Calc will then create the financial program into which you will be entering data. This is accomplished by writing and **EXEC**'ing a text file. At this point the program will ask you to reinsert the disk onto which you wish to save your model. Press the **SPACE BAR** to continue, and your new program will be written onto the disk under the name you have chosen. After Micro-Calc has finished writing your new program, it will automatically run the new program for you.

YOUR CUSTOMIZED PERSONAL FINANCE PROGRAM

First the new program will ask you to **enter data**. This can be done in one of five ways: **1) CUSTOM, 2) BY ROW, 3) BY COLUMN, 4) RANDOM, or 5) LOAD FILE**. With choice (1), you may enter the data in the order that you previously determined. With choice (2) **BY ROW**, you may enter the data a row at a time. Choice (3) **BY COLUMN**, allows you to enter the data a column at a time. Choice (4) **RANDOM**, will allow you to enter the data in any order you wish; however, you must specify each location as you enter the data. This choice is best suited for changing data or entering data for small programs. Choice (5) **LOAD FILE**, cannot be used until you have saved a file of data; this will be explained later.

When entering data, the program uses an **adding machine format**. It will **assume two places after the decimal point** unless you enter the decimal point. This is also true for calculations and the display. If you are using your program for other than accounting purposes, or you have a situation where you require more than two decimal places, you may modify the program to your specifications.

After all of the data locations have been filled, the program will perform all calculations and display the results. If the word **ERROR** is displayed at any of the locations, it means that the data calculated for that location is too large to fit into that column. It is important to realize at this point that the calculations will not be 100% accurate. The APPLE has difficulty handling very large or very small numbers. It also seems to have its own set of rules for **rounding numbers**. **BEWARE**. This will not be a problem in most circumstances, especially if simple calculations are performed. If you are performing complicated calculations or are relying on results of calculations for further calculating, the final answer may be incorrect. However this should be at most a 1% or 2% error. Most situations have been accounted for in the program by the routines in line 2250 of **SCRATCH-PAD**, but some problems may still occur.

REVIEWING THE FILE

The entire file can be viewed at this time by the use of the arrow keys. Thirty characters will be displayed at a time, which is a minimum of three columns. A handy feature of

the display is the ability to **freeze** part of the display and still **view another part**. If you press the **+** key, an **asterisk** will appear over the display. This asterisk may be positioned by the use of the **+** and **-** keys. **Everything displayed to the left of the asterisk will be frozen** on the display. Anything to the right can be changed by using the arrow keys. This is especially useful if you wish to look at two columns at the same time which are not normally adjacent to one another.

MORE OPTIONS

At this point you have several options available. By pressing **C**, you can change any data you have entered. Pressing **Q** will end the program, while pressing **P** will allow you to print out the display. To do this, you must **enter the numbers of the first and last columns and rows you want printed**. The last option, chosen by pressing **S**, will allow you to save all of the data that has been entered and calculated. This feature will allow you to save your model's information onto another disk if you so choose, or to add a special modifier to the name of the file saved. This is useful in any situation where you wish to save the data and possibly use it or modify it at a later time. The Save feature in itself is a very useful one because it allows you to update a file (without having to reenter all of the data) by using the **LOAD FILE** option the next time you run the program.

HOW IT WORKS

Micro-Calc is actually composed of five separate programs: two machine language programs and 3 Applesoft programs. When Micro-Calc is run, the first thing that the program does is to set **HIMEM to 32510**. This is to protect the information that will be stored in memory. The short machine language program, **CLEARER**, is then run. This program clears out the section of memory from **\$8000 to \$94FF** by writing blanks at each location. Micro-Calc continues prompting you to enter the format of your desired model. This information is made into a series of strings, one for each row. These strings are set up so that they will all be the same length. **Each string is then POKE'd directly into memory, character by character**. This process allows more efficient manipulation of this information than can be accomplished with Applesoft. This is because we can use another machine language program, **DISPLAYER**, to quickly read a section of memory and display it directly onto the screen. Once the data in this section is verified as correct, it is saved to the disk as a binary file. This process is also faster than saving the information as strings.

The rest of the information that you have entered in, such as formulas and the names of rows and columns, are stored as strings. These strings, however, are stored as program lines in the customized finance program. The formulas are stored in a specific sequence so that the calculations will be done in a predetermined order. All of these program strings, plus a series of DOS commands, are saved as the text file **WORKER**. This file is then **EXEC**'ed by the Micro-Calc program and then: **1) the second Applesoft program, SCRATCH-PAD**, is loaded into memory, **2) the new program lines** are added to it, **3) the program pauses** to let you insert the disk you want to save your program to, **4) saves the program** under the name you've chosen, and **5) runs your new program**.

HOW TO MODIFY MICRO-CALC

Micro-Calc is a complicated program, especially since it is actually four programs in one. Even though the program is flexible, there are some changes that can be implemented to make it even more useful to you. Depending on the type of change you wish to make, this could be an easy task or a very difficult one. For example, adding more columns would be relatively easy, but adding more rows would be quite difficult. If you want to add more columns, just be sure that the total length (sum of column widths plus boundaries) is less than 255 characters. It would also be easy to combine two or more columns to form one very wide one.

A useful feature that was not included with this program would be the ability to plot or graph the data entered. This could be done with a separate program, and would not require modification of your accounting program. When you save a file with your book-keeping program, all the data that you've entered plus all the data that has been calculated is saved to a disk in the text file **D.[program name][special modifier]**. Examine line numbers **1210 to 1350** in **SCRATCH-PAD** to see how to read this information.

HANDLING DATA

Micro-Calc uses a somewhat unique and useful method to handle data. Most of the data that is going to be displayed on the screen is **POKE'd** directly into a range of memory. Once this is done there are two ways of retrieving the information. You can either use **BASIC to PEEK** each memory location and read what is stored there, or use a machine language program to read the data. That is what the machine language program **DISPLAYER** does. What happens is that memory locations are read and then stored into new memory locations. The trick here is that **the new memory locations are actually the locations where the screen display is stored**. That is, if you place a character into one of these locations it will automatically be displayed on the screen without using a print command. This is great for displaying information except for three things. The **first** is that the screen is only 40 characters wide, while a file in this program can be up to 211 characters wide. The **second** is that characters are not stored as their ASCII equivalents. To display an "A" for example, you must store **193(\$C1)** in memory, and not **65(\$41)**, the normal ASCII value for "A". The **third** problem is that a starting point to view the file must be determined. This starting point is calculated in Applesoft and then **POKE'd** into memory **\$03**. Location **\$04** is also important if you are displaying a split file on the screen. It indicates the point on the screen at which the display will be "frozen". Location **\$03** is read by **DISPLAYER**, and a loop is performed 29 times (or fewer depending on the value in **\$04**) which reads one memory location and then stores the value found there into the corresponding location to be displayed. If you examine the listing for **DISPLAYER**, it may not seem to follow a logical order. This is because the screen display locations are stored in an interleaving fashion, that is, the next line on the screen is not the next memory location.

Micro-Calc (Cont.)

CLEARING MEMORY

Another trick used here is to clear out the memory you will be using before reading or writing information into it. This is accomplished with the machine language program **CLEARER**. The program clears a section of memory from \$8000 to \$94FF by writing the character for a space (\$A0) into each memory location. This is also accomplished with a loop, and uses a minimal number of commands. In fact, both of these machine language programs are simple, but they work very well. Direct memory storage is a very useful trick and could be applied to any number of programs that utilize large amounts of information that must be displayed.

It may take you some time and planning to set up your personal finance model with Micro-Calc, but it is worth the effort. You will have a program that is tailored to fit your needs, and one that can be saved, printed, and changed. Because the accounting program you create with Micro-Calc can stand alone, you can have Micro-Calc free to write other programs.

HOW TO ENTER MICRO-CALC:

Using Micro-Calc requires keying in the five programs shown here (two machine language and three Applesoft). It is very important that you **save all five programs onto the same disk.**

First, enter the monitor with CALL -151 and key in the **DISPLAYER** program from the following listing:

```
7F00- 18 A6 04 A4 03 B9 00 80
7F08- 9D 0B 05 E8 C8 E0 1D 90
7F10- F4 18 A6 04 A4 03 B9 00
7F18- 81 9D 0B 06 B9 00 82 9D
7F20- 8B 06 B9 00 83 9D 0B 07
7F28- B9 00 84 9D 8B 07 B9 00
7F30- 85 9D 33 04 B9 00 86 9D
7F38- B3 04 B9 00 87 9D 33 05
7F40- B9 00 88 9D B3 05 B9 00
7F48- 89 9D 33 06 B9 00 8A 9D
7F50- B3 06 B9 00 8B 9D 33 07
7F58- B9 00 8C 9D B3 07 B9 00
7F60- 8D 9D 5B 04 B9 00 8E 9D
7F68- DB 04 B9 00 8F 9D 5B 05
7F70- B9 00 90 9D DB 05 B9 00
7F78- 91 9D 5B 06 B9 00 92 9D
7F80- DB 06 B9 00 93 9D 5B 07
7F88- B9 00 94 9D DB 07 E8 C8
7F90- E0 1D 90 82 60 8D FF 7F
7F98- 18 AD 00 C0 C9 A0 D0 F9
7FA0- AD 10 C0 AD FF 7F 60 00
7FA8- FF AD 40 7D 85 36 AD 41
7FB0- 18 D8 A5 6A 8D C6 7F 8D
7FB8- CD 7F A5 69 8D C5 7F 8D
7FC0- CC 7F A0 03 B9 30 31 8D
7FC8- DD 7F C8 B9 30 31 8D DE
7FD0- 7F 18 A2 00 A9 7F 65 05
7FD8- 8D E3 7F 18 BD 28 7E 69
7FE0- 80 9D 00 82 E8 E0 0C F0
7FE8- 03 4C D4 7F 60
```

Save the program with **BSAVE DISPLAYER, A\$7F00, L\$ED.**

Next, key in the following program:

```
7F00- 18 A0 FF A9 A0 99 00 94
7F08- 99 00 80 99 00 81 99 00
7F10- 82 99 00 83 99 00 84 99
7F18- 00 85 99 00 86 99 00 87
7F20- 99 00 88 99 00 89 99 00
7F28- 8A 99 00 8B 99 00 8C 99
7F30- 00 8D 99 00 8E 99 00 8F
7F38- 99 00 90 99 00 91 99 00
7F40- 92 99 00 93 C8 C0 FF 90
7F48- BC 60
```

Save it with **BSAVE CLEARER, A\$7F00, L\$4A.**

Now you may key in the Applesoft program MICRO-CALC save it to disk, and then key in and save MICRO-CALC.II. Thereafter you need only RUN MICRO-CALC to use the system.



```
10 REM *****
11 REM * MICROCALC *
12 REM * BY STEPHEN BOHN *
13 REM * COPYRIGHT (C) 1982 *
14 REM * BY MICRO-SPARC INC *
15 REM * LINCOLN, MA. 01773 *
16 REM * ALL RIGHTS RESERVED *
17 REM *****
20 POKE 4,0: HIMEM: 32510
25 G% = "

26 REM 225 SPACES BETWEEN QUOTES IN LINE 25
30 PRINT CHR$(4);"BRUN CLEARER": PRINT CHR$(4);"BLO
AD DISPLAYER"
40 DIM LC(21),CL$(21),RO$(21),WK(21,21),TS$(21),CT$(21,
21)
50 Z% = CHR$(34):Y% = Z% + CHR$(58):W0% = "
": REM TEN SPACES
60 D$(3) = ".0$(4) = ".0$(5) = ".0$(6) =
".0$(7) = ".0$(8) = ".0$(9) =
"
70 D$(3) = ".00":D$(4) = ".00":D$(5) = ".00":D$(6) =
".00":D$(7) = ".00":D$(8) = ".00":D$(9) =
".00"
80 REM --- COLUMN WIDTHS ---
90 TEXT : HOME : POKE 33,41: FOR I = 5 TO 24: VTAB (I):
HTAB 21: PRINT ".....": NEXT I: POKE
33,40
100 VTAB 1: HTAB 1: INVERSE : PRINT "ENTER> COLUMN WIDT
H": PRINT "1-9 FOR WIDTH ": PRINT "Q TO QUIT
": PRINT "
110 FOR I = 1 TO 20: VTAB (4 + I): HTAB 1: PRINT "
": NEXT I: NORMAL : VTAB 4: HTAB 21: FOR I =
1 TO 20
120 GET A$: IF A$ = "Q" AND I > 1 THEN 210
130 IF A$ = CHR$(8) THEN I = I - 1: PRINT A$: CALL -
860: IF I < 1 THEN I = 1: PRINT "
140 IF A$ = CHR$(8) AND I < 1 THEN I = 1: PRINT "
150 IF A$ = CHR$(8) THEN 120
160 A = VAL (A$): IF A < 1 THEN 120
170 IF I < 1 THEN I = 1
180 LC(I) = A: PRINT A$: NEXT I
190 VTAB 4: HTAB 20: GET A$: IF A$ = CHR$(8) THEN FOR
I = 1 TO 20:LC(I) = 0: GOTO 90
200 IF A$ < > CHR$(81) THEN PRINT CHR$(7): GOTO
190
210 NC = I - 1: VTAB 1: HTAB 1: PRINT W0%;W0%: PRINT W0%
:W0%: PRINT W0%;W0%
220 REM --- COLUMN NAMES ---
230 VTAB 3: HTAB 20: CALL - 860: INVERSE : VTAB 1: HTAB
1: PRINT "ENTER> COLUMN NAMES": NORMAL : PRINT "
": FOR I = 1 TO NC:I = 0: VTAB 3: HTAB 1: PRINT
W0%;
240 VTAB 2: HTAB 1: INVERSE : PRINT "NAME FOR COLUMN #
":I: VTAB 4: HTAB 1: PRINT "COL. NAME ": NORMAL : VTAB
3: HTAB 1: PRINT W0%;
250 LM = LC(1):B% = ""
260 VTAB 3: HTAB 10: GET A$
270 IF A$ = CHR$(8) THEN IF B% = "" THEN 320
```

```
280 IF A$ = CHR$(8) THEN VTAB 3: HTAB 1: PRINT W0%:
INVERSE : VTAB 4 + 1: HTAB 1: PRINT W0%: NORMAL :
B% = " ": GOSUB 380:B% = "": GOTO 260
290 IF LEN (B%) = LM THEN IF A$ < > CHR$(13) THEN
PRINT CHR$(7): GOTO 260
300 IF A$ = CHR$(13) THEN 350
310 B% = B% + A$: GOSUB 380: VTAB 3: HTAB 10 - LEN (B%)
: INVERSE : PRINT B%: NORMAL : GOTO 260
320 B% = " ": GOSUB 380: VTAB 3: HTAB 1: PRINT W0%: VTAB
4 + 1: HTAB 1: INVERSE : PRINT "
": NORMAL
:I = I - 1: IF I < 1 THEN I = 1
330 GOSUB 380: VTAB 4 + 1: HTAB 1: INVERSE : PRINT "
": NORMAL : VTAB 2: HTAB 18: PRINT "
": INVERSE
: VTAB 2: HTAB 18: PRINT I: NORMAL
340 B% = "": GOTO 250
350 IF LEN (B%) = LM THEN CL$(1) = B%:B% = "": GOTO 37
8
360 CL$(1) = B% + LEFT$(W0%,LM - LEN (B%))
370 VTAB 3: HTAB 1: PRINT W0%: HTAB 1: VTAB 4 + 1: INVERSE
: PRINT CL$(1): VTAB 4 + 1: HTAB 18: PRINT "-":I:
NORMAL : NEXT I: NORMAL : GOTO 420
380 IF LEN (B%) > 2 THEN VTAB 3: HTAB 20 + 1: PRINT MID$
(B%,3,1):
390 IF LEN (B%) > 1 THEN VTAB 2: HTAB 20 + 1: PRINT MID$
(B%,2,1):
400 VTAB 1: HTAB 20 + 1: PRINT LEFT$(B%,1): RETURN
410 REM --- ROW NAMES ---
420 VTAB 1: HTAB 1: PRINT W0%;W0%: PRINT W0%;W0%: PRINT
W0%;W0%
430 B% = "":A$ = "": VTAB 1: HTAB 1: INVERSE : PRINT "HO
W MANY ROWS ?": NORMAL :N = 0
440 GET A$:N = N + 1: IF A$ = CHR$(13) THEN 470
450 IF A$ = CHR$(8) THEN N = 0: GOTO 420
460 PRINT A$:B% = B% + A$: IF N < 2 THEN 440
470 A = VAL (B%): IF A < 1 OR A > 20 THEN 420
480 VTAB 1: HTAB 1: PRINT W0%;W0%: PRINT W0%;W0%: PRINT
W0%;A$ = "":B% = "
490 VTAB 4: HTAB 11: PRINT " ROW NAME": FOR XX = 1 TO
20: VTAB 4 + XX: HTAB 9 + (XX < 10): PRINT XX:"
": NEXT XX
500 VTAB 1: HTAB 1: INVERSE : PRINT "ENTER> ROW NAMES
": NORMAL :NR = A: FOR I = 1 TO NR
510 FX = 0
520 FOR J = 1 TO 10
530 VTAB 4 + 1: HTAB 20: GET A$: IF A$ = CHR$(13) THEN
RO$(I) = B%:B% = "": GOTO 600
540 IF FX = 1 AND A$ < > CHR$(13) THEN PRINT CHR$(
7): GOTO 530
550 IF A$ = CHR$(8) AND B% = "" THEN I = I - 1: IF I <
1 THEN I = 1
560 IF A$ = CHR$(8) AND B% = "" THEN RO$(I) = "": VTAB
4 + 1: HTAB 11: PRINT W0%: GOTO 520
570 IF A$ = CHR$(8) THEN B% = "":RO$(I) = "": VTAB 4 +
1: HTAB 11: PRINT W0%: GOTO 520
580 VTAB 4 + 1: HTAB 20 - LEN (B%):B% = B% + A$: PRINT
B%:
590 NEXT J:FX = 1: GOTO 530
600 IF LEN (RO$(I)) = 10 THEN NEXT I: GOTO 630
610 RO$(I) = LEFT$(W0%,10 - LEN (RO$(I))) + RO$(I):
NEXT I
```

continued on next page

Micro-Calc (Cont.)

```

620 REM - CHANGE SOMETHING -
630 VTAB 1: HTAB 1: PRINT W0$;W0$: PRINT W0$;W0$: PRINT
W0$;W0$
640 INVERSE : VTAB 1: HTAB 1: PRINT "DO YOU WISH TO
": PRINT "CHANGE A ROW OR ": PRINT "COLUMN NA
ME (Y/N) ?": GET A$: PRINT A$: IF A$ = "N" THEN 7
18
650 IF A$ < > "Y" THEN 640
660 VTAB 1: HTAB 1: PRINT "CHANGE 1)ROW NAME ": PRINT
" 2)COLUMN NAME": PRINT " 3)QUIT/ALL OK
": PRINT W0$;W0$: VTAB 4: HTAB 1: PRINT "WHICH ?
": GET A$: PRINT A$: A = VAL (A$): IF A <
1 OR A > 3 THEN 660
670 NORMAL : IF A = 3 THEN 710
680 B$ = "": ON A GOSUB 1210,1090
690 VTAB 2: HTAB 1: PRINT W0$;W0$: VTAB 4: HTAB 1: PRINT
"MORE(Y/N)?":W0$: VTAB 4: HTAB 1: GET A$: PRINT A
$: IF A$ = "Y" THEN INVERSE : GOTO 660
700 IF A$ < > "N" THEN 690
710 VTAB 4: CALL - 860
720 VTAB 5: HTAB 1: INVERSE : PRINT "C=COMMENT ": PRINT
"F=FORMULA ": PRINT "D=DATA ": PRINT "<=BKSPACE
": NORMAL
730 POKE 33,20: VTAB 1: HTAB 1: PRINT W0$;W0$;W0$;W0$;W
0$;W0$;W0$;W0$: POKE 33,40: INVERSE : FOR XX = 9 TO
24: HTAB 1: VTAB XX: PRINT W0$: NEXT XX: NORMAL
740 FOR I = 1 TO 20: VTAB 4 + 1: HTAB 9 + (1 < 10): PRINT
I: NEXT I: VTAB 4: HTAB 21: PRINT "12345678912345
67890":
750 POKE 33,41: FOR N = 1 TO NR: FOR M = 1 TO NC: GOSUB
760: GOSUB 880: GOSUB 930: NEXT M: NEXT N: GOTO 960
760 INVERSE : GOSUB 880
770 IF FY = 0 THEN GOSUB 930
780 NORMAL
790 VTAB 4 + N: HTAB 20 + M: GET A$
800 IF LC(M) < 4 THEN A$ = "C"
810 IF A$ = "C" THEN PRINT A$;WK(N,M) = 1: RETURN
820 IF A$ = "F" THEN PRINT A$;WK(N,M) = 2: RETURN
830 IF A$ = "D" THEN PRINT A$;WK(N,M) = 3:ND = ND + 1
: RETURN
840 IF A$ = CHR$(8) THEN NORMAL : GOSUB 880: GOSUB 9
30:M = M - 1: IF M = 0 THEN M = NC:N = N - 1: IF N =
0 THEN RETURN
850 REM BACKSPACE
860 GOTO 760
870 REM --- PRINT ROUTINE ---
880 AA = LEN (CL$(M)): IF AA = 0 THEN C1$ = CHR$(32):
C2$ = C1$:C3$ = C1$: GOTO 920
890 IF AA = 1 THEN C1$ = CHR$(32):C2$ = C1$:C3$ = LEFT$
(CL$(M),1): GOTO 920
900 IF AA = 2 THEN C1$ = CHR$(32):C2$ = LEFT$ (CL$(M
),1):C3$ = RIGHT$ (CL$(M),1): GOTO 920
910 C1$ = LEFT$ (CL$(M),1):C2$ = MID$ (CL$(M),2,1):C3$
= MID$ (CL$(M),3,1)
920 VTAB 1: HTAB 20 + M: PRINT C1$: VTAB 2: HTAB 20 +
M: PRINT C2$: VTAB 3: HTAB 20 + M: PRINT C3$: RETURN
930 VTAB 4 + N: HTAB 11: PRINT RO$(N);
940 RETURN
950 REM --- IS THIS CORRECT ---
960 B$ = "": VTAB 1: HTAB 1: PRINT W0$;W0$: PRINT W0$;W0
$
970 VTAB 1: HTAB 1: INVERSE : PRINT "IS THIS CORRECT ?"
: NORMAL : GET A$: PRINT A$: IF A$ = "Y" THEN GOTO
1340
980 IF A$ < > "N" THEN 960
990 VTAB 1: HTAB 1: PRINT W0$;W0$
1000 VTAB 1: HTAB 1: INVERSE : PRINT "ENTER COLUMN NO."
: NORMAL : GET A$: IF A$ = CHR$(13) THEN 1000
1010 PRINT A$;B$ = A$: GET A$: IF A$ = CHR$(13) THEN
1030
1020 PRINT A$;B$ = B$ + A$
1030 M = VAL (B$): IF M < 1 OR M > NC THEN 990
1040 VTAB 2: HTAB 1: INVERSE : PRINT "ENTER ROW NO.":
: NORMAL : GET A$: IF A$ = CHR$(13) THEN 1040
1050 PRINT A$;B$ = A$: GET A$: IF A$ = CHR$(13) THEN
1070
1060 PRINT A$;B$ = B$ + A$
1070 N = VAL (B$): IF N < 1 OR N > NR THEN 1040
1080 GOSUB 760: GOSUB 880: NORMAL : VTAB 4 + N: HTAB 11
: PRINT RO$(N): GOTO 960
1090 VTAB 1: HTAB 1: POKE 33,20: PRINT W0$;W0$;W0$;W0$;
W0$;W0$;W0$;W0$: VTAB 1: HTAB 1: INVERSE : INPUT "E
NTER COLUMN NO.->":A$: NORMAL :M = VAL (A$): IF M <
1 OR M > NC THEN 1090
1100 POKE 33,40: VTAB 2: HTAB 1: PRINT "OLD NAME =":CL$
(M)
1110 VTAB 4: HTAB 1: INVERSE : PRINT "NEW NAME ": NORMAL
1120 VTAB 4: HTAB 11: GET A$: IF A$ = CHR$(13) THEN VTAB
1: HTAB 1: PRINT W0$;W0$: GOTO 1100
1130 VTAB 4: HTAB 10: PRINT " " + B$ + A$
1140 IF A$ = CHR$(8) THEN B$ = "": VTAB 4: HTAB 11: PRINT
W0$: GOTO 1120
1150 B$ = B$ + A$: IF LEN (B$) = LC(M) THEN 1170
1160 GOTO 1120
1170 VTAB 4: HTAB 11: GET A$: IF A$ < > CHR$(13) THEN
PRINT CHR$(7): GOTO 1170
1180 IF B$ = "" THEN RETURN
1190 IF LEN (B$) = LC(M) THEN CL$(M) = B$: GOSUB 880:
: INVERSE : VTAB 4 + M: HTAB 1: PRINT CL$(M): NORMAL
: RETURN
1200 CL$(M) = B$ + LEFT$ (W0$,LC(M) - LEN (B$)): GOSUB
880: INVERSE : VTAB 4 + M: HTAB 1: PRINT CL$(M): NORMAL
: RETURN
1210 VTAB 1: HTAB 1: POKE 33,20: PRINT W0$;W0$;W0$;W0$;
W0$;W0$;W0$;W0$: VTAB 1: HTAB 1: INVERSE : INPUT "
ENTER ROW NO. ->":A$: NORMAL :N = VAL (A$): IF N <
1 OR N > NR THEN 1210

```

```

1220 POKE 33,40: VTAB 2: HTAB 1: PRINT "OLD NAME =":RO$
(N)
1230 VTAB 4: HTAB 1: INVERSE : PRINT "NEW NAME ": NORMAL
1240 VTAB 4: HTAB 11: GET A$: IF A$ = CHR$(13) THEN VTAB
1: HTAB 1: PRINT W0$;W0$: GOTO 1300
1250 VTAB 4: HTAB 10: PRINT " " + B$ + A$
1260 IF A$ = CHR$(8) THEN B$ = "": VTAB 4: HTAB 11: PRINT
W0$: GOTO 1240
1270 B$ = B$ + A$: IF LEN (B$) = 10 THEN 1290
1280 GOTO 1240
1290 VTAB 4: HTAB 11: GET A$: IF A$ < > CHR$(13) THEN
PRINT CHR$(7): GOTO 1290
1300 IF B$ = "" THEN RETURN
1310 IF LEN (B$) = 10 THEN RO$(N) = B$: GOSUB 930: RETURN
1320 RO$(N) = LEFT$ (W0$,10 - LEN (B$)) + B$: GOSUB
930: RETURN
1330 REM --- ENTER COMMENTS ---
1340 HOME : VTAB 10: PRINT "ONE MOMENT PLEASE.":X = FRE
(0): HOME
1350 VTAB 4: HTAB 12: INVERSE : PRINT "ENTER COMMENTS":
NORMAL
1360 FOR N = 1 TO NR: FOR M = 1 TO NC: IF WK(N,M) = 1 THEN
1380
1370 NEXT M: NEXT N:B$ = "": GOTO 1520
1380 VTAB 11: HTAB 8: PRINT " ": VTAB 11: HTAB 8: PRINT
N: VTAB 9: HTAB 21: PRINT " ": VTAB 9: HTAB 21: PRINT
M
1390 VTAB 10: HTAB 21: PRINT W0$: VTAB 10: HTAB 21: INVERSE
: PRINT CL$(M):XL = LC(M): VTAB 11: HTAB 10: PRINT
W0$: VTAB 11: HTAB 20 - LEN (RO$(N)): PRINT RO$(N)
: NORMAL
1400 NN = N:MM = M + 1: IF MM > NC THEN MM = 0:NN = 0
1410 IF WK(NN,MM) = 1 THEN XL = LC(M) + 1
1420 VTAB 11: HTAB 21 + LEN (B$): GET A$: IF A$ = CHR$
(13) THEN 1470
1430 IF A$ = CHR$(8) THEN B$ = "": VTAB 11: HTAB 20: PRINT
W0$;W0$: GOTO 1420
1440 IF LEN (B$) < 1 AND A$ = CHR$(36) THEN PRINT CHR$
(7): GOTO 1420
1450 IF LEN (B$) < XL THEN B$ = B$ + A$: VTAB 11: HTAB
21: PRINT B$: GOTO 1420
1460 PRINT CHR$(7): GOTO 1420
1470 CT$(N,M) = B$:B$ = "": VTAB 11: HTAB 21: PRINT W0$:
IF LEN (CT$(N,M)) > LC(M) THEN 1490
1480 CT$(N,M) = CT$(N,M) + LEFT$ (W0$,LC(M) - LEN (CT$
(N,M)))
1490 IF FY = 1 THEN RETURN
1500 GOTO 1370
1510 REM --- STORE IN MEMORY ---
1520 TEXT : HOME : VTAB 8: HTAB 8: INVERSE : PRINT "
COMPILING DATA ": VTAB 9: HTAB 8: PRINT "MAY
TAKE ": FLASH : PRINT "SEVERAL": INVERSE : PRINT
" MINUTES": NORMAL
1530 FOR N = 1 TO NR:TS$(N) = CHR$(50): FOR M = 1 TO
NC: IF WK(N,M) = 2 THEN TS$(N) = TS$(N) + D$(LC(M))
+ CHR$(50): GOTO 1580
1540 IF WK(N,M) = 3 THEN TS$(N) = TS$(N) + D$(LC(M)) +
CHR$(50): GOTO 1580
1550 NN = N:MM = M + 1: IF MM > NC THEN MM = 0:NN = 0
1560 IF LEN (CT$(N,M)) > LC(M) AND WK(NN,MM) = 1 THEN
TS$(N) = TS$(N) + CT$(N,M): GOTO 1580
1570 TS$(N) = TS$(N) + CT$(N,M) + CHR$(50)
1580 NEXT M: NEXT N:TS$(0) = CHR$(50): FOR M = 1 TO N
C:TS$(0) = TS$(0) + CL$(M) + CHR$(50): NEXT M
1590 U = LEN (TS$(1)): POKE 32742,U + 1: FOR I = 0 TO N
R: POKE 5,1 + I:G$ = " " + TS$(1): CALL 32688:TS$(I
) = "": NEXT
1600 REM --- DISPLAY ---
1610 TEXT : HOME : INVERSE : VTAB 4: HTAB 11: PRINT W0$
;W0$;W0$: NORMAL : PRINT : POKE 3,1: CALL 32512: FOR
I = 1 TO NR
1620 VTAB 4 + I: HTAB 1: PRINT RO$(I): NEXT I: NORMAL
:B$ = "Z"
1630 VTAB 1: HTAB 1: INVERSE : PRINT "(- OR < SHIFT FIL
E) OR -) S = SAVE ": PRINT " (< OR =RESTAR
T) C = CHANGE ": NORMAL
1640 IF B$ < > "Z" THEN 1610
1650 VTAB 4: HTAB 11: GET A$: IF A$ = CHR$(8) THEN Y =
Y + 10
1660 IF A$ = CHR$(21) THEN Y = Y - 10
1670 IF A$ = CHR$(60) OR A$ = CHR$(44) THEN Y = Y +
1
1680 IF A$ = CHR$(62) OR A$ = CHR$(46) THEN Y = Y -
1
1690 IF A$ = CHR$(83) THEN 1750
1700 IF A$ = CHR$(67) THEN 1770
1710 IF A$ = CHR$(81) THEN PRINT : HOME : VTAB 12:
INVERSE : PRINT "TO ERASE CURRENT FORMAT AND RESTA
RT ": PRINT "TYPE 'E' ": NORMAL : GET B$: IF
B$ = "E" THEN PRINT B$: PRINT CHR$(4):"RUN MICRO
-CALC"
1720 IF Y > 255 THEN Y = Y - 255
1730 IF Y < 0 THEN Y = 255 + Y
1740 POKE 3,Y: CALL 32512: GOTO 1640
1750 VTAB 1: HTAB 1: CALL - 860: VTAB 2: CALL - 860: VTAB
1: HTAB 9: INVERSE : PRINT "IS ALL INFORMATION CORR
ECT ?": GET A$: NORMAL : PRINT A$: IF A$ = "Y" THEN
HOME : GOTO 2260
1760 IF A$ < > "N" THEN PRINT CHR$(7): GOTO 1750
1770 TEXT : HOME :B$ = " "
1780 VTAB 1: HTAB 1: INVERSE : PRINT " ENTER NU
MBER TO CHANGE ": PRINT "1)COLUMN NAME 2
)ROW NAME 3)COMMENTS ": PRINT "4)ALL CORRECT 5
)VIEW FILE ": NORMAL
1790 VTAB 4: HTAB 1: INVERSE : PRINT "CHANGES WILL NOT
APPEAR ON DISPLAY YET": NORMAL : VTAB 1: HTAB 40
: GET A$:A = VAL (A$): IF A < 1 OR A > 5 THEN 1790
1800 PRINT A$:
1810 IF A = 4 THEN FX = 0: GOTO 1520
1820 FX = 1: ON A GOSUB 1840,1960,2080,2530,2530
1830 GOTO 1770
1840 TEXT : HOME : VTAB 1: HTAB 1: INVERSE : INPUT "ENT
ER COLUMN NUMBER ->":A$: NORMAL :M = VAL (A$): IF
M < 1 OR M > NC THEN 1780

```



```

1850 VTAB 2: HTAB 1: PRINT CL$(M)
1860 VTAB 4: HTAB 1: INVERSE : PRINT "ENTER COLUMN NAME
->"; NORMAL
1870 VTAB 4: HTAB 21: GET A$: IF A$ = CHR$(13) THEN 1
930
1880 VTAB 4: HTAB 21: PRINT B$ + A$
1890 IF A$ = CHR$(8) THEN B$ = "": GOTO 1870
1900 B$ = B$ + A$: IF LEN(B$) = LC(M) THEN 1920
1910 GOTO 1870
1920 VTAB 4: HTAB 21: GET A$: IF A$ < > CHR$(13) THEN
PRINT CHR$(7); GOTO 1920
1930 IF B$ = "" THEN RETURN
1940 IF LEN(B$) = LC(M) THEN CL$(M) = B$: B$ = "": RETURN

1950 CL$(M) = B$ + LEFT$(W$(LC(M) - LEN(B$)): RETURN

1960 TEXT : HOME : VTAB 1: HTAB 1: INVERSE : INPUT "ENT
ER ROW NUMBER ->"; A$: NORMAL : N = VAL(A$): IF N <
1 OR N > NR THEN 1780
1970 VTAB 2: HTAB 1: PRINT R$(N)
1980 VTAB 4: HTAB 1: INVERSE : PRINT "ENTER ROW NAME ->
"; NORMAL
1990 VTAB 4: HTAB 18: GET A$: IF A$ = CHR$(13) THEN 2
050
2000 VTAB 4: HTAB 18: PRINT B$ + A$
2010 IF A$ = CHR$(8) THEN B$ = "": GOTO 1990
2020 B$ = B$ + A$: IF LEN(B$) = 10 THEN 2040
2030 GOTO 1990
2040 VTAB 4: HTAB 21: GET A$: IF A$ < > CHR$(13) THEN
PRINT CHR$(7); GOTO 2040
2050 IF B$ = "" THEN RETURN
2060 IF LEN(B$) = 10 THEN R$(N) = B$: B$ = "": RETURN

2070 R$(N) = LEFT$(W$(10 - LEN(B$)): B$: RETURN

2080 TEXT : HOME
2090 VTAB 1: HTAB 1: INVERSE : INPUT "ENTER ROW NUMBER
->"; A$: N = VAL(A$): IF N < 1 OR N > NR THEN 1780
2100 VTAB 2: HTAB 1: INPUT "ENTER COLUMN NUMBER ->"; A$:
M = VAL(A$): IF M < 1 OR M > NC THEN 1780
2110 IF WK(N,M) < > 1 THEN RETURN
2120 VTAB 18: HTAB 1: PRINT "OLD COMMENT ->"; NORMAL :
PRINT CT$(N,M): INVERSE
2130 VTAB 5: HTAB 1: PRINT R$(N); VTAB 4: HTAB 12: PRINT
CL$(M); NORMAL : B$ = ""
2140 NN = N: MM = M + 1: IF MM > NC THEN MM = 0: NN = 0
2150 LX = LC(M): IF WK(NN,MM) = 1 THEN LX = LX + 1
2160 VTAB 5: HTAB 12 + LEN(B$): GET A$
2170 VTAB 5: HTAB 12: PRINT B$ + A$
2180 IF A$ = CHR$(13) THEN 2240
2190 IF A$ = CHR$(8) THEN B$ = "": VTAB 5: HTAB 12: PRINT
W$: GOTO 2160
2200 B$ = B$ + A$: IF LEN(B$) = LX THEN PRINT CHR$(
7); GOTO 2220
2210 GOTO 2160
2220 VTAB 5: HTAB 13 + LX: GET A$: IF A$ = CHR$(8) THEN
B$ = "": VTAB 5: HTAB 13: PRINT W$: W$: GOTO 2160

2230 IF A$ < > CHR$(13) THEN PRINT CHR$(7); GOTO
2220
2240 IF LEN(B$) < LX THEN CT$(N,M) = B$ + LEFT$(W$
,LX - LEN(B$)): RETURN
2250 CT$(N,M) = B$: RETURN
2260 HOME : B$ = "": VTAB 1: HTAB 1: INVERSE : PRINT "
ENTER NAME OF FILE
"; NORMAL : PRINT
"->";
2270 VTAB 2: HTAB 4: PRINT CHR$(68) + CHR$(45)
2280 VTAB 1: HTAB 1: GET A$
2290 IF A$ = CHR$(13) THEN 2360
2300 IF A$ = CHR$(8) THEN B$ = "": GOTO 2260
2310 VTAB 2: HTAB 3: PRINT B$ + A$ + CHR$(68) + CHR$(
45)
2320 B$ = B$ + A$: IF LEN(B$) > 18 THEN PRINT CHR$(
7); GOTO 2340
2330 GOTO 2280
2340 PRINT : PRINT "USE 18 CHARACTERS OR LESS !": VTAB
1: HTAB 1: GET A$: IF A$ = CHR$(8) THEN B$ = "": GOTO
2260
2350 IF A$ < > CHR$(13) THEN PRINT CHR$(7); B$ =
"": GOTO 2260
2360 IF B$ = "" THEN 2280
2370 VTAB 6: PRINT "PLEASE INSERT DISK THAT YOU ARE SAV
ING": INVERSE : PRINT B$: NORMAL : PRINT " ONTO": PRINT
: INVERSE : VTAB 9: HTAB 8: INPUT "PRESS RETURN WHE
N READY"; A$: NORMAL
2380 PRINT : PRINT CHR$(4); "BSAVE DISPLAYER"; CHR$(4
4); "L$ED"
2390 N$ = B$: PRINT A$: PRINT : PRINT CHR$(4); "BSAVE
F.": N$: CHR$(44); "A32768": CHR$(44); "L5376"
2400 HOME : VTAB 6: HTAB 8: PRINT "PLEASE REINSERT DISK
ETTE": VTAB 7: HTAB 8: PRINT "CONTAINING ": INVERSE
: PRINT "MICRO - CALC"
2410 FOR I = 1 TO 4: PRINT CHR$(7); NEXT I
2420 INVERSE : VTAB 9: HTAB 8: INPUT "PRESS RETURN WHEN
READY"; A$: NORMAL
2430 TEXT : HOME : VTAB 12: HTAB 15: INVERSE : PRINT "S
AVING DATA": NORMAL
2440 PRINT : PRINT CHR$(4); "OPEN TEMP-FILE"
2450 PRINT CHR$(4); "WRITE TEMP-FILE"
2460 PRINT NR: PRINT NC: PRINT N$
2470 FOR I = 1 TO NC: PRINT LC(I): NEXT I
2480 FOR I = 1 TO NR: PRINT R$(I): NEXT I
2490 FOR I = 1 TO NC: PRINT CL$(I): NEXT I
2500 FOR N = 1 TO NR: FOR M = 1 TO NC: PRINT WK(N,M): NEXT
M: NEXT N
2510 PRINT CHR$(4); "CLOSE TEMP-FILE"
2520 PRINT CHR$(4); "RUN MICRO-CALC.11"
2530 TEXT : HOME : INVERSE : VTAB 4: HTAB 11: PRINT W$
; W$: W$: NORMAL : PRINT : POKE 3,1: CALL 32512: FOR
I = 1 TO NR
2540 VTAB 4 + I: HTAB 1: PRINT R$(I): NEXT I: NORMAL

2550 VTAB 1: HTAB 1: INVERSE : PRINT "(- OR ( SHIFT FIL
E ) OR -) R = RESUME ": NORMAL
2560 VTAB 4: HTAB 11: GET A$
2570 IF A$ = CHR$(8) THEN Y = Y + 10
2580 IF A$ = CHR$(21) THEN Y = Y - 10
2590 IF A$ = CHR$(68) OR A$ = CHR$(44) THEN Y = Y +
1

```

```

2600 IF A$ = CHR$(62) OR A$ = CHR$(46) THEN Y = Y -
1
2610 IF A$ = CHR$(82) THEN LZ = 1: LZ$ = B$: RETURN
2620 IF Y > 255 THEN Y = Y - 255
2630 IF Y < 0 THEN Y = 255 + Y
2640 POKE 3,Y: CALL 32512: GOTO 2560

```

```

1 REM *****
2 REM * MICRO-CALC.11 *
3 REM * BY STEVEN BOHN *
4 REM * COPYRIGHT (C) 1982 *
5 REM * BY MICRO-SPARC INC *
6 REM * LINCOLN, MA. 01773 *
7 REM *****
20 ONERR GOTO 2660
30 POKE 4,0: HIMEM: 32510
40 DIM LC(21),CL$(21),R$(21),WK(21,21),TS$(21),F$(21,2
1),CC$(100)
50 W$ = " " : REM TEN SPACES
60 PRINT : PRINT CHR$(4); "OPEN TEMP-FILE"
70 PRINT CHR$(4); "READ TEMP-FILE"
80 INPUT NR: INPUT NC: INPUT N$
90 FOR I = 1 TO NC: INPUT LC(I): NEXT I
100 FOR I = 1 TO NR: INPUT R$(I): IF LEN(R$(I)) < 1
0 THEN R$(I) = LEFT$(W$,10 - LEN(R$(I))) + R
$(I)
110 NEXT I
120 FOR I = 1 TO NC: INPUT CL$(I): IF LEN(CL$(I)) < L
C(I) THEN CL$(I) = LEFT$(W$,LC(I) - LEN(CL$(I)
)) + CL$(I)
130 NEXT I
140 FOR N = 1 TO NR: FOR M = 1 TO NC: INPUT WK(N,M): NEXT
M: NEXT N
150 PRINT CHR$(4); "CLOSE TEMP-FILE": PRINT CHR$(4);
"DELETE TEMP-FILE"
160 REM --- BUILD SAVE FILE ---
170 Z$ = CHR$(34): Y$ = Z$ + CHR$(58): LN = 10000: CC =
CC + 1: A$ = "10000": CC$(CC) = A$ + "REM --- DATA --
-": CC = CC + 1: LN = LN + 1: A$ = STR$(LN)
180 ND = 0: FOR N = 1 TO NR: FOR M = 1 TO NC: IF WK(N,M)
= 3 THEN ND = ND + 1
190 NEXT M: NEXT N
200 CC$(CC) = A$ + "NR=" + STR$(NR) + CHR$(58) + "NC
=" + STR$(NC) + CHR$(58) + "ND=" + STR$(ND) +
CHR$(58) + "N$=" + CHR$(34) + N$ + CHR$(34)

210 CC = CC + 1: LN = LN + 1: A$ = STR$(LN)
220 CC$(CC) = A$ + "R$(1)=" + Z$ + R$(1) + Y$ + "R$(2
)=" + Z$ + R$(2) + Y$ + "R$(3)=" + Z$ + R$(3) +
Y$ + "R$(4)=" + Z$ + R$(4) + Y$ + "R$(5)=" + Z$ +
R$(5) + Z$
230 CC = CC + 1: LN = LN + 1: A$ = STR$(LN)
240 CC$(CC) = A$ + "R$(6)=" + Z$ + R$(6) + Y$ + "R$(7
)=" + Z$ + R$(7) + Y$ + "R$(8)=" + Z$ + R$(8) +
Y$ + "R$(9)=" + Z$ + R$(9) + Y$ + "R$(10)=" + Z$
+ R$(10) + Z$
250 CC = CC + 1: LN = LN + 1: A$ = STR$(LN)
260 CC$(CC) = A$ + "R$(11)=" + Z$ + R$(11) + Y$ + "R$(
12)=" + Z$ + R$(12) + Y$ + "R$(13)=" + Z$ + R$(
13) + Y$ + "R$(14)=" + Z$ + R$(14) + Y$ + "R$(15
)=" + Z$ + R$(15) + Z$
270 CC = CC + 1: LN = LN + 1: A$ = STR$(LN)
280 CC$(CC) = A$ + "R$(16)=" + Z$ + R$(16) + Y$ + "R$(
17)=" + Z$ + R$(17) + Y$ + "R$(18)=" + Z$ + R$(
18) + Y$ + "R$(19)=" + Z$ + R$(19) + Y$ + "R$(20
)=" + Z$ + R$(20) + Z$
290 CC = CC + 1: LN = LN + 1: A$ = STR$(LN)
300 CC$(CC) = A$ + "CL$(1)=" + Z$ + CL$(1) + Y$ + "CL$(2
)=" + Z$ + CL$(2) + Y$ + "CL$(3)=" + Z$ + CL$(3) +
Y$ + "CL$(4)=" + Z$ + CL$(4) + Y$ + "CL$(5)=" + Z$ +
CL$(5) + Z$
310 CC = CC + 1: LN = LN + 1: A$ = STR$(LN)
320 CC$(CC) = A$ + "CL$(6)=" + Z$ + CL$(6) + Y$ + "CL$(7
)=" + Z$ + CL$(7) + Y$ + "CL$(8)=" + Z$ + CL$(8) +
Y$ + "CL$(9)=" + Z$ + CL$(9) + Y$ + "CL$(10)=" + Z$
+ CL$(10) + Z$
330 CC = CC + 1: LN = LN + 1: A$ = STR$(LN)
340 CC$(CC) = A$ + "CL$(11)=" + Z$ + CL$(11) + Y$ + "CL$(
12)=" + Z$ + CL$(12) + Y$ + "CL$(13)=" + Z$ + CL$(
13) + Y$ + "CL$(14)=" + Z$ + CL$(14) + Y$ + "CL$(15
)=" + Z$ + CL$(15) + Z$
350 CC = CC + 1: LN = LN + 1: A$ = STR$(LN)
360 CC$(CC) = A$ + "CL$(16)=" + Z$ + CL$(16) + Y$ + "CL$(
17)=" + Z$ + CL$(17) + Y$ + "CL$(18)=" + Z$ + CL$(
18) + Y$ + "CL$(19)=" + Z$ + CL$(19) + Y$ + "CL$(20
)=" + Z$ + CL$(20) + Z$
370 CC = CC + 1: LN = LN + 1: A$ = STR$(LN): CC$(CC) = A$
+ "RETURN": FX = 0
380 CC = CC + 1
390 REM --- ENTER FORMULA ---
400 X = FRE(0)
410 FOR N = 1 TO NR: FOR M = 1 TO NC: IF WK(N,M) = 2 THEN
430
420 HOME : NEXT M: NEXT N: GOTO 990
430 HOME : INVERSE : VTAB 2: HTAB 4: PRINT " EN
TER FORMULA " : VTAB 3: HTAB 4: PRINT " E
SCAPE TO ENTER ROW/COLUMN " : VTAB 4: HTAB 4: PRINT
"RETURN WHEN DONE--CTRL-V TO VIEW": NORMAL : POKE 3
4,5
440 VTAB 6: HTAB 6: PRINT "FORMULA FOR ROW "; N: ", COLU
MN "; M
450 IF DF = 1 THEN VTAB 13: HTAB 1: INVERSE : PRINT "O
LD FORMULA": NORMAL : PRINT " = "; F$(N,M)
460 VTAB 8: HTAB 12: INVERSE : PRINT CL$(M): VTAB 10: HTAB
11 - LEN(R$(N)): PRINT R$(N): NORMAL : B$ = ""
470 IF LZ = 1 THEN LZ = 0: B$ = LZ$: LZ$ = ""
480 VTAB 18: HTAB 12: PRINT B$: GET A$
490 IF A$ = CHR$(13) THEN 570
500 IF A$ = CHR$(22) THEN GOSUB 860: GOTO 430
510 IF A$ = CHR$(27) THEN 640
520 IF A$ = CHR$(8) THEN B$ = "": C$ = "": HOME : GOTO
440
530 IF A$ = CHR$(94) THEN 550

```

continued on next page

Micro-Calc (Cont.)

```

540 A = ASC (A$): IF A < 48 OR A > 62 OR A = 44 OR A =
58 OR A = 59 OR A = 61 THEN 480
550 B$ = B$ + A$: IF LEN (B$) > 200 THEN B$ = "": C$ = "
": HOME: GOTO 440
560 C$ = C$ + A$: GOTO 480
570 IF B$ = "" THEN 480
580 VTAB 16: HTAB 11: INVERSE: PRINT "IS THIS CORRECT
?": NORMAL: GET A$: IF A$ = "Y" THEN F$(N,M) = C$
: C$ = "": HOME
590 IF A$ = "Y" THEN IF FX = 1 THEN RETURN
600 IF A$ = "Y" THEN 420
610 IF A$ = "N" THEN B$ = "": C$ = B$: D$ = B$: VTAB 16: HTAB
11: PRINT W0$: W0$: VTAB 18: HTAB 12: CALL - 950: GOTO
480
620 PRINT CHR$(7): GOTO 580
630 REM -- GET ROW & COLUMN --
640 VTAB 15: HTAB 1: INVERSE: PRINT "ENTER ROW NUMBER"
: NORMAL: PRINT W0$: VTAB 15: HTAB 17: GET A$: IF
A$ = CHR$(8) THEN 850
650 IF A$ = CHR$(13) THEN PRINT CHR$(7): GOTO 640
660 D$ = A$: PRINT A$: GET A$: IF A$ = CHR$(8) THEN
850
670 IF A$ = CHR$(8) THEN D$ = "": A$ = "": GOTO 640
680 IF A$ = CHR$(13) THEN D = VAL (D$): IF D < 1 THEN
PRINT CHR$(7): GOTO 640
690 IF A$ = CHR$(13) THEN 730
700 D$ = D$ + A$: PRINT A$
710 GET A$: IF A$ = CHR$(8) THEN D$ = "": A$ = "": GOTO
640
720 IF A$ < > CHR$(13) THEN PRINT CHR$(7): GOTO
710
730 D = VAL (D$): IF D < 1 OR D > 20 THEN PRINT CHR$(
7): GOTO 640
740 VTAB 16: HTAB 1: INVERSE: PRINT "ENTER COLUMN NUMB
ER": NORMAL: PRINT W0$: VTAB 16: HTAB 20: GET A$
: IF A$ = CHR$(8) THEN 850
750 IF A$ = CHR$(13) THEN PRINT CHR$(7): GOTO 740
760 D$ = A$: PRINT A$: GET A$: IF A$ = CHR$(8) THEN
850
770 IF A$ = CHR$(8) THEN D$ = "": A$ = "": GOTO 740
780 IF A$ = CHR$(13) THEN E = VAL (D$): IF E < 1 THEN
PRINT CHR$(7): GOTO 740
790 IF A$ = CHR$(13) THEN 830
800 D$ = D$ + A$: PRINT A$
810 GET A$: IF A$ = CHR$(8) THEN D$ = "": A$ = "": GOTO
740
820 IF A$ < > CHR$(13) THEN PRINT CHR$(7): GOTO
810
830 E = VAL (D$): IF E < 1 OR E > 20 THEN PRINT CHR$(
7): GOTO 740
840 D$ = "": A$ = CHR$(9) + STR$(D) + CHR$(44) + STR$(
E) + CHR$(93): B$ = B$ + A$: E$ = CHR$(68) + CHR$(
40) + STR$(D) + CHR$(44) + STR$(E) + CHR$(
41): C$ = C$ + E$
850 VTAB 15: HTAB 1: CALL - 950: VTAB 18: HTAB 12: PRINT
B$: GOTO 480
860 TEXT: HOME: INVERSE: VTAB 4: HTAB 11: PRINT W0$:
W0$: W0$: NORMAL: PRINT: POKE 3,1: CALL 32512: FOR
I = 1 TO NR
870 VTAB 4 + I: HTAB 1: PRINT R0$(I): NEXT I: NORMAL
880 VTAB 1: HTAB 1: INVERSE: PRINT "<- OR < SHIFT FILE
" OR "<- R = RESUME": NORMAL
890 VTAB 4: HTAB 11: GET A$
900 IF A$ = CHR$(8) THEN Y = Y + 10
910 IF A$ = CHR$(21) THEN Y = Y - 10
920 IF A$ = CHR$(68) OR A$ = CHR$(44) THEN Y = Y +
1
930 IF A$ = CHR$(62) OR A$ = CHR$(46) THEN Y = Y -
1
940 IF A$ = CHR$(82) THEN LZ = 1: LZ$ = B$: RETURN
950 IF Y > 255 THEN Y = Y - 255
960 IF Y < 0 THEN Y = 255 + Y
970 POKE 3,Y: CALL 32512: GOTO 890
980 REM --- REVIEW FORMULAS ---
990 B$ = "": TEXT: HOME: VTAB 12: HTAB 1: INVERSE: PRINT
"DO YOU WISH TO REVIEW FORMULAS?": NORMAL: GET A
$: IF A$ < > "Y" THEN 1110
1000 FOR N = 1 TO NR: FOR M = 1 TO NC
1010 IF WK(N,M) = 2 THEN 1030
1020 NEXT M: NEXT N: GOTO 1110
1030 TEXT: HOME: VTAB 5: HTAB 21: PRINT "COLUMN #": M
1032 INVERSE: VTAB 6: HTAB 21: PRINT CL$(M): NORMAL:
VTAB 7: HTAB 1: PRINT "ROW #": N: "": INVERSE: PRINT
R0$(N): NORMAL
1040 VTAB 7: HTAB 21: PRINT F$(N,M)
1050 TEXT: VTAB 22: HTAB 3: PRINT "(PRESS 'Q' TO QUIT,
'C' TO CHANGE)": GET A$: IF A$ = CHR$(13) THEN
1020
1060 IF A$ = "Q" THEN FOR N = 1 TO 2: NEXT N: FOR M =
1 TO 2: NEXT M: HOME: VTAB 18: PRINT "PRESS RETURN
TO QUIT,": PRINT "ANY OTHER KEY TO REVIEW/CHANGE":
GET B$: PRINT B$: IF B$ < > CHR$(13) THEN HOME:
GOTO 1000
1070 IF B$ = CHR$(13) THEN 1230
1080 IF A$ = "C" THEN 1110
1090 PRINT CHR$(7): GOTO 1050
1100 REM --- CHANGE FORMULA ---
1110 TEXT: HOME: INVERSE: VTAB 3: HTAB 11: PRINT "HI
T (R) TO REVIEW": NORMAL: VTAB 1: HTAB 1: PRINT "D
O YOU WISH TO CHANGE A FORMULA?": GET A$: IF A$ =
"N" THEN FOR N = 1 TO 2: NEXT N: FOR M = 1 TO 2: NEXT
M: GOTO 1230
1120 IF A$ = "R" THEN 990
1130 IF A$ < > "Y" THEN 1110
1140 FX = 1: HOME: HTAB 11: PRINT "(PRESS 'Q' TO QUIT)"
: PRINT: INPUT "ENTER ROW NUMBER ->": N$
1150 IF LEFT$(N$,1) = "Q" THEN 990
1160 N = VAL (N$): IF N < 1 OR N > NR THEN 1140
1170 HOME: HTAB 11: PRINT "(PRESS 'Q' TO QUIT)": PRINT
: INPUT "ENTER COLUMN NUMBER ->": N$
1180 IF LEFT$(N$,1) = "Q" THEN 990

```

```

1190 M = VAL (N$): IF M < 1 OR M > NC THEN 1170
1200 IF WK(N,M) < > 2 THEN 1140
1210 DF = 1: GOSUB 430: FX = 0: DF = 0: GOTO 1110
1220 REM --- BUILD DATA ---
1230 HOME: VTAB 12: HTAB 13: INVERSE: PRINT "SAVING F
ORMULAS": NORMAL
1240 CC$(CC) = STR$(11000) + "REM--- FORMULAS ---": CC =
CC + 1
1250 FOR M = 1 TO NC
1260 CC = CC + 1: LN = 11000 + (100 * M): A$ = STR$(LN)
1270 CC$(CC) = A$ + "REM--- NO DATA ---"
1280 FOR N = 1 TO NR
1290 IF F$(N,M) = "" THEN NEXT N: GOTO 1330
1300 B$ = "F(" + STR$(N) + CHR$(44) + STR$(M) + CHR$(
41) + CHR$(61) + F$(N,M)
1310 IF LEN (A$) + LEN (B$) < 200 THEN A$ = A$ + B$ +
CHR$(58): CC$(CC) = A$: NEXT N: GOTO 1330
1320 CC$(CC) = A$: CC = CC + 1: LN = LN + 1: A$ = STR$(LN
) + B$ + CHR$(58): CC$(CC) = A$: NEXT N
1330 CC = CC + 1: LN = LN + 1: CC$(CC) = STR$(LN) + "RET
URN": NEXT M: FOR M = 1 TO NC: FOR N = 1 TO NR: F$(N
,M) = "": NEXT N: NEXT M: X = FRE (0): GOSUB 1350: GOTO
1920
1340 REM --- CUSTOM DATA ---
1350 HOME: FX = 0: CC = CC + 1: LN = LN + 1: ND = 0: FOR N
= 1 TO NR: FOR M = 1 TO NC: IF WK(N,M) = 3 THEN ND
= ND + 1
1360 NEXT M: NEXT N: DIM DF$(ND + 1): NE = 5
1370 TEXT: HOME: HTAB 2: INVERSE: PRINT "ENTER
CUSTOM DATA SEQUENCE": VTAB 2: HTAB 2: PRINT "Q
=QUIT R=RESTART CTRL-V=VIEW FILE"
1380 VTAB 3: HTAB 2: PRINT " (- MOVE DATA -)
": NORMAL
1390 VTAB 4: CALL - 950: NE = NE + 1: IF NE > ND + 5 THEN
1650
1400 FOR I = - 5 TO 5: IF DF$(NE + I) < > "" THEN HTAB
15: VTAB 13 + I: PRINT NE - 5 + I: "": DF$(NE + I):
1410 NEXT I: VTAB 13: HTAB 8: PRINT "ROW ->": INVERSE
: PRINT NE - 5: "": NORMAL: PRINT DF$(NE): VTAB
13: HTAB 25: PRINT "(- COLUMN"
1420 IF DF$(NE) = "" THEN 1440
1430 INVERSE: VTAB 14: HTAB 1
1435 PRINT R0$(VAL (LEFT$(DF$(NE),2))) : VTAB 14: HTAB
28: PRINT CL$(VAL (RIGHT$(DF$(NE),1)) + 10 * VAL
(MID$(DF$(NE), LEN (DF$(NE)) - 1,1)))
1440 NORMAL: VTAB 5: HTAB 1: PRINT "
": VTAB 5: HTAB 1: PRINT "ENTER ROW NUMBER
": GET A$
1450 IF A$ = "Q" THEN RETURN
1460 IF A$ = "R" THEN NE = 5: GOTO 1390
1470 IF A$ = CHR$(22) THEN GOSUB 860: NE = NE - 1: GOTO
1370
1480 IF A$ = CHR$(8) THEN NE = NE - 2: IF NE < 5 THEN
NE = 5: GOTO 1390
1490 IF A$ = CHR$(8) THEN 1390
1500 IF A$ = CHR$(21) THEN IF DF$(NE) < > "" THEN 1
390
1510 IF VAL (A$) < 1 THEN 1440
1520 PRINT A$: INPUT "": B$: B$ = A$ + B$: B = VAL (B$):
IF B < 1 OR B > 20 THEN 1440
1530 N = B
1540 VTAB 5: HTAB 1: PRINT "ENTER COLUMN NUMBER": GET
A$
1550 IF A$ = "Q" THEN RETURN
1560 IF A$ = "R" THEN NE = 5: GOTO 1390
1570 IF A$ = CHR$(22) THEN GOSUB 860: NE = NE - 1: GOTO
1370
1580 IF A$ = CHR$(8) THEN NE = NE - 2: IF NE < 5 THEN
NE = 5: GOTO 1390
1590 IF A$ = CHR$(8) THEN 1390
1600 IF A$ = CHR$(21) THEN IF DF$(NE) < > "" THEN 1
390
1610 IF VAL (A$) < 1 THEN 1440
1620 PRINT A$: INPUT "": B$: B$ = A$ + B$: B = VAL (B$):
IF B < 1 OR B > 20 THEN 1540
1630 M = B: IF WK(N,M) < > 3 THEN 1440
1640 DF$(NE) = STR$(N) + CHR$(44) + STR$(M): GOTO
1390
1650 VTAB 22: HTAB 1: PRINT "DONE, PRESS 'S' TO SAVE OR
(-) TO VIEW": NE = 5
1660 GET A$: IF A$ = "S" THEN PRINT A$: HOME: INVERSE
: VTAB 18: HTAB 5: PRINT "SAVING CUSTOM DATA SEQUEN
CE": VTAB 11: HTAB 5: PRINT "MAY TAKE SEVERAL MI
NUTES": NORMAL: GOTO 1810
1670 VTAB 3: HTAB 2: FLASH: PRINT "S=SAVE": VTAB 3: HTAB
29: PRINT "C=CHANGE": NORMAL
1680 VTAB 4: CALL - 950
1690 IF A$ = "R" THEN NE = 5: GOTO 1390
1700 IF A$ = CHR$(8) THEN NE = NE - 1: IF NE < 6 THEN
NE = ND + 5
1710 IF A$ = CHR$(44) OR A$ = CHR$(68) THEN NE = NE
- 10: IF NE < 6 THEN NE = ND
1720 IF A$ = CHR$(21) THEN NE = NE + 1: IF NE > ND +
5 THEN NE = 6
1730 IF A$ = CHR$(46) OR A$ = CHR$(62) THEN NE = NE
+ 10: IF NE > ND + 5 THEN NE = 11
1740 IF A$ = CHR$(22) THEN GOSUB 860: HOME: HTAB 2
1742 INVERSE: PRINT "ENTER CUSTOM DATA SEQUENCE
": HTAB 2: PRINT "Q=QUIT R=RESTART CTRL-V=VIEW
FILE": HTAB 2: PRINT " (- MOVE DATA -)
": NORMAL: GOTO 1670
1750 IF A$ = "C" THEN 1850
1760 FOR I = - 5 TO 5: IF DF$(NE + I) < > "" THEN HTAB
15: VTAB 13 + I: PRINT NE - 5 + I: "": DF$(NE + I):
1770 NEXT I: VTAB 13: HTAB 8: PRINT "ROW ->": INVERSE
: PRINT NE - 5: "": NORMAL: PRINT DF$(NE): VTAB
13: HTAB 25: PRINT "(- COLUMN"
1780 IF DF$(NE) = "" THEN 1800
1790 INVERSE: VTAB 14: HTAB 1
1795 PRINT R0$(VAL (LEFT$(DF$(NE),2))) : VTAB 14: HTAB
28: PRINT CL$(VAL (RIGHT$(DF$(NE),1)) + 10 * VAL
(MID$(DF$(NE), LEN (DF$(NE)) - 1,1)))
1800 NORMAL: GOTO 1660
1810 NE = 5: CD = 0: CC$(CC) = STR$(LN) + "DATA": FOR I
= 6 TO ND + 5
1820 CD = CD + 1: IF CD < 44 THEN CC$(CC) = CD$(CC) + DF
$(I) + CHR$(44): NEXT I: RETURN

```



```

1830 CC$(CC) = CC$(CC) + DF$(1):CC = CC + 1:LN = LN + 1:
    CC$(CC) = STR$(LN) + "DATA":CD = 0
1840 NEXT I: RETURN
1850 VTAB 5: HTAB 1: PRINT "ENTER STEP NUMBER, ROW, COL
    UN": INPUT "ENTER ->":A$,B$,C$:A = VAL(A$):R =
    VAL(B$):C = VAL(C$)
1860 IF A > ND THEN 1680
1870 IF R > NR THEN 1680
1880 IF C > NC THEN 1680
1890 IF WK(R,C) < 3 THEN 1680
1900 DF$(A + 5) = STR$(R) + CHR$(44) + STR$(C): GOTO
    1680
1910 REM --- SAVE TO DISK ---
1920 TEXT: HOME: NORMAL
1930 VTAB 12: HTAB 5: PRINT "NOW WRITING AND SAVING:"
1932 INVERSE: VTAB 14: HTAB 5: PRINT "NORMAL: VTAB
    16: HTAB 5: PRINT "MAY TAKE SEVERAL MINUTES"
1940 PRINT CHR$(13) + CHR$(4):"OPEN WORKER": PRINT
    CHR$(4):"DELETE WORKER"
1950 PRINT CHR$(4):"OPEN WORKER": PRINT CHR$(4):"WR
    ITE WORKER"
1960 PRINT "LOAD SCRATCH-PAD"
1970 PRINT "TEXT"
1980 FOR I = 1 TO CC: PRINT CC$(I): NEXT I: PRINT "HOME
    "
1990 PRINT "PRINT" + CHR$(34) + "PLEASE INSERT THE D1
    SKETTE THAT YOU ARE SAVING YOUR PROGRAM ONTO" + CHR$(
    34)
2000 PRINT "PRINT" + CHR$(34) + "PRESS THE >>>>SPACE
    BAR<<<< WHEN READY" + CHR$(34): PRINT "CALL 326
    61"
2010 PRINT "SAVE":N$:
2020 PRINT "RUN"
2030 PRINT CHR$(4):"CLOSE WORKER"
2040 PRINT CHR$(4):"EXEC WORKER"
2050 END
2060 A = PEEK(222):B = PEEK(218) + PEEK(219) * 256
    : IF A = 9 THEN HOME: PRINT CHR$(7): VTAB 18: FLASH
    : PRINT "DISK FULL - PLEASE USE ANOTHER": NORMAL: FOR
    I = 1 TO 1000: GOTO 1520
2070 HOME: VTAB 18: PRINT "ERROR NUMBER":A: PRINT "IN
    LINE NUMBER":B: END
}
10 REM SCRATCH-PAD
    BY STEPHEN BOHN
20 POKE 3,0: POKE 4,0: HIMEM: 32510
30 ONERR GOTO 130
40 TEXT: HOME
50 DIM F(21,21),D(21,21),CL$(21),RO$(21),A$(30)
60 GOSUB 10000
70 PRINT CHR$(4):"BLOAD F.":N$:
80 BL$(1) = "00":BL$(2) = BL$(1):BL$(3) = BL$(1):BL$(4) =
    "000"
90 BL$(5) = "00":BL$(6) = "00":BL$(7) = "00":BL$(8) = "00":BL
    $(9) = "00":BL$(10) = "00"
100 TEXT
110 PRINT CHR$(4):"BLOAD DISPLAYER"
120 GOTO 2500
130 REM --- HANDLE ERRORS ---
140 QX = PEEK(222)
150 IF QX = 42 THEN RESTORE: GOTO 2500
160 IF QX = 6 THEN GOSUB 1370: HOME: FLASH: PRINT "F
    ILE NOT FOUND": NORMAL: GOTO 1220
170 IF QX = 9 THEN HOME: INVERSE: VTAB 18: PRINT "D1
    SK FULL - PLEASE USE ANOTHER": NORMAL: PRINT CHR$(
    7): FOR I = 1 TO 1000: NEXT I: GOTO 2500
180 IF QX = 133 THEN TEXT: HOME: PRINT "DIVISION BY
    ZERO": PRINT "PROBLEM WITH FORMULA": FOR I = 1 TO
    8000: NEXT I: CLEAR: RESTORE: GOTO 30
190 IF QX = 254 THEN GOTO 1350
200 IF QX = 5 THEN 1350
210 TEXT: HOME: PRINT "PROBLEM IN LINE #": PRINT PEEK
    (218) + PEEK(219) * 256
220 END
230 REM --- GET DATA VIA CUSTOM ---
240 IC = IC + 1:B$ = "": READ R: READ C
250 HOME: PRINT "ENTER DATA FOR ROW":R,"COLUMN":C
260 GOSUB 2010: INVERSE: HTAB 1: VTAB 4 + R: PRINT RO$(
    R): NORMAL
270 GOSUB 1920: INVERSE: VTAB 3: HTAB 12: PRINT CL$(C)
    : NORMAL
280 GOSUB 660
290 IF A$ = "Q" THEN RESTORE: IC = 0: RETURN
300 IF IC > ND THEN RESTORE: IC = 0: RETURN
310 GOTO 240
320 REM --- GET DATA VIA ROW ---
330 FOR R = 1 TO NR: FOR C = 1 TO NC
340 HOME: PRINT "ENTER DATA FOR ROW":R,"COLUMN":C:
    B$ = ""
350 GOSUB 1990: IF PEEK(32768 + (R + 256) + LL) < )
    164 THEN 400
360 GOSUB 2010: INVERSE: HTAB 1: VTAB 4 + R: PRINT RO$(
    R): NORMAL
370 GOSUB 1920: INVERSE: VTAB 3: HTAB 12: PRINT CL$(C)
    : NORMAL
380 GOSUB 660
390 IF A$ = "Q" THEN RETURN
400 NEXT C: NEXT R: RETURN
410 REM --- GET DATA VIA COLUMN ---
420 FOR C = 1 TO NC: FOR R = 1 TO NR
430 HOME: PRINT "ENTER DATA FOR ROW":R,"COLUMN":C:
    B$ = ""
440 GOSUB 1990: IF PEEK(32768 + (R + 256) + LL) < )
    164 THEN 400
450 GOSUB 2010: INVERSE: HTAB 1: VTAB 4 + R: PRINT RO$(
    R): NORMAL
460 GOSUB 1920: INVERSE: VTAB 3: HTAB 12: PRINT CL$(C)
    : NORMAL
470 GOSUB 660
480 IF A$ = "Q" THEN RETURN
490 NEXT R: NEXT C: RETURN
500 REM --- ADD/CHANGE DATA ---
510 HOME: VTAB 1: HTAB 11: INVERSE: PRINT "ADD OR CHA
    NGE DATA": HTAB 11: PRINT "(PRESS 'Q' TO QUIT)": NORMAL
520 VTAB 3: HTAB 1: CALL - 950: PRINT "ENTER ROW NUMBE
    R ->": GET A$: PRINT A$: IF A$ = "Q" THEN RETURN
530 IF A$ = CHR$(13) THEN 520
540 INPUT "":B$:A$ = A$ + B$
550 R = VAL(A$): IF R < 1 OR R > 20 THEN 520

```

```

560 VTAB 3: HTAB 1: CALL - 950: PRINT "ENTER COLUMN NU
    MBER ->": GET A$: PRINT A$: IF A$ = "Q" THEN RETURN
570 IF A$ = CHR$(13) THEN 560
580 INPUT "":B$:A$ = A$ + B$
590 C = VAL(A$): IF C < 1 OR C > 20 THEN 560
600 GOSUB 2010: INVERSE: HTAB 1: VTAB 4 + R: PRINT RO$(
    R): NORMAL
610 GOSUB 1920: INVERSE: VTAB 3: HTAB 12: PRINT CL$(C)
    : NORMAL
620 B$ = BL$(LEN(CL$(C))): GOSUB 2040: INVERSE: VTAB
    3: HTAB 12: PRINT CL$(C): NORMAL
630 VTAB 1: HTAB 11: PRINT "ENTER DATA FOR ROW":R,"C
    OLUMN":C:B$ = "": PRINT " (PRESS 'Q' TO
    QUIT)"
640 VTAB 3: HTAB 1: PRINT " ": GOSUB 660: IF
    A$ = "Q" THEN RETURN
650 GOTO 510
660 REM --- GET DATA ---
670 B$ = "": VTAB 2: HTAB 11: PRINT "(PRESS 'Q' TO QUIT)
    "
680 VTAB 4 + R: HTAB 11 + LEN(CL$(C)): GET A$
690 IF A$ = CHR$(13) THEN D(R,C) = VAL(B$) / 100: RETURN
700 IF A$ = CHR$(83) THEN RETURN
710 IF A$ = CHR$(67) THEN B$ = "S" + BL$(LEN(CL$(C)
    ) - 1): GOSUB 2040: INVERSE: VTAB 3: HTAB 12: PRINT
    CL$(C): NORMAL: GOTO 670
720 IF A$ = CHR$(8) THEN B$ = "S" + BL$(LEN(CL$(C)
    ) - 1): GOSUB 2040: INVERSE: VTAB 3: HTAB 12: PRINT
    CL$(C): NORMAL: GOTO 670
730 IF A$ = "Q" THEN RETURN
740 IF A$ = CHR$(46) THEN 840
750 IF A$ = CHR$(45) THEN 770
760 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN PRINT CHR$(
    7): GOTO 600
770 B$ = B$ + A$: GOSUB 2040: INVERSE: VTAB 3: HTAB 12:
    PRINT CL$(C): NORMAL
780 IF LEN(B$) < LEN(CL$(C)) - 1 THEN 600
790 VTAB 4 + R: HTAB 11 + LEN(CL$(C)): GET A$
800 IF A$ = CHR$(67) OR A$ = CHR$(8) THEN B$ = "S" +
    BL$(LEN(CL$(C)) - 1): GOSUB 2040: INVERSE: VTAB
    3: HTAB 12: PRINT CL$(C): NORMAL: GOTO 670
810 IF A$ < > CHR$(13) THEN PRINT CHR$(7): GOTO
    790
820 D(R,C) = VAL(B$) / 100: RETURN
830 GOTO 600
840 VTAB 4 + R: HTAB 11 + LEN(CL$(C)): GET A$
850 IF A$ = CHR$(13) THEN D(R,C) = VAL(B$):B$ = B$ +
    "00": GOSUB 2040: RETURN
860 IF A$ = CHR$(83) THEN RETURN
870 IF A$ = CHR$(67) THEN B$ = "S" + BL$(LEN(CL$(C)
    ) - 1): GOSUB 2040: INVERSE: VTAB 3: HTAB 12: PRINT
    CL$(C): NORMAL: GOTO 670
880 IF A$ = CHR$(8) THEN B$ = "S" + BL$(LEN(CL$(C)
    ) - 1): GOSUB 2040: INVERSE: VTAB 3: HTAB 12: PRINT
    CL$(C): NORMAL: GOTO 670
890 IF A$ = CHR$(81) THEN RETURN
900 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN PRINT CHR$(
    7): GOTO 840
910 B$ = B$ + A$: GOSUB 2040: INVERSE: VTAB 3: HTAB 12:
    PRINT CL$(C): NORMAL
920 IF LEN(B$) < LEN(CL$(C)) - 1 THEN PRINT CHR$(
    7): GOTO 670
930 VTAB 4 + R: HTAB 11 + LEN(CL$(C)): GET A$
940 IF A$ = CHR$(13) THEN D(R,C) = VAL(B$) / 100:B$ =
    B$ + "0": GOSUB 2040: RETURN
950 IF A$ = CHR$(83) THEN RETURN
960 IF A$ = CHR$(67) THEN B$ = "S" + BL$(LEN(CL$(C)
    ) - 1): GOSUB 2040: INVERSE: VTAB 3: HTAB 12: PRINT
    CL$(C): NORMAL: GOTO 670
970 IF A$ = CHR$(8) THEN B$ = "S" + BL$(LEN(CL$(C)
    ) - 1): GOSUB 2040: INVERSE: VTAB 3: HTAB 12: PRINT
    CL$(C): NORMAL: GOTO 670
980 IF A$ = CHR$(81) THEN RETURN
990 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN PRINT CHR$(
    7): GOTO 930
1000 B$ = B$ + A$: GOSUB 2040: INVERSE: VTAB 3: HTAB 12
    : PRINT CL$(C): NORMAL
1010 D(R,C) = VAL(B$) / 100
1020 VTAB 4 + R: HTAB 11 + LEN(CL$(C)): GET A$
1030 IF A$ = CHR$(67) OR A$ = CHR$(8) THEN B$ = "S" +
    BL$(LEN(CL$(C)) - 1): GOSUB 2040: INVERSE: VTAB
    3: HTAB 12: PRINT CL$(C): NORMAL: GOTO 670
1040 IF A$ < > CHR$(13) THEN PRINT CHR$(7): GOTO
    1020
1050 RETURN
1060 REM --- SAVE FILE ---
1070 TEXT: HOME:MD$ = ""
1080 PRINT "INSERT DISK TO SAVE DATA TO": PRINT: PRINT
    "DO YOU WISH TO USE A SPECIAL": PRINT "NAME MODIFIER
    Y/N?": GET A$: IF A$ = "N" THEN 1120
1090 IF A$ < > "Y" THEN 1070
1100 PRINT: INPUT "ENTER NAME MODIFIER ->":MD$:
1110 IF LEN(MD$) > 10 THEN PRINT: INVERSE: PRINT "
    PLEASE USE A MODIFIER OF 10 CHARACTERS": PRINT "OR L
    ESS": VTAB 5: HTAB 1:
    CALL - 860: GOTO 1100
1120 PRINT: PRINT CHR$(4):"BSAVE E.":N$:MD$: CHR$(
    44):"A32768": CHR$(44):"L5376"
1130 FOR R = 1 TO NR: FOR C = 1 TO NC
1140 IF D(R,C) < 8 THEN II = II + 1
1150 NEXT C: NEXT R
1160 PRINT CHR$(4):"OPEN D.":N$:MD$: PRINT CHR$(
    4):"WRITE D.":N$:MD$: PRINT II
1170 FOR R = 1 TO NR: FOR C = 1 TO NC
1180 IF D(R,C) < 8 THEN PRINT R: PRINT C: PRINT D(R
    ,C)
1190 NEXT C: NEXT R: PRINT CHR$(4):"CLOSE D.":N$:MD$: M
    D$: RETURN
1200 REM --- LOAD FILE ---
1210 NF = 0: TEXT: HOME:MD$ = ""
1220 VTAB 18: PRINT "IF YOU WOULD LIKE A LISTING OF FIL
    ES": PRINT "AVAILABLE, PRESS THE 'L' KEY": GET A$:
    PRINT A$: IF A$ < > "L" THEN 1260
1230 HOME: VTAB 3: PRINT "INSERT DISK WITH FILE(S) TO
    BE LOADED": INPUT "PRESS RETURN WHEN READY":A$: GOSUB
    1370
1240 REM NF = NO FILES
1250 IF NF = 1 THEN 2500
1260 VTAB 24: HTAB 11: PRINT "(PRESS 'Q' TO QUIT)": HTAB
    1: VTAB 19: PRINT "INSERT DISK WITH FILE TO BE LOA
    D": PRINT: PRINT "IS THERE A SPECIAL NAME MODIFIE
    R Y/N?": GET A$: IF A$ = "N" THEN 1320
1270 IF A$ = "Q" THEN 2500
1280 IF A$ < > "Y" THEN 1260
1290 PRINT: INPUT "ENTER SPECIAL MODIFIER ->":MD$:
1300 IF LEN(MD$) > 10 THEN MD$ = LEFT$(MD$,10)

```

continued on next page

Micro-Calc (Cont.)

```

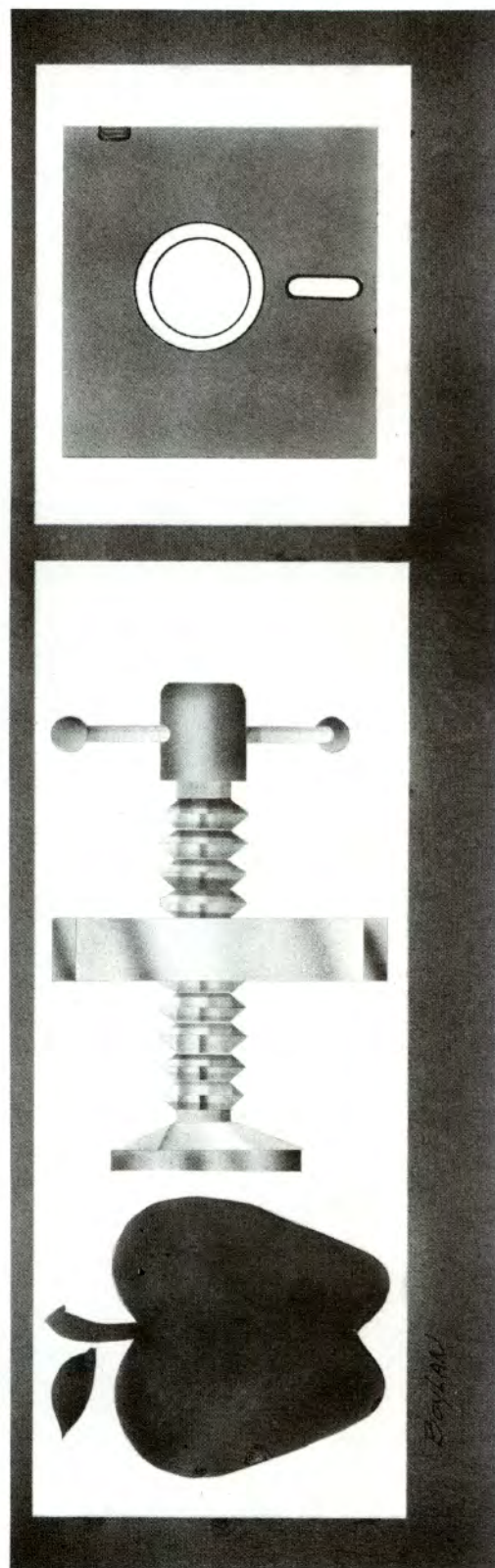
1310 IF NF = 1 THEN 2500
1320 PRINT : PRINT CHR$(4); "BLOAD E.": NF = MD%
1330 PRINT CHR$(4); "OPEN D.": NF = MD%: PRINT CHR$(4); "READ D.": NF = MD%: INPUT I
1340 FOR I = 1 TO 11: INPUT R: INPUT C: INPUT D(R,C): NEXT I
1350 PRINT CHR$(4); "CLOSE D.": NF = MD%: GOTO 2140
1360 REM --- FILE ERROR ---
1370 NF = 0: FOR Z2 = 0 TO 10: IF Z2 = 0 THEN 1380: NEXT Z2
1380 TEXT : HOME : PRINT : PRINT CHR$(4); "CATALOG": INVERSE : PRINT "DO NOT ENTER ANY COMMANDS YET": NORMAL
1390 IF PEEK(1031) = 197 THEN I = 1031: A = 1: GOTO 14
1400 IF PEEK(1159) = 197 THEN I = 1159: A = 2: GOTO 14
1410 IF PEEK(1287) = 197 THEN I = 1287: A = 3: GOTO 14
1420 IF PEEK(1415) = 197 THEN I = 1415: A = 4: GOTO 14
1430 IF PEEK(1543) = 197 THEN I = 1543: A = 5: GOTO 14
1440 IF PEEK(1671) = 197 THEN I = 1671: A = 6: GOTO 14
1450 IF PEEK(1799) = 197 THEN I = 1799: A = 7: GOTO 14
1460 IF PEEK(1927) = 197 THEN I = 1927: A = 8: GOTO 14
1470 IF PEEK(1071) = 197 THEN I = 1071: A = 9: GOTO 14
1480 IF PEEK(1199) = 197 THEN I = 1199: A = 10: GOTO 14
1490 IF PEEK(1327) = 197 THEN I = 1327: A = 11: GOTO 14
1500 IF PEEK(1455) = 197 THEN I = 1455: A = 12: GOTO 14
1510 IF PEEK(1583) = 197 THEN I = 1583: A = 13: GOTO 14
1520 IF PEEK(1711) = 197 THEN I = 1711: A = 14: GOTO 14
1530 IF PEEK(1839) = 197 THEN I = 1839: A = 15: GOTO 14
1540 IF PEEK(1967) = 197 THEN I = 1967: A = 16: GOTO 14
1550 IF PEEK(1111) = 197 THEN I = 1111: A = 17: GOTO 14
1560 IF PEEK(1239) = 197 THEN I = 1239: A = 18: GOTO 14
1570 IF PEEK(1367) = 197 THEN I = 1367: A = 19: GOTO 14
1580 IF PEEK(1495) = 197 THEN I = 1495: A = 20: GOTO 14
1590 IF PEEK(1623) = 197 THEN I = 1623: A = 21: GOTO 14
1600 IF PEEK(1751) = 197 THEN I = 1751: A = 22: GOTO 14
1610 IF PEEK(1879) = 197 THEN I = 1879: A = 23: GOTO 14
1620 IF PEEK(2007) = 197 THEN I = 2007: A = 24: GOTO 14
1630 A = 24
1640 FOR J = 1 TO LEN(NF)
1650 IF PEEK(I + J + 1) < (ASC(MID$(NF, J, 1)) + 128) THEN 1710
1660 NEXT J
1670 FOR J = 1 TO 10
1680 A$(N) = A$(N) + CHR$(PEEK(I + J + LEN(NF) + 1)) - 128
1690 NEXT J
1700 N = N + 1
1710 FOR J = 1 TO 2: NEXT J
1720 IF A = 24 AND N = 8 THEN 1800
1730 IF A = 24 THEN 1790
1740 IF A > 12 THEN 1770
1750 ON A GOTO 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500, 1510
1760 PRINT "A = "A": IN LINE 1540 ?": END
1770 A = 12: ON A GOTO 1520, 1530, 1540, 1550, 1560, 1570, 1580, 1590, 1600, 1610, 1620
1780 GOTO 1800
1790 HOME : PRINT "FILES AVAILABLE FOR "NF": ARE:"
1800 PRINT : INVERSE
1810 FOR J = 0 TO N - 2
1820 PRINT
1830 HTAB 10: PRINT A$(J):
1840 IF A$(J) = " " THEN NORMAL : PRINT "NO MODIFIER": INVERSE
1850 NEXT J
1860 PRINT
1870 NORMAL : NF = 0: RETURN
1880 HOME : INVERSE : VTAB 12: PRINT "NO FILES AVAILABLE E": NORMAL
1890 NF = 1
1900 FOR J = 1 TO 4: PRINT CHR$(7): NEXT J: FOR J = 1 TO 2000: NEXT J
1910 RETURN
1920 REM --- DISPLAY FILE ROUTINE ---
1930 LL = 1
1940 IF C = 1 THEN LL = 2: GOTO 1900
1950 FOR I1 = 0 TO C - 1
1960 LL = LL + LEN(CHR$(I1)) + 1
1970 NEXT I1
1980 POKE 3, LL: CALL 32512: RETURN
1990 LL = 1: IF C = 1 THEN LL = 2: RETURN
2000 FOR I1 = 0 TO C - 1: LL = LL + LEN(CHR$(I1)) + 1: NEXT I1: RETURN
2010 REM --- PRINT ROW ROUTINE ---
2020 FOR I = 1 TO NR: HTAB 1: VTAB 4 + I: PRINT ROW$(I):
2030 NEXT I: RETURN
2040 REM --- DISPLAY DATA VALUES ---
2050 IF B% = "" THEN RETURN
2060 B = LEN(B%)
2070 FOR J = 1 TO B: PP = 32768 + (R * 256) + LL + LEN(CHR$(C)) - J
2080 IF B = 1 THEN POKE PP, 176: POKE PP - 1, 176: POKE PP - 2, 176
2090 IF J = 1 THEN POKE PP, ASC(RIGHT$(B%, 1)) + 128: GOTO 2130
2100 IF J = 2 THEN POKE PP, ASC(MID$(B%, 1 + B - J, 1)) + 128: GOTO 2130
2110 IF J = 3 THEN POKE PP, 174
2120 POKE PP - 1, ASC(MID$(B%, 1 + B - J, 1)) + 128
2130 CALL 32512: NEXT J: RETURN
2140 REM --- FETCH FORMULA ---
2150 HOME : VTAB 12: HTAB 0: INVERSE : PRINT "CALCULATING FORMULAS": VTAB 13: HTAB 0: PRINT "MAY TAKE SEVERAL MINUTES": NORMAL
2160 FOR K = 1 TO NR: FOR M = 1 TO NC
2170 ON M GOSUB 1100, 11200, 11300, 11400, 11500, 11600, 11700, 11800, 11900, 12000, 12100, 12200, 12300, 12400, 12500, 12600, 12700, 12800, 12900, 13000, 13100

```

```

2180 FOR N = 1 TO NR: IF F(N,M) < 0 THEN B = F(N,M): B = INT((B + .005000005) * 100): B% = STR$(B): DO N, M) = B / 100
2190 NEXT M: NEXT N: NEXT K
2200 HOME : GOSUB 2010
2210 FOR M = 1 TO NC
2220 ON M GOSUB 1100, 11200, 11300, 11400, 11500, 11600, 11700, 11800, 11900, 12000, 12100, 12200, 12300, 12400, 12500, 12600, 12700, 12800, 12900, 13000, 13100
2230 C = M
2240 FOR R = 1 TO NR: IF F(R,C) = 0 THEN 2310
2250 B = F(R,C): B = INT((B + .005000005) * 100): B% = STR$(B): F(R,C) = B / 100
2260 IF B > 1 * 10 ^ (LEN(CHR$(C)) - 1) - 1 THEN B% = LEFT$( "ERROR", LEN(CHR$(C)) - 1)
2270 GOSUB 1930
2280 C% = B%: B% = LEFT$( " ", LEN(CHR$(C)) - 1): GOSUB 2040: B% = C%
2290 GOSUB 2050
2300 D(R,C) = F(R,C)
2310 NEXT R
2320 NEXT M
2330 REM --- DISPLAY DATA ---
2340 TEXT : HOME : VTAB 1: HTAB 14: INVERSE : PRINT "DISPLAY FILE": FOR I = 1 TO NR: HTAB 1: VTAB 4 + I: PRINT ROW$(I): NEXT I
2350 VTAB 2: HTAB 2: PRINT "(P)RINT (C)HANGE (S)AVE (E)XTER (O)UIT"
2360 NORMAL : POKE 4, 0
2370 Y = 1: POKE 3, Y: CALL 32512
2380 VTAB 1: HTAB 41: GET A%
2390 VTAB 4: HTAB 1: CALL - 868
2400 IF A% = CHR$(81) THEN HOME : END : REM "Q"
2410 IF A% = CHR$(67) THEN GOSUB 510: GOSUB 2150: GOTO 2330: REM "C"
2420 IF A% = CHR$(83) THEN GOSUB 1070: GOTO 2330: REM "S"
2430 IF A% = CHR$(69) THEN POKE 4, 0: GOTO 2500: REM "E"
2440 IF A% = CHR$(80) THEN 2670: REM "P"
2450 IF A% = CHR$(8) THEN Y = Y + 10: IF Y > 255 THEN Y = Y - 255
2460 IF A% = CHR$(21) THEN Y = Y - 10: IF Y < 0 THEN Y = 255 + Y
2470 IF A% = CHR$(68) THEN Y = Y + 1: IF Y > 255 THEN Y = Y - 255
2480 IF A% = CHR$(44) THEN Y = Y + 1: IF Y > 255 THEN Y = Y - 255
2490 IF A% = CHR$(46) THEN Y = Y - 1: IF Y < 0 THEN Y = 255 + Y
2500 IF A% = CHR$(62) THEN Y = Y - 1: IF Y < 0 THEN Y = 255 + Y
2510 IF A% = CHR$(45) THEN STAR = STAR - 1: IF STAR < 0 THEN STAR = 0
2520 IF A% = CHR$(43) THEN STAR = STAR + 1: IF STAR > 27 THEN STAR = 27
2530 IF A% = CHR$(59) THEN STAR = STAR + 1: IF STAR > 27 THEN STAR = 27
2540 IF STAR = 0 THEN VTAB 4: HTAB 11 + STAR: INVERSE : PRINT CHR$(42): NORMAL
2550 IF Y + STAR > 255 THEN POKE 3, Y + STAR - 255: POKE 4, STAR: CALL 32512: GOTO 2380
2560 POKE 3, Y + STAR: POKE 4, STAR: CALL 32512
2570 GOTO 2380
2580 HOME : VTAB 4: HTAB 15: PRINT "ENTER DATA": VTAB 6: HTAB 15: PRINT "1) CUSTOM": VTAB 7: HTAB 15: PRINT "2) BY ROW": VTAB 8: HTAB 15: PRINT "3) BY COLUMN": VTAB 9: HTAB 15: PRINT "4) RANDOM"
2590 VTAB 10: HTAB 15: PRINT "5) LOAD FILE": VTAB 11: HTAB 15: PRINT "Q) TO QUIT"
2600 VTAB 13: HTAB 0: PRINT " ": VTAB 13: HTAB 0: INVERSE : PRINT "WHICH ?": NORMAL : GET A%
2610 IF A% = "Q" THEN 2140
2620 A = VAL(A%): IF A < 1 OR A > 5 THEN 2600
2630 IF A = 5 THEN 1210
2640 ON A GOSUB 230, 320, 410, 500
2650 GOTO 2140
2660 REM --- PRINT OUT ---
2670 TEXT : HOME : VTAB 5: INPUT "ENTER STARTING COLUMN NUMBER ->": A%: S = VAL(A%): IF S < 1 OR S > NC THEN 2670
2680 VTAB 6: CALL - 950: VTAB 7: HTAB 1: INPUT "ENTER ENDING COLUMN NUMBER ->": A%: E = VAL(A%): IF E < S OR E > NC THEN 2600
2690 VTAB 8: CALL - 950: VTAB 9: HTAB 1: PRINT "ENTER STARTING ": INVERSE : PRINT "ROW": NORMAL : INPUT "NUMBER ->": A%: SY = VAL(A%): IF SY < 1 OR SY > N R THEN 2690
2700 VTAB 10: CALL - 950: VTAB 11: HTAB 1: PRINT "ENTER ENDING ": INVERSE : PRINT "ROW": NORMAL : INPUT "NUMBER ->": A%: EY = VAL(A%): IF EY < SY OR EY > NR THEN 2700
2710 SX = S: EX = E: HOME : VTAB 5: PRINT "STARTING COLUMN NUMBER = "SX: PRINT "ENDING COLUMN NUMBER = "EX: PRINT "STARTING ROW = "SY: INVERSE : PRINT "ROW": NORMAL : PRINT "NUMBER = "SY
2720 PRINT : PRINT "ENDING ": INVERSE : PRINT "ROW": NORMAL : PRINT "NUMBER = "EY
2730 VTAB 14: CALL - 950: VTAB 14: HTAB 1: PRINT "IS THIS CORRECT ?": GET A%: PRINT A%: IF A% < "Y" THEN 2330
2740 PRINT : PRINT CHR$(4); "PR#1"
2750 TS = SX
2760 TE = TS - 1: LL = 0: FOR I = TS TO EX
2770 IF TE < 1 - 1 THEN 2790
2780 IF LL + LEN(CHR$(I)) < 68 THEN LL = LL + 1 + LEN(CHR$(I)): TE = I
2790 NEXT I
2800 PRINT " ": TC = 0: GOSUB 2050: PRINT
2810 PRINT " ": FOR I = 1 TO LL + 1: PRINT CHR$(61): NEXT I: PRINT
2820 FOR TC = SY TO EY: PRINT ROW$(TC): GOSUB 2050: PRINT : NEXT TC
2830 IF TE > EX THEN PRINT : PRINT CHR$(4); "PR#2"
2840 HOME : GOTO 2330
2840 TS = TE + 1: PRINT : GOTO 2760
2850 TL = - 1: FOR K = 0 TO TS - 1: TL = TL + 1 + LEN(CHR$(K)): NEXT K
2860 TT = 32769 + (TC * 256) + TL
2870 FOR J = TT TO TT + LL
2880 IF PEEK(J) = 184 THEN PRINT CHR$(124): NEXT J: RETURN
2890 PRINT CHR$(PEEK(J) - 128): NEXT J: RETURN

```



APPLESOFT VARIABLE CRUNCHER

Applesoft Variable Cruncher

by Alan and Valerie Floeter
4333 N. 71 St.
Milwaukee, WI 53216

Have you ever run out of memory when writing a large program, or does your program run too slowly? There are several approaches to help solve these problems. One of these is to use as many single letter variables as possible. Since each letter in a variable name takes one byte of memory, some savings can be gained by reducing the lengths of the variables. And making your programs smaller may reduce string "garbage collection", speeding execution dramatically.

Unfortunately, it can be a major task to rename variables to shorter names. This process is also very prone to error. You need to find all occurrences of a variable and change it to a variable name that is not already in use. Not only is this tedious work, but just doing all of the editing is reason enough to avoid using this method of optimizing an Applesoft program.

However, we can let the Apple do the tedious work. Computers are very useful for doing repetitive tasks. The Applesoft Variable Cruncher is a utility written to help you shorten programs, eliminating the need for doing extensive manual editing.

VARIABLES

In Applesoft, the first two letters of a variable name must be unique. But a programmer may use variable names of any length. For example, the variable name **ZIPCODE** is interpreted by Applesoft as **ZI**, even though the programmer uses seven letters to identify it.

Using longer variable names has its advantages and disadvantages. The word **ZIPCODE** means a lot more to someone reading a program than just **ZI** or even **Z**. A program is much more self-documenting when the variable names are related to the actual contents they hold.

Unfortunately, problems involving similar variable names are sometimes hard to isolate. The variable names **OLDX** and **OLDY** will be interpreted as the same variable by Applesoft. The **longer variable names also take more memory**. As mentioned previously, each letter occupies one byte of memory. So if the variable name **ZIPCODE** is used five times in a program, 35 bytes are occupied, as compared to only 5 bytes if **Z** is used instead.

CRUNCHER STRUCTURE

If you commonly use long variable names, the Variable Cruncher could be a worthwhile addition to your collection of utilities. The program does just as its name suggests. It makes **four passes** through an Applesoft program. The **first pass** is to detect which single variable names are already in use. Twenty-six bytes in memory, one for each letter, are marked when a single letter is used for a variable name. Of course, **there are three groups of 26 bytes for each of the types of variables — floating point, integer, and string**. In the assembly language, the symbol **TYPTAB** holds this information.

Applesoft Variable Cruncher
by Al and Valerie Floeter
The Software Experience

Copyright (c) 1982 by
Micro-SPARC, Inc.
Lincoln, MA 01773

JULY 13, 1981

THIS ROUTINE REDUCES APPLESOFT VARIABLES
TO AT LEAST TWO CHARACTERS, PLUS ATTEMPTS
TO RENAME TO ONLY ONE CHARACTER.

THIS IS DONE IN FOUR PHASES.

PHASE I
MARKS ALL SINGLE CHARACTER VARIABLES

PHASE II
SHORTENS LONG VARIABLE NAMES TO 2 CHARACTERS

PHASE III
ATTEMPTS TO REDUCE TO FIRST LETTER

PHASE IV
ASSIGNS SINGLE LETTER VARIABLE NAMES TO
REMAINING VARIABLES, WHEN POSSIBLE

DOUT EQU \$FDED
CR EQU \$FDBE
IND EQU \$FC
PIND EQU \$FE
YBAV EQU \$FB
BEGN EQU \$67
PEND EQU \$AF
LOM EQU \$69

DSC \$19
PHASE DFS 1
JDIR DFS 2
TYPE DFS 1
YPOS DFS 1
SAVEX DFS 1

ORG \$6000

LDA \$AA59 ;FOR BRUN FROM DOS

PHA

LDA \$6A59

PHA

LDA \$0

STA THFLAG

STA PHASE

LDX \$26F3-1

STA TYPTAB,X

DEX

BPL LOOP

NEXTPH JBR CR

LDX \$PPHASE-ARROW

JBR M3

LDA PHASE

CLC

ADC \$*1

JBR DOUT

JBR CR

JBR CR

LDA PHASE

ASL

TAX

LDA JTAB,X

STA JDIR

INX

LDA JTAB,X

STA JDIR+1

LDA BEGN

STA IND

LDA BEGN+1

STA IND+1

NEXTLN LDY \$1

LDA (IND),Y

BEQ QUITIT

LDY \$4

CAGAIN JBR JBRRDIR

LDY \$0

LDA (IND),Y

PHA

INY

LDA (IND),Y

STA IND+1

PLA

STA IND

JMP NEXTLN

LDA THFLAG

BNE PHASN

INC PHASE

LDA PHASE

CHP \$PHNUM

BCC NEXTPH

PLA

STA \$6A59

PLA

STA \$AA59

RTS

JTAB

DFC PHASE1,PHASE1/256

DFC PHASE2,PHASE2/256

DFC PHASE3,PHASE3/256

continued on next page

Applesoft Variable Cruncher (Cont.)

The **second pass** reduces any variable names longer than two characters to exactly two characters. Of course, any special symbol is also carried with the variable name. So this means the variable name ZIPCODE would now be changed to ZI.

The **third pass** attempts to reduce the two character variables one step further. Using the list of single variable names created in the first phase, the two character variables are checked to see if they can be shortened to one character. If the first letter is already used as a variable name, it remains the same. For example, if the variable ZI% is in the program, the Variable Cruncher checks to see if the variable ZI% has already been used. This is simply done by checking to see if it is in TYPTAB. If Z% is already in use, ZI% is left intact. However, if Z% is not being used, all occurrences of ZI% would be changed to Z%, and TYPTAB would be updated to show that the variable name Z% is taken.

The **fourth** and final pass tries to rename all the remaining two character variables to any single letter variable name that is available. So if a variable name ZI% is still found in the Applesoft program at this point, TYPTAB is searched for the first single character variable name available. If F% is the first one found free in TYPTAB, all ZI%'s would be renamed to F%. Of course, if all the single letter integer variable names have already been used, then ZI% is left unchanged.

SYSTEM REQUIREMENTS

The Variable Compressor is written in assembly language and can be run as is on 48K systems, either with disk or cassette, or 32K cassette systems. To run on any 16K system or 32K disk system, the program should be re-ORG'd at \$3800.

ENTERING THE PROGRAM

To enter the program, type in either the hex codes (if you don't have an assembler) or instructions from the source listing. NOTE: At the symbol JTAB and at the end of the program two pseudo-ops are used that will not be accepted by the mini-assembler and possibly other assemblers as well. **ASC** means to store the **ASCII** values of the string that follows it. **DFC** means to store the **hex** values following it directly into memory. **You must use the monitor to enter these bytes directly into memory**, unless you have an assembler with similar pseudo-ops. **But also be aware that this is data, not executable instructions, so that if you try to disassemble that part of the program, you may find "???"s and other assorted nonsense.** Do not let this bother you as long as the hex codes match those given in the listing.

To save the program:

BSAVE VARCOMP,A\$6000,L\$372 (for disk)

OR

6000.637FW (for cassette).

To run the program:

1) Load in Applesoft program to be compressed.

```

106 607B 31 61 DFC PHASE4,PHASE4/256
107 PHNUM EQU .-JTAB/2
108 607D 6C 1A 00 JBRDIR JMP (JDIR)
109
110
111 6080 20 73 62 PHASE1 JSR GTVR ;GET VARIABLE
112 6083 90 0E BCC NOVAR ;NONE IN THAT LINE
113 6085 E0 01 CPX #1 ;SINGLE LETTER?
114 6087 D0 F7 BNE PHASE1 ;NO
115 6089 20 C5 61 JSR CALC
116 608C A9 01 LDA #1 ;NOW USED
117 608E 9D 7B 64 STA TYPTAB,X
118 6091 D0 ED BNE PHASE1
119 6093 60 NOVAR RTS
120 6094 4C 1B 61 PHASN JMP PHASEN
121
122 6097 20 73 62 PHASE2 JSR GTVR ;GET VARIABLE
123 609A 90 F7 BCC NOVAR
124 609C E0 01 CPX #1 ;SINGLE LETTER ONLY?
125 609E F0 F7 BEQ PHASE2 ;YES
126 60A0 20 D0 61 JSR PACK ;CUT DOWN TO ONE LETTER
127 60A3 AD 77 63 LDA OLET+2
128 60A6 CD 74 63 CMP LET+2
129 60A9 F0 EC BEQ PHASE2
130 60AB 20 64 62 JSR NEW
131 60AE 20 35 63 JSR VARP
132 60B1 20 45 63 JSR PARRON
133 60B4 20 53 63 JSR VARPON
134 60B7 20 BE FD JSR CR
135 60BA 4C 97 60 JMP PHASE2
136
137 60BD AD C6 64 PHASE3 LDA THFLAG
138 60C0 D0 47 BNE PHAB33
139 60C2 20 73 62 JSR GTVR
140 60C5 90 CC BCC NOVAR
141 60C7 E0 01 CPX #1 ;SINGLE LETTER
142 60C9 F0 F2 BEQ PHAB33
143 60CB 20 C5 61 JSR CALC
144 60CE BD 7B 64 LDA TYPTAB,X
145 60D1 D0 EA BNE PHAB33 ;LETTER USED?
146 60D3 A9 01 LDA #1 ;YES
147 60D5 9D 7B 64 STA TYPTAB,X ;USED NOW
148 60D8 20 64 62 JSR NEW ;SAVE WHAT WE FOUND
149 60DB C6 1D DEC YPOS ;BACK ONE FOR CUT TO 2
150 60DD 20 D0 61 JSR PACK ;PACK IT DOWN
151 60E0 AD 77 63 LDA OLET+2
152 60E3 BD 76 63 STA OLET+1
153 60E6 A9 00 LDA #0
154 60E8 BD 77 63 STA OLET+2
155 60EB 20 53 63 JSR VARPON
156 60EE 20 45 63 JSR PARRON
157 60F1 20 35 63 JSR VARP
158 60F4 20 BE FD JSR CR
159 60F7 A9 01 LDA #1
160 60F9 BD C6 64 STA THFLAG
161 60FC A5 FC LDA IND
162 60FE BD CA 64 STA INDBAV
163 6101 A5 FD LDA IND+1
164 6103 BD CB 64 STA INDBAV+1
165 6106 BC CC 64 STY YSAVE2
166 6109 20 73 62 PHAB33 JSR GTVR ;SAVE THESE
167 610C 90 85 BCC NOVAR ;GET NEXT ONE
168 610E 20 22 63 JSR COMP
169 6111 D0 F6 BNE PHAB33 ;SAME ONE?
170 6113 C6 1D DEC YPOS ;NO
171 6115 20 D0 61 JSR PACK
172 6118 4C 09 61 JMP PHAB33
173 611B A9 00 PHABEN LDA #0
174 611D BD C6 64 STA THFLAG ;NONE NOW
175 6120 AC CC 64 LDY YSAVE2 ;Y BACK
176 6123 AD CB 64 LDA INDBAV+1
177 6126 85 FD STA IND+1
178 6128 AD CA 64 LDA INDBAV
179 612B 85 FC STA IND
180 612D 4C 4C 60 JMP CABAIN ;MAKE CALL AGAIN
181 6130 60 NOVAR2 RTS
182
183 6131 AD C6 64 PHASE4 LDA THFLAG
184 6134 D0 71 BNE PHAB44
185 6136 20 73 62 JSR GTVR ;GET VARIABLE
186 6139 90 F5 BCC NOVAR2 ;NONE
187 613B E0 01 CPX #1 ;SINGLE?
188 613D F0 F2 BEQ PHASE4 ;YES
189 613F A6 1C LDX TYPE ;GET OFFSET FOR TYPE
190 6141 A9 1A LDA #26 ;ALPHABET
191 6143 BD CB 64 STA TEMP
192 6146 BD 7B 64 LOOKIT LDA TYPTAB,X ;ANY FREE?
193 6149 F0 08 BEQ GOTIT ;YES
194 614B EB INX
195 614C CE CB 64 DEC TEMP
196 614F D0 F5 BNE LOOKIT
197 6151 F0 DE BEQ PHASE4 ;NONE HERE
198 6153 A9 01 GOTIT LDA #1
199 6155 9D 7B 64 STA TYPTAB,X ;USED NOW
200 6158 8A TXA
201 6159 38 NOBMAL SEC
202 615A E9 1A SBC #26
203 615C B0 FB BCS NOBMAL
204 615E 18 CLC
205 615F 69 5B ADC #26+"A"-#80 ;MAKE INTO LETTER
206 6161 BD C9 64 STA NEWLET ;SAVE IT
207 6164 20 64 62 JSR NEW
208 6167 C6 1D DEC YPOS
209 6169 20 D0 61 JSR PACK
210 616C AD C9 64 LDA NEWLET
211 616F BD 75 63 STA OLET
212 6172 AD 77 63 LDA OLET+2
213 6175 BD 76 63 STA OLET+1
214 6178 A9 00 LDA #0
215 617A BD 77 63 STA OLET+2
216 617D 20 53 63 JSR VARPON
217 6180 20 45 63 JSR PARRON
218 6183 20 35 63 JSR VARP

```


Applesoft Variable Cruncher (Cont.)

2) BLOAD VARCRUNCH

3) CALL 24576

VARCRUNCH can also be **BRUN**'ed.

When the program is executing, it will indicate how far along it is and what changes it is making. It will print **PHASE X**, where **X** is 1 through 4 at the start of each phase. Since nothing is changed in **Phase 1**, no further information will be printed. In **Phase 2**, each variable that is shortened will be printed to the screen, such as

ZIPCODE --> ZI

This tells you the original variable name and its Applesoft interpreted form. If there are three occurrences of ZIPCODE, you will see this printed out three times as cruncher changes each one.

In **Phase 3** and **4**, any variable names crunched to one letter are printed out as above. This time you will see each variable appear only once, even if it is used in several places. The reason for this is that when a variable can be reduced, all occurrences of it must be changed at the same time. A sample run is shown in Example 1.

In order to get a hard copy listing of these changes, just activate the printer before executing the Variable Compressor.

SAVING PROGRAMS

Once you have compressed the variables, you can save or run your Applesoft program as usual. You might want to save both the original and the compressed versions. The original will probably be easier to read with its longer variable names. You can always use this version when making changes, and then just run the Variable Compressor when you want to produce the more efficient version.

SPEED IMPROVEMENTS

Long variables not only take more memory, they also slow down execution. Before we wrote this program, we never really knew how much of a difference, if any, it would make on execution time. To test this we wrote the simple program shown in **Example 2**. We compressed the first version and then ran both. The original version executed in about **61 seconds**, while the compressed version ran in **56 seconds**. That is about an **8% improvement**. This could be quite significant in some programs.

MODIFICATIONS

There may be occasions when you want some of the phases performed, but not all of them. For example, let's say you want only to remove the extra characters from the variable names. This is performed in Phase 2 of Variable Cruncher. You may modify the program to do this. At the symbol **JTAB** is the list of subroutines to be executed in the order given. If you do not want Phase 3 or Phase 4, change JTAB to

JTAB DFC PHASE1,PHASE1/256
DFC PHASE2,PHASE2/256
DFC PHASE2,PHASE2/256
DFC PHASE2,PHASE2/256

219	6186	20 BE FD	JBR CR	
220	6189	78	TYA	;SAVE Y
221	618A	48	PHA	
222	618B	A4 1D	LDY YPOS	
223	618D	88	DEY	
224	618E	AD C9 64	LDA NEWLET	
225	6191	91 FC	STA (IND),Y	
226	6193	68	PLA	
227	6194	A8	TAY	
228	6195	A9 02	LDA #2	
229	6197	8D C6 64	STA THFLAG	;4TH PHASE FLAG
230	619A	A5 FC	LDA IND	
231	619C	8D CA 64	STA INDBAV	
232	619F	A5 FD	LDA IND+1	
233	61A1	8D CB 64	STA INDBAV+1	
234	61A4	8C CC 64	STY YSAVE2	;SAVE FOR LATER
235	61A7	20 73 62	JBR GTVR	;GET NEXT ONE
236	61AA	90 84	BCC NOVAR2	
237	61AC	20 22 63	JBR COMP	;SAME ONE?
238	61AF	D0 F6	BNE PHAS44	;NO
239	61B1	C6 1D	DEC YPOS	
240	61B3	20 D0 61	JBR PACK	
241	61B6	78	TYA	
242	61B7	48	PHA	;SAVE Y
243	61B8	A4 1D	LDY YPOS	
244	61BA	88	DEY	
245	61BB	AD C9 64	LDA NEWLET	
246	61BE	91 FC	STA (IND),Y	
247	61C0	68	PLA	
248	61C1	A8	TAY	
249	61C2	4C A7 61	JMP PHAS44	
250				
251				
252	61C5	AD 72 63	CALC LDA LET	
253	61C8	38	SEC	
254	61C9	E9 41	SBC #A-#80	;GET INDEX
255	61CB	18	CLC	
256	61CC	65 1C	ADC TYPE	
257	61CE	AA	TAX	
258	61CF	60	NOVAR3 RTS	
259				
260				
261	61D0	38	PACK SEC	
262	61D1	78	TYA	;POSITION FOR END OF LINE
263	61D2	E5 1D	SBC YPOS	;GET LENGTH
264	61D4	F0 F9	BEQ NOVAR3	
265	61D6	85 FB	STA YSAV	;SAVE IT
266	61D8	A5 FC	LDA IND	
267	61DA	48	PHA	
268	61DB	85 FE	STA PIND	
269	61DD	A5 FD	LDA IND+1	
270	61DF	48	PHA	
271	61E0	85 FF	STA PIND+1	
272	61E2	A0 01	LDY #1	
273	61E4	B1 FE	KEEP LDA (PIND),Y	;CHANGE ADDRESS POINTERS
274	61E6	F0 19	BEQ ZMOR	;END
275	61E8	48	PHA	;SAVE HI ORDER
276	61E9	88	DEY	
277	61EA	B1 FE	LDA (PIND),Y	;LO ORDER
278	61EC	48	PHA	;SAVE IT TOO
279	61ED	38	SEC	
280	61EE	E5 FB	SBC YSAV	;BACK IT OFF
281	61F0	91 FE	STA (PIND),Y	;NEW LO ORDER
282	61F2	C8	INY	
283	61F3	B1 FE	LDA (PIND),Y	;HI ORDER
284	61F5	E9 00	SBC #0	;IF OVERFLOW
285	61F7	91 FE	STA (PIND),Y	;NEW HI ORDER
286	61F9	68	PLA	
287	61FA	85 FE	STA PIND	
288	61FC	68	PLA	
289	61FD	85 FF	STA PIND+1	;NEXT LINE
290	61FF	B0 E3	BCS KEEP	;CARRY SET HERE
291	6201	18	CLC	
292	6202	A5 1D	LDA YPOS	;GET WHERE TO PACK TO
293	6204	65 FC	ADC IND	
294	6206	85 FC	STA IND	
295	6208	48	PHA	;SAVE IT FOR PIND
296	6209	A5 FD	LDA IND+1	
297	620B	69 00	ADC #0	
298	620D	85 FD	STA IND+1	
299	620F	68	PLA	
300	6210	65 FB	ADC YSAV	;TO OLD NEXT LINE
301	6212	85 FE	STA PIND	
302	6214	A5 FD	LDA IND+1	
303	6216	69 00	ADC #0	
304	6218	85 FF	STA PIND+1	
305	621A	A0 00	LDY #0	
306	621C	B1 FE	OK LDA (PIND),Y	;MOVE DOWN EVERYBODY!!
307	621E	91 FC	STA (IND),Y	
308	6220	E6 FC	INC IND	
309	6222	D0 02	BNE NOH1	
310	6224	E6 FD	INC IND+1	
311	6226	E6 FE	NOH1 INC PIND	
312	6228	D0 02	BNE NOH2	
313	622A	E6 FF	NOH2 INC PIND+1	;NEXT LOCATION
314	622C	A5 FF	LDA PIND+1	
315	622E	C5 B0	CMP PEND+1	;END OF APPLESOFT?
316	6230	90 EA	BCC OK	;NO
317	6232	A5 FE	LDA PIND	
318	6234	C5 AF	CMP PEND	;LO ORDER END?
319	6236	90 E4	BCC OK	;NO
320	6238	A5 AF	LDA PEND	
321	623A	E5 FB	SBC YSAV	;CHANGE END
322	623C	85 AF	STA PEND	;NEW END
323	623E	85 69	STA LOM	;LOMEM TOO
324	6240	85 6B	STA LOM+2	
325	6242	85 6D	STA LOM+4	
326	6244	A5 B0	LDA PEND+1	
327	6246	E9 00	SBC #0	;HI ORDER
328	6248	85 B0	STA PEND+1	
329	624A	85 6A	STA LOM+1	
330	624C	85 6C	STA LOM+3	
331	624E	85 6E	STA LOM+5	

continued on next page

continued on next page

Applesoft Variable Cruncher (Cont.)

This tells the program to repeat Phase 2 three times, which doesn't do anything after its first execution.

If you do not want Phase 4, change JTAB to

```
JTAB DFC PHASE1,PHASE1/256
      DFC PHASE2,PHASE2/256
      DFC PHASE3,PHASE3/256
      DFC PHASE3,PHASE3/256
```

This will cause Phase 3 to be executed twice, but the end result is that nothing is done during the second execution.

CAUTIONS

There is an interesting side-effect created by this program. If you've ever used the variable name NO, you have probably found that Applesoft may interpret some character sequences as part of a token. For example, if you enter

100 IF X=NO THEN 150

Applesoft interprets it as

100 IF X=NOT HEN 150

and this causes a syntax error.

Now assume you have a variable **NO** that is compressed to **NO**. It is possible to have a statement

100 IF X=NO THEN 150

that does not cause an error when the program is run. However, if you copy over the line with the cursor, it will be retokenized as

100 IF X=NOT HEN 150

So running your Applesoft program through the Variable Cruncher will not change the way in which it works. If the program worked before it was crunched, it will still work. However, it may create some statements that cannot be recopied. This is particularly important if you compress the variables of a program, and then make a listing which you give to someone else to type into their system. If any variables are shortened to names that may be interpreted as tokens, the other person will not be able to get his program running. There is no easy way to enter the statement, except by putting it in the original form and then running the Cruncher. Fortunately, the chances of this happening are remote, and only become a problem when typing in a program or modifying it.

THE LONG AND SHORT ...

Long variable names are lovely to look at, but not so beautiful to hold in memory. At least with Variable Cruncher you can have both options — one long version for development and one short version for execution.



```

332 6250 A0 00 LDY #0
333 6252 98 TYA ;Y=0
334 6253 91 FC STA (IND),Y ;END OF PROGRAM HAS 0'S
335 6255 C8 INY
336 6256 91 FC STA (IND),Y
337 6258 C8 INY
338 6259 91 FC STA (IND),Y
339 625B A4 1D LDY YPOS
340 625D 68 PLA
341 625E 85 FD STA IND+1
342 6260 68 PLA
343 6261 85 FC STA IND
344 6263 60 RTS

;
;SAVE NEW VARIABLE FOUND IN LOWEST ONE
;
NEW LDX #2
N2 LDA LET,X
STA LLET,X
DEX
BPL N2
RTS
BEND CLC
RTS

;
;GET VARIABLE IN LINE
;CARRY SET IF ONE FOUND
;STORE IT IN LET
;SAVE UP TO 3 CHARACTERS
;
BTVRP PLA
BTVR LDA #0
STA TYPE
STA OLDVAR
LDA (IND),Y
BEQ BEND ;END OF LINE
INY
CMP #178 ;REM?
BEQ BEND ;YES-IGNORE LINE
LDX #0 ;FOR COLON
CMP #131 ;DATA TOKEN
BEQ NYT ;FIND COLON
INX ;FOR QUOTE
CMP #22 ;QUOTE?
BNE OKYT ;NO -OK YET
LDA (IND),Y
BEQ BEND ;END OF LINE
INY
CMP QCOL,X ;QUOTE OR COLON
BNE NYT
OKYT CMP #A-#80
BCC GTVR
CMP #Z+1-#80 ;VALID LETTER?
BCS GTVR ;NO
PHA
CMP #E-#80 ;E?
BNE NOEXP ;NOT EXPONENT
CPY #6 ;AT START OF LINE?
BCC NOEXP ;YES
DEY
DEY
LDA (IND),Y ;GET CHAR BEFORE IT
INY
INY
CMP #.-#80 ;DECIMAL POINT?
BEQ GTVRP ;YES - NOT VARIABLE
CMP #0-#80 ;NUMBER?
BCC NOEXP ;NO
CMP #9+1-#80 ;NUMBER?
BCC GTVRP ;YES-IS EXPONENT
LDX #2
NOEXP LDA #0
ZLOP STA LET,X
DEX
BPL ZLOP ;ZERO STORAGE
LDX #0
ZLOP2 STA OLET,X
DEX ;ZERO IT OUT
BNE ZLOP2
TAX
PLA
GOOD CPX #2 ;ENOUGH?
BCS LOOK ;YES
STY YPOS
STA LET,X ;SAVE IT
INX
LOOK STX SAVEX
LDX OLDVAR
STA OLET,X ;SAVE IT FOR PRINTING
INC OLDVAR
LDX SAVEX
LDA (IND),Y
BEQ NFND
INY
CMP #A-#80
BCC CHKM
CMP #Z+1-#80 ;LETTER?
BCC GOOD ;YES
CMP #0-#80 ;NUMBER?
BCC CHK2
CMP #9+1-#80
BCC GOOD ;YES
CHK2 CMP #E-#80 ;STRING?
BEQ FEND ;STOP
CMP #X-#80 ;INTEGER?
BNE NFND ;NO
PHA ;SAVE A
LDA #26#2
FEND2 STA TYPE
PLA
STA LET,X ;SAVE IT
STX SAVEX ;SAVE X
LDX OLDVAR
STA OLET,X ;SAVE IT FOR PRINTING

```


Applesoft Variable Cruncher (Cont.)

```

446 6314 A6 1E      LDX SAVEX      ;GET X BACK
447 6316 B8         DEY           ;BACK TO INVALID
448 6317 38         NFND         SEC      ;JUST STOP
449 6318 A9 01      LDA #1       ;NOT EQUAL
450 631A 60         FEND         RTS
451 631B 48         PHA          ;SECOND GROUP
452 631C A9 1A      LDA #26      ;SECOND GROUP
453 631E D0 E6      BNE FEND2
454          ;DATA TOKEN,QUOTE
455 6320 3A 22      QCOL DFC #3A,#22
456          ;
457          ;LET =PRESENT
458          ;LLET=LEAST ONE SO FAR
459          ;
460          ;EQUALS SET IF EQUAL
461          ;
462 6322 A2 00      COMP LDX #0
463 6324 BD 75 64   CMP2 LDA LLET,X
464 6327 DD 72 63   CMP2 CMP LLET,X
465 632A D0 06      BNE NOK       ;NOT YET
466 632C EB        INX
467 632D E0 03      CPX #3
468 632F 90 F3      BCC CMP2
469          ; EQUAL
470 6331 60         RTS
471 6332 A9 01      NOK LDA #1     ;NOT EQUAL
472 6334 60         RTS
473          ;
474          ;PRINT VARIABLE
475          ;
476 6335 A2 00      VARP LDX #0
477 6337 BD 75 63   M2 LDA DLET,X
478 633A F0 08      BEQ VQUIT
479 633C 09 B0      ORA #90
480 633E 20 ED FD   JSR DOUT
481 6341 EB        INX
482 6342 D0 F3      BNE M2
483 6344 60         VQUIT RTS
484          ;
485          ;PRINT ARROW
486          ;
487 6345 A2 00      PARROW LDX #0
488 6347 BD 65 63   M3 LDA ARROW,X
489 634A F0 06      BEQ VQUIT2
490 634C 20 ED FD   JSR DOUT
491 634F EB        INX
492 6350 D0 F5      BNE M3
493 6352 60         VQUIT2 RTS
494          ;
495 6353 A2 00      VARPON LDX #0
496 6355 BD 75 64   M LDA LLET,X
497 6358 F0 0A      BEQ QUIT
498 635A 09 B0      ORA #90
499 635C 20 ED FD   JSR DOUT
500 635F EB        INX
501 6360 E0 03      CPX #3
502 6362 90 F1      BCC M
503 6364 60         QUIT RTS
504          ;
505 6365 A0 AD AD    ARROW ABC " --> "
506 636A 00         BE A0
507 636B D0 C8 C1    PPHAS DFC 0
508 6371 00         ASC "PHASE "
509          ;
510          ;
511          ;DSC
512          ;
513          ;LET DFS 3
514          ;OLET DFS 256
515          ;LLET DFS 3
516          ;TYPTAB DFS 26#3
517          ;THFLAG DFS 1
518          ;OLDVAR DFS 1
519          ;TEMP DFS 1
520          ;NEWLET DFS 1
521          ;INDSAV DFS 2
522          ;YSAVE2 DFS 1

```

JLIST

Example 1

```

10 REM THIS IS A NONSENCE ROUTINE FOR
20 REM AN EXAMPLE
30 B = C = D = E = F = G = H = I = J = K = L = M
  = N = O = P = Q = R = S = T = U = V = W = X = Y = 0
40 ABCD = 2
50 BEEP = 1.5:ABC = 10.2
60 IF BEG = 1.5 THEN BE = 2.0
70 CHNG = 5
80 PRINT CHNG,B,AB,BED
90 GOTO 100
100 PRINT "ABCD"
110 END

```

JBRUN VARCOMP
PHASE 1

PHASE 2

```

ABCD --> AB
BEEP --> BE
ABC --> AB
BEG --> BE
CHNG --> CH
CHNG --> CH
BED --> BE

```

PHASE 3

AB --> A

PHASE 4

BE --> Z

Example 2

JLIST

```

10 REM THIS IS A NONSENCE ROUTINE FOR
20 REM AN EXAMPLE
30 B = C = D = E = F = G = H = I = J = K = L = M
  = N = O = P = Q = R = S = T = U = V = W = X = Y = 0
40 A = 2
50 Z = 1.5: A = 10.2
60 IF Z = 1.5 THEN Z = 2.0
70 CH = 5
80 PRINT CH,B,A,Z
90 GOTO 100
100 PRINT "ABCD"
110 END

```

JPR#0

JLIST

```

10 REM
20 ABCDE = 0
30 ABCDE = ABCDE + 1
35 QRST = QRST + 1
40 IF ABCDE > 5000 THEN 100
50 GOTO 30
100 PRINT "STOP"
120 END

```

JPR#0

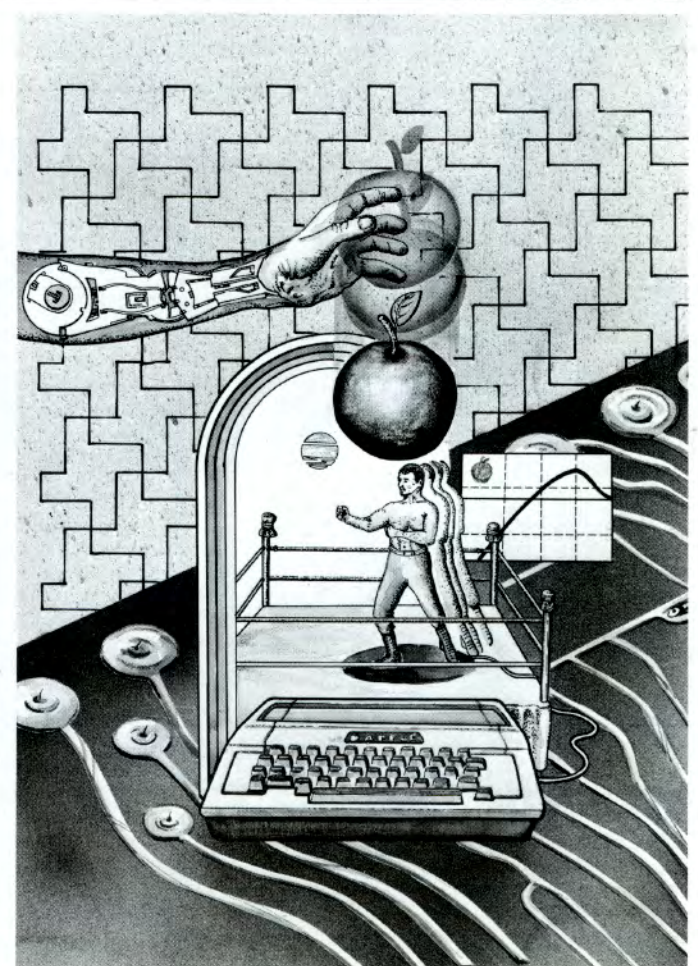
JLIST

```

10 REM
20 A = 0
30 A = A + 1
35 Q = Q + 1
40 IF A > 5000 THEN 100
50 GOTO 30
100 PRINT "STOP"
120 END

```

JPR#0



LIFE

by Raul Cartaya
1040 Coast Blvd. So. #401
La Jolla, CA 92037

"... And pattern itself being weightless,
The life integrities are apparently Inher-
ently immortal."

Buckminster Fuller,
from "How Little I Know"

INTRODUCTION

LIFE is a game of patterns in which the player enters the initial pattern, or generation 0, and then sits back and watches the patterns of successive generations being displayed. LIFE determines the pattern for the next generation by applying **three very simple laws** to the current generation: the laws of **birth**, **survival**, and **death**. These laws, although very simple, are extremely cumbersome to apply by hand to any but the simplest of patterns without making mistakes. This, in a way, makes LIFE a game for computers only.

Our LIFE's universe is basically a checker-board of **35 by 20 squares** (although the separations between the squares are normally not visible). A pattern, or generation, is made by placing markers (circles) on some of the squares (or places), and leaving the rest empty. **Any one cell will have somewhere between 0 and 8 neighbors**, or occupied adjacent places (note that in our version of LIFE, the universe is made to **wrap around** in both dimensions, so that the top and bottom rows are considered adjacent, as are the left-most and rightmost columns). The three laws governing LIFE can then be stated as follows:

BIRTH: an empty place will give birth to a marker in the next generation if it has exactly 3 neighbors.

SURVIVAL: an inhabited place will still be inhabited in the next generation if it has either 2 or 3 neighbors.

DEATH: an inhabited place will become empty in the next generation (i.e., the marker will die) if it has less than 2 neighbors or more than 3 (death from loneliness or overcrowding).

There is nothing new about LIFE, even though this program is completely original. The popularity of this game reached crazy proportions among computer people about 10 years ago, which led to articles in a number of publications including *Time* and *Scientific American*. At one point there must have been hundreds, if not thousands of different implementations of LIFE eating up machine cycles in as many computers from coast to coast. Why the proliferation? First, because it is a fascinating game to watch: patterns evolve in elegantly complex and seemingly unpredictable ways, guided by such strong inter-reactions that **changing the location of one cell will often result in radically different evolutions**. The second and perhaps strongest reason for the popularity of LIFE is how simply it is defined: the three laws of LIFE are, in essence, the complete algorithm for the game; armed with knowledge of these laws and a little ingenuity, any programmer could implement his own version of LIFE.

Individual versions of LIFE may disappear or be forgotten, but the integrity of the pattern-processing algorithm expressed by the three laws is immortal: the laws will con-

tinue to produce identical subsequent generations given the same input pattern, whether in the abstract, manually, or in automated implementations, such as ours.

THE PROGRAM

In a brute force approach to LIFE, one would determine the next generation by applying the three laws to every single location on the 35 × 20 board. That is, for every location on the board, one would count the surrounding neighbors and determine the fate of that location in the next generation. This would result in a valid but very slow version of LIFE: a lot of time would be spent looking at locations where nothing was going to change. Our version of LIFE uses a number of shortcuts that are best described by explaining the important variable and their usage.

THE VARIABLES

P%(34,19) is the board itself, the population array. Each element corresponds to a place on the board. A 1 indicates an inhabited place, and a 0 an empty place.

N%(34,19) is the neighbor array. After each location is scanned, the corresponding element contains the number of neighbors at that location (in the previous generation). The location itself is counted, so contents can range from 0 to 9.

D%(34,19) is the delta array. At the end of a scan, it shows the change in the number of neighbors for each location. When a marker is born (or dies) during the scan process at a particular location, the corresponding element and the eight neighboring ones are incremented (or decremented) by 1. Contents may range from -9 to 9. Obviously, changes can only occur in locations whose D% element is non-zero.

CL%(734), DL%(734) are the delta lists. At the end of each scan, when D% is stable, these two arrays are used to summarize the information in D% in a way designed to speed up the scanning process. D% is then set to 0 so it is ready for the next scan. CL% is, essentially, a list of the locations where the number of neighbors has changed (i.e., non-zero elements in D%), so that they are potential candidates for births or deaths. Rather than list both X and Y values for the location, only the Y values are listed, and a change in X is indicated by a negative value. The corresponding element in DL% then shows the change in the number of neighbors for that location. The list of values shown in Figure 1 indicates that the first location to be scanned should be X=5, Y=7, where the number of neighbors increased by 3. Note that, since these arrays are treated as lists, 734 represents only their maximum length. The actual length of the lists at any time is kept in CN.

CL%	-1	-1	-1	-1	7
DL%	?	?	?	?	3

Figure 1

XL, YL, XH, and YH are, finally, the margin variables. They contain the low and high values for X and Y for any location in which changes have occurred during the scan process. They are used to speed up the generation of CL% and DL%.

ORGANIZATION

The program is organized as follows:

Lines 1000-1850: this is the **player state**, where you can place or remove markers, turn the grid on or off, copy a pattern to disk (or add a pattern to the screen from disk), wipe the screen clear, or enter the run state. The **X command** (place/remove marker) and the **A command** (add pattern from disk) make use of the birth/death routine at 4000. ESC passes control to the run state at line 3500.

Lines 3000-3120: this is the scan portion of the **run state**. For each location listed in CL%, N% is updated by the contents of DL%. P% and N% are then used to determine whether a birth or death occurs (routine at 4000). The graphic cursor is turned on and off by lines 3040 and 3110, so the scan may be seen on the screen.

Lines 3500-3560: this portion **prints the generation statistics**. If ESC was pressed during the scan process, the program returns to the player state at 195. If all the markers disappear or if the pattern becomes stable, the program keeps looping until ESC is pressed. Otherwise, the post-scan process at 4500 is called, and then control goes back to 3000.

Lines 4000-4190: birth or death at X, Y. P%, D%, and the display are updated, as well as generation statistics and the margin variables.

Lines 4500-4640: the post-scan process. The rectangle delineated by the margin variables is examined in D%. CL% and DL% are generated, and D% is zeroed out. Finally, the margin variables are reset and control returns to the caller.

The rest of the program should be relatively simple to follow. Some unusual constructions are used in a few places (32, 34, 3080-3090, 4025-4080); these simply proved to be a bit faster than the more straightforward constructions.

RUNNING LIFE

LIFE is very easy to use. When you **RUN** the program, you will be offered three pages of instructions, which will paraphrase much of what has been stated here. Next, you will be shown the **HELP menu** (subroutine 250), describing all of the commands available to you while in the player state.

There is a marker on/off command (X) which reverses the population at the location shown by the graphic cursor, two grid commands (G for on, N for off), 8 cursor movement commands (U, D, L, R are self-explanatory; Y and I for up/left and up/right, and S and F for down/left and down/right; if you have trouble remembering the diagonal commands, just note that they are placed on the corresponding side of the respective Up and Down keys). These commands allow you to move the cursor around the board in order to add or remove markers.

The next two commands allow you to **Copy the whole screen to disk (C)** or **Add a disk pattern to the current screen (A)**. In both commands you are given the option to **name the file** (replying "X" will access the file "LIFE PATTERN X"), and to **measure marker positions from the board origin (0,0, the upper left hand corner) or from the current cursor position**.

LIFE (Cont.)

OTHER COMMANDS

The remaining commands should be described individually. **W** clears the board and all arrays and statistics, and returns to the HELP menu. **ESC** puts the game in its run state. **H** leaves everything as is but re-displays the HELP menu. Finally, **Q** quits the program by doing a DOS FP (the reason for this will be explained below).

Once you have studied the HELP menu, pressing any key will take the game to the true player state, with a Hi-Res display of the board and some lines of information summarizing commands, cursor position, and population. To enter a command, just depress the appropriate key (no RETURN). RETURN is only needed when answering the questions resulting from an A or C command.

Once you have entered a pattern in the player state, press ESC to enter the run state. The graphic display will remain the same, but now the lower lines will contain generation statistics. There will be a pause while the program post-scans, then the bottom of the screen will be blanked as the program scans the pattern and predicts the next generation.

Normally, after the scan the program will display the new generation statistics and go into the post-scan. If you press ESC during the scan process, however, the end of the scan will take you back to the player state, from where patterns may again be manipulated. Also, if during the scan process the program determines that all markers have died or that all remaining patterns are stable, the appropriate message will be displayed and the program will wait for an ESC to go to the player state.

THE LIFE PATTERNS

The theoretical number of distinct generation 0's that can be entered into LIFE is a mind-boggling 2 to the 700th power. Logs will tell you that this is a number larger than a 1 followed by 210 zeros, a pretty big number.

Never fear, though, for the great majority of these numbers are just meaningless patterns, such as a blank or fully populated screen, or a series representing just one simple pattern as it appears on every possible location on the screen.

You will soon see, however, that even some of the simplest patterns will result in very complex geometrical evolutions. Take a straight horizontal line of cells; depending on the length of the line, the fate of the population will be entirely different. For lengths of 1 to 10, the following are the end results:

1. dies in 1 generation.
2. dies in 1 generation.
3. flips to a vertical line in 1, then continues to oscillate back and forth every 2 generations (a flag).
4. becomes stable in 2 generations.
5. becomes 4 flags in 6 generations, which then oscillate every 2.
6. dies in 12 generations.
7. becomes stable in 14 generations (4 of the patterns resulting from a 4-long line).
8. becomes stable in 49 generations (!!).
9. becomes 8 flags in 20 generations (2 of the patterns resulting from a 5-long line), then oscillates in 2.
10. reaches a shape in 2 generations which will then oscillate every 15 generations.

Two other interesting patterns you might try are SHIP and CHESHIRE (Figures 2 and 3). In 4 generations, SHIP will look exactly the same (no gains or losses), but it will have moved one location to the right and one down. In 560 generations it will be exactly where it started on the screen. It is also interesting to watch it (and rotations thereof) collide with themselves or with other patterns.

CHESHIRE is a cute one. Of course, it is the cat without a smile of Lewis Carroll. Let him run and in 6 generations he will be a smile without a cat, and finally just a stable paw-print.

SPECIAL NOTES

Once LIFE has been RUN, it should not be interrupted in order to SAVE it or modify it, because the resulting version will not work. Some of the introductory material and shapable instructions which are used very early in the program are actually kept at the end of the code (beyond line 5000), and most of these statements are lost when entering Hi-Res graphics. Once you exit the program, consider the copy in memory to be gone (the Q command in the player state ensures this for you).

Some care should be exercised when modifying a pattern that had been evolving (i.e., the player state was reached by pressing ESC during scan). If you add a marker to a location where one has just died (or vice versa), the change will appear on the screen as you desired, but the changes entered into the delta array for that location and the surrounding ones will have been nullified. This may result in that area of the board evolving bizarrely for at least a while. For best results, if you want to try something like this, first save the pattern, then do a W, add the pattern back in and then put in your changes.



Figure 2

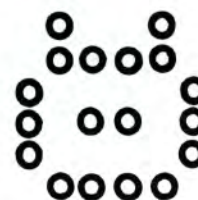


Figure 3

LIST

```

1 REM *****
2 REM $ "LIFE" $
3 REM $ BY RAUL CARTAYA $
4 REM $ COPYRIGHT (C) 1982 $
5 REM $ BY MICRO-SPARC INC. $
6 REM $ ALL RIGHTS RESERVED $
7 REM $ REQUIRES APPLESOFT IN ROM $
8 REM $ AND DISK. $
9 REM *****
10 TEXT : HOME : VTAB 7: LOMEM: 4 $ 4096
11 P$ = "WELCOME TO LIFE"
12 GOSUB 20: GOTO 22
13 XDRAW 1 AT FN PC(X), FN PC(Y): RETURN
14 PRINT TAB( 20 - LEN (P$) / 2);P$: RETURN
15 PRINT TAB( 20 - LEN (P$) / 2);P$: RETURN
16 VTAB 10:P$ = "DO YOU NEED AN INTRODUCTION?": GOSUB 20
17 P$ = "(Y OR N, RETURN)": GOSUB 21
18 INPUT "":A$
19 IF A$ = "Y" THEN GOSUB 5000
20 GOSUB 6000
21 DEF FN NX(X) = 35 $ (X < 0) + X $ (X < 35)
22 DEF FN NY(Y) = 20 $ (Y < 0) + Y $ (Y < 20)
23 DEF FN PC(Z) = B $ Z + 4
24 DIM P$(34,19),NX(34,19),DX(34,19)
25 DIM CL$(734),DL$(734)
26 DD$ = CHR$(4):XX$ = "LIFE PATTERN"
27 REM PREPARE FOR PLAYER STATE
28 GOSUB 250
29 HGR :NM = 0
30 GOSUB 4630
31 X = 17:Y = 10
32 GOTO 1000
33 REM EXPLAIN PLAYER MENU
34 HOME
35 PRINT "PRESS ANY KEY TO GO INTO PLAYER STATE."
36 PRINT "IN THAT STATE, THE FOLLOWING COMMANDS"
37 PRINT "ARE AVAILABLE:"
38 POKE 33,32: POKE 32,8: PRINT

```

```

310 PRINT "X - PLACE/REMOVE MARKER"
320 PRINT "G - TURN ON GRID"
330 PRINT "N - TURN OFF GRID"
340 PRINT "U - MOVE CURSOR UP"
350 PRINT "D - MOVE DOWN"
360 PRINT "L - MOVE LEFT"
370 PRINT "R - MOVE RIGHT"
380 PRINT "Y - UP AND LEFT"
390 PRINT "I - UP AND RIGHT"
400 PRINT "S - DOWN AND LEFT"
410 PRINT "F - DOWN AND RIGHT"
420 PRINT "C - COPY TO DISK"
430 PRINT "A - ADD FROM DISK"
440 PRINT "W - WIPE AND RE-START"
450 PRINT "ESC - GO TO RUN STATE"
460 PRINT "H - RE-DISPLAY THIS PAGE"
470 PRINT "Q - QUIT PROGRAM (END)"
478 TEXT
480 POKE 32,0
490 PRINT "HIT ANY KEY TO CONTINUE": GET Z$: HOME : RETURN

1000 REM PLAYER STATE
1010 POKE - 16304,0: HOME : VTAB 21
1020 PRINT "COMMANDS - X G N U D L R Y I S F C A Q"
1030 PRINT "H FOR HELP ESC TO RUN"
1040 PRINT "MARKERS: X,Y: "
1050 VTAB 23: HTAB 10: PRINT NM
1060 HTAB 31: PRINT X",Y"
1065 GOSUB 18
1070 GET A$
1080 GOSUB 18
1090 IF A$ = "X" THEN 1300
1100 IF A$ = "G" THEN 1370
1110 IF A$ = "N" THEN 1350
1120 IF A$ = "U" THEN 1410
1130 IF A$ = "D" THEN 1420
1140 IF A$ = "L" THEN 1430
1150 IF A$ = "R" THEN 1440
1160 IF A$ = "Y" THEN 1450
1170 IF A$ = "I" THEN 1460

```

continued on next page

LIFE (Cont.)

```

1180 IF A$ = "S" THEN 1470
1190 IF A$ = "F" THEN 1480
1200 IF A$ = "C" THEN 1500
1210 IF A$ = "A" THEN 1700
1220 IF A$ = "M" THEN CLEAR : TEXT : GOTO 32
1230 IF A$ = CHR$(27) THEN 3500
1240 IF A$ = "H" THEN TEXT : GOSUB 250: GOTO 1000
1250 IF A$ = "Q" THEN PRINT : TEXT : HOME : PRINT DD$"FP"

1260 GOTO 1050
1300 REM PLACE/REMOVE MARKER
1320 GOSUB 4000: GOTO 1050
1350 REM GRID OFF (ON AT 2ND LINE)
1360 HCOLOR= 0
1370 FOR Y2 = 8 TO 152 STEP 8: FOR X2 = 8 TO 272 STEP 8: HPLLOT
    X2,Y2: NEXT X2
1380 HCOLOR= 3: GOTO 1050
1400 REM CURSOR MOVES
1410 DY = - 1: GOTO 1443
1420 DY = 1: GOTO 1443
1430 DX = - 1: GOTO 1443
1440 DX = 1
1443 X = FN NX(X + DX)
1445 Y = FN NY(Y + DY)
1448 DX = 0: DY = 0: GOTO 1050
1450 DX = - 1: GOTO 1410
1460 DX = 1: GOTO 1410
1470 DX = - 1: GOTO 1420
1480 DX = 1: GOTO 1420
1500 REM COPY TO DISK
1510 IF NOT NM THEN 1050
1520 P$ = "COPY TO DISK": GOSUB 1800
1530 PRINT DD$"WRITE" P$
1540 PRINT NM
1550 FOR X = 0 TO 34: FOR Y = 0 TO 19
1560 IF NOT P$(X,Y) THEN 1600
1570 X2 = X: Y2 = Y
1580 IF X0 THEN X2 = X2 - X4: Y2 = Y2 - Y4
1590 GOSUB 18: PRINT X2: PRINT Y2: GOSUB 18
1600 NEXT Y: NEXT X
1610 GOTO 1780
1700 REM ADD FROM DISK
1710 P$ = "ADD FROM DISK": GOSUB 1800
1720 PRINT DD$"READ" P$
1730 INPUT M
1740 FOR I = 1 TO M
1750 INPUT X,Y: IF X0 THEN X = X + X4: Y = Y + Y4
1755 IF X > 34 THEN X = X - 34
1757 IF Y > 19 THEN Y = Y - 19
1760 X = FN NX(X): Y = FN NY(Y): GOSUB 18: GOSUB 4000: GOSUB
    18
1770 NEXT
1780 HOME : VTAB 21: P$ = F$ + " DONE"
1790 PRINT DD$"CLOSE" F$: X4: Y4: GOSUB 20: GOSUB 18
    : GOTO 1000
1800 HOME : VTAB 21: GOSUB 18: GOSUB 20: X4 = X: Y4 = Y
1810 PRINT "NAME FOR " X$: INPUT F$: F$ = X$: F$
1820 PRINT DD$"OPEN" F$
1830 INPUT "MEASURE FROM CURSOR? (Y OR N)": Z$
1840 X0 = 0: IF Z$ = "Y" THEN X0 = 1
1850 RETURN
2999 REM RUN STATE
3000 NC = 0: ND = 0: NG = NG + 1: HOME
3010 X = 0
3020 FOR I = 1 TO CN
3030 Y = CLX(I): IF Y < 0 THEN X = X + 1: GOTO 3120
3040 GOSUB 18
3050 D = DLX(I)
3070 N = NX(X,Y) + D: NX(X,Y) = N
3080 IF P$(X,Y) THEN ON N GOTO 3100,3100,3110,3110,3100,
    3100,3100,3100,3100
3090 IF N < 3 THEN 3110
3100 GOSUB 4000
3110 GOSUB 18
3120 NEXT
3490 REM GENERATION STATS
3500 HOME
3505 IF PEEK(-16384) = 155 THEN I = PEEK(-16368):
    GOTO 195
3510 VTAB 21: PRINT "GENERATION " NG: TAB(21) "MARKERS: " N
    M
3520 PRINT "NET CHANGE: " ND: TAB(21) "BIRTHS+DEATHS: " NC
3530 IF NOT NM THEN PRINT "ALL LIFE HAS BECOME EXTINCT.
    ...": GOTO 3505
3540 IF NOT NC THEN PRINT "LIFE IS STABLE...": GOTO 350
    5
3550 PRINT "HIT ESC FOR PLAYER STATE":
3560 GOSUB 4500: GOTO 3000
3990 REM BIRTH/DEATH
4000 XDRAW 2 AT FN PC(X), FN PC(Y)
4010 P$(X,Y) = 1 - P$(X,Y)
4020 D = - 1: IF P$(X,Y) THEN D = 1
4025 IF X = 0 OR X = 34 OR Y = 0 OR Y = 19 THEN 4120
4030 DX(X-1,Y-1) = DX(X-1,Y-1) + D
4035 DX(X-1,Y) = DX(X-1,Y) + D
4040 DX(X-1,Y+1) = DX(X-1,Y+1) + D
4045 DX(X,Y-1) = DX(X,Y-1) + D
4047 DX(X,Y) = DX(X,Y) + D
4050 DX(X,Y+1) = DX(X,Y+1) + D
4060 DX(X+1,Y-1) = DX(X+1,Y-1) + D
4070 DX(X+1,Y) = DX(X+1,Y) + D
4080 DX(X+1,Y+1) = DX(X+1,Y+1) + D
4090 NM = NM + D
4100 NC = NC + 1
4105 ND = ND + D
4106 IF X < XL THEN XL = X
4107 IF Y < YL THEN YL = Y
4108 IF X > XH THEN XH = X
4109 IF Y > YH THEN YH = Y
4110 RETURN

```

```

4115 REM BIRTH/DEATH AT EDGE
4120 FOR X2 = X - 1 TO X + 1
4130 FOR Y2 = Y - 1 TO Y + 1
4140 X3 = FN NX(X2): Y3 = FN NY(Y2)
4150 DX(X3,Y3) = DX(X3,Y3) + D
4160 NEXT Y2: NEXT X2
4190 GOTO 4090
4490 REM LIST OF "CHANGES?"
4500 CN = 0
4510 XL = FN NX(XL-1): XH = FN NX(XH+1): IF XL > = X
    H THEN I = XL: XL = XH: XH = 1: IF XL = 0 OR XH = 34 THEN
    XL = 0: XH = 34
4520 YL = FN NY(YL-1): YH = FN NY(YH+1): IF YL > = Y
    H THEN I = YL: YL = YH: YH = 1: IF YL = 0 OR YH = 19 THEN
    YL = 0: YH = 19
4530 IF XL = 0 THEN 4550
4540 FOR I = 0 TO XL - 1: CN = CN + 1: CLX(CN) = - 1: NEXT
    I
4550 FOR X = XL TO XH: FOR Y = YL TO YH
4560 IF NOT DX(X,Y) THEN NEXT Y: GOTO 4610
4570 D = DX(X,Y)
4580 CN = CN + 1: DLX(X,Y) = 0
4590 CLX(CN) = Y: DLX(CN) = D
4600 NEXT Y
4610 CN = CN + 1: CLX(CN) = - 1
4620 NEXT X
4630 XL = 34: XH = 0: YL = 19: YH = 0
4640 RETURN
5000 REM INSTRUCTIONS
5010 HOME
5020 PRINT "LIFE IS A GAME OF PATTERNS. A PATTERN"
5030 PRINT "IS MADE UP OF INHABITED PLACES (OCCUPIED),"
5040 PRINT "BY MARKERS) AND EMPTY PLACES. ANY ONE"
5050 PRINT "PATTERN IS CALLED A GENERATION.": PRINT
5060 PRINT "YOU PLAY LIFE BY ENTERING THE PATTERN"
5070 PRINT "FOR THE FIRST GENERATION, AND THEN WAIT-"
5080 PRINT "CHING IT EVOLVE. EACH GENERATION DETER-"
5090 PRINT "MINES THE PATTERN FOR THE NEXT GENERA-"
5100 PRINT "TION BASED ON THE LAWS OF BIRTH,"
5110 PRINT "SURVIVAL, AND DEATH. YOU SIMPLY WATCH"
5120 PRINT "THE PATTERNS CHANGE AS THE GENERATIONS"
5130 PRINT "EVOLVE.": PRINT
5140 PRINT "IN ADDITION, YOU MAY CHANGE THE PATTERN"
5150 PRINT "AT THE END OF ANY GENERATION. PRESSING"
5160 PRINT "'ESC' WILL SWITCH FROM THE RUN STATE"
5170 PRINT "BACK TO PLAYER STATE, WHERE YOU MAY MO-"
5180 PRINT "DIFY THE PATTERN, SAVE IT TO DISK, RE-"
5190 PRINT "TURN TO THE RUN STATE, OR END THE PRO-"
5200 PRINT "GRAM.": GOSUB 5470
5210 PRINT "FOLLOWING ARE THE THREE BASIC LAWS."
5220 PRINT "REMEMBER THAT EACH PLACE ON THE BOARD"
5230 PRINT "HAS EIGHT ADJACENT PLACES. 'HAVING TWO"
5240 PRINT "'NEIGHBORS' MEANS HAVING TWO ADJACENT"
5250 PRINT "PLACES WHICH ARE INHABITED.": PRINT
5260 PRINT "BIRTH - AN EMPTY PLACE WILL GIVE BIRTH"
5270 HTAB 9: PRINT "TO A MARKER IN THE NEXT GENERA-"
5280 HTAB 9: PRINT "TION IF IT HAS EXACTLY 3 NEIGH-"
5290 HTAB 9: PRINT "BORS.": PRINT
5300 PRINT "SURVIVAL - AN INHABITED PLACE WILL STILL"
5310 HTAB 9: PRINT "BE INHABITED IN THE NEXT GENERA-"
5320 HTAB 9: PRINT "TION IF IT HAS 2 OR 3 NEIGHBORS.": PRINT
5330 PRINT "DEATH - AN INHABITED PLACE WILL BECOME"
5340 HTAB 9: PRINT "EMPTY IN THE NEXT GENERATION"
5350 HTAB 9: PRINT "(THE MARKER WILL DIE) IF IT HAS"
5360 HTAB 9: PRINT "LESS THAN 2 NEIGHBORS OR MORE"
5370 HTAB 9: PRINT "THAN 3 (DEATH FROM LONELINESS OR)"
5380 HTAB 9: PRINT "OVERCROWDING.": GOSUB 5470
5390 PRINT "LIFE IS PLAYED ON A 35X20 BOARD. HORI-"
5400 PRINT "ZONTAL LOCATIONS (X) ARE (LEFT TO RIGHT)":
5410 PRINT "0 THROUGH 34. VERTICAL LOCATIONS (Y) ARE":
5420 PRINT "(TOP TO BOTTOM) 0 THROUGH 19.": PRINT
5430 PRINT "THE BOARD WRAPS AROUND BOTH VERTICALLY"
5440 PRINT "AND HORIZONTALLY, SO THE TOP AND BOT-"
5450 PRINT "TOM ROWS (OR THE LEFTMOST AND RIGHTMOST"
5460 PRINT "COLUMNS) ARE ADJACENT. HAVE FUN!"
5470 VTAB 23: GOTO 490
6000 REM PUT IN SHAPE TABLE
6010 POKE 768,2: POKE 769,0: POKE 770,6: POKE 771,0:
6020 POKE 772,35: POKE 773,0: POKE 774,8: POKE 775,8:
6030 POKE 776,8: POKE 777,54: POKE 778,54: POKE 779,54:
6040 POKE 780,39: POKE 781,36: POKE 782,36: POKE 783,60:
6050 POKE 784,54: POKE 785,54: POKE 786,54: POKE 787,39:
6060 POKE 788,36: POKE 789,36: POKE 790,60: POKE 791,54:
6070 POKE 792,54: POKE 793,54: POKE 794,39: POKE 795,36:
6080 POKE 796,36: POKE 797,60: POKE 798,54: POKE 799,54:
6090 POKE 800,54: POKE 801,95: POKE 802,0: POKE 803,8:
6100 POKE 804,8: POKE 805,200: POKE 806,59: POKE 807,247:
6110 POKE 808,30: POKE 809,54: POKE 810,14: POKE 811,14:
6120 POKE 812,45: POKE 813,12: POKE 814,12: POKE 815,36:
6130 POKE 816,28: POKE 817,92: POKE 818,0:
6140 POKE 232,0: POKE 233,3: REM SHAPE TABLE POINTER.
6150 HCOLOR= 3: SCALE= 1: ROT= 0
6160 RETURN

```



Initializing Arrays With Specific Values

by Philippe Francois
31 Chemin Joseph Aiguier
13274 Marseille Cedex 2 FRANCE

There is no command in APPLESOFT Basic to fill an entire numeric or string array with some numeric or string value.

If an entire array is to be filled in this manner, a loop must be used to assign the specific numeric or string value to each element in the array.

```
10 DIM A$(10),B%(10),C(10)
20 A$="THIS IS A TEST":B%=327767:C=0.009
30 FOR I=0 TO 10
40 : A$(I)=A$
50 : B%(I)=B%
60 : C(I)=C
70 NEXT I
```

This approach, however, becomes inefficient and time consuming as the dimension of the array increases (47 seconds to initialize a 10000 element array!).

An initialization with zero (numeric array) or null (string array) can be made with the **CLEAR** Basic command but, unfortunately, this command clears **all** simple or array variables.

This short machine language routine efficiently **initializes individual numeric/string arrays** with zero/null or with a value specified by a numeric/string variable (a 10000 element array is filled in only 2.5 seconds).

SYNTAX:

We use the ampersand (&) command

1. zero/null initialization

& Z (<name of first array to initialize>, <name of second array>,...), e.g.

& Z (A\$,B%,C,F\$)

2. specific value initialization

& V (<name of specific variable>,<name of first array to initialize with the specific variable>,<name of second array to initialize with the same specific value>,.../<name of other specific value>,<name of first array>,...)

& V ((I\$),A\$,B\$/(G),A,B)

INITIA operates transparently from within an Applesoft program. The program itself begins at location hex \$300 (\$300-\$3F4 is available for short machine language programs) and runs with an Apple of any size.

The ampersand command must be initialized to \$4C \$00 \$03 (JUMP \$0300) with **POKE 1013,76:POKE 1014,0:POKE 1015,3** at the beginning of your program. The INITIA DEMO program illustrates the use of INITIA from Applesoft.

```
0000 1 PAG
0000 2 ;
0000 3 ;
0000 4 ;
0000 5 ; *****
0000 6 ; *****
0000 7 ; *****
0000 8 ; *****
0000 9 ; *****
0000 10 ; *****
0000 11 ; *****
0000 12 ; *****
0000 13 ; *****
0000 14 ; *****
0000 15 ; *****
0000 16 ; *****
0000 17 ; AN APPLE II UTILITIE WHICH INITIALIZE
0000 18 ; NUMERIC OR STRING ARRAYS WITH ZERO/NULL OR SPECIFIC VALUES
0000 19 ;
0000 20 ;
0000 21 ; ORG $300
0000 22 ;
0000 23 ;
0000 24 ; EQUATES...
0000 25 ;
0000 26 ENDST EPZ #00 ; ADDR. OF THE LAST ARRAY ELEMENT
0000 27 LEN EPZ ENDST+#02 ; LENGTH OF EACH ARRAY ELEMENT
0000 28 ADR EPZ LEN+#01 ; STORAGE AREA FOR VARIABLE
0000 29 ;
0000 30 ; APPLE EQUATES...
0000 31 ;
0000 32 NAME EPZ #01 ; STATUS NAME VARIABLE
0000 33 POINTL EPZ #00 ; POINTER TO THE START OF THE
0000 34 POINTH EPZ POINTL+#01 ; STORAGE AREA OF THE VARIABLE
0000 35 ARRAYS EPZ #04 ; POINTER TO THE FIRST ELEMENT
0000 36 ARRAVH EPZ ARRAVH+#01 ; OF THE ARRAY
0000 37 CHRG EPZ #01 ; ADVANCE TEXT POINTER AND LOAD
0000 38 ; NEXT CHARACTER IN THE A REG.
0000 39 RELOAD EPZ #07 ; LOAD CURRENT CHARACTER ANOTHER
0000 40 ; TIME IN ACCUMULATOR
0000 41 GETVAR EQU #0FE3 ; FIND THE NAME,THE STATUS
0000 42 ; AND THE STARTING LOCATION OF THE
0000 43 ; STORAGE AREA OF SIMPLE VARIABLE
0000 44 CHECKC EQU #DEBE ; CHECK WHETHER THE TEXT POINTER
0000 45 ; POINTS TO A COMMA
0000 46 GETARR EQU #F7D9 ; FIND THE NAME,THE STATUS
0000 47 ; AND THE STARTING LOCATION OF THE
0000 48 ; STORAGE AREA OF ARRAY
0000 49 CHECKL EQU #DEBB ; CHECK LEFT PARENTHESIS
0000 50 CHECKR EQU #DEBB ; CHECK RIGHT PARENTHESIS
0000 51 ERROR EQU #DEC9 ; ERROR "SYNTAX ERROR"
0000 52 SLASH EPZ #CB ; #CB IS THE SLASH TOKEN
0000 53 ;
0000 54 ;
0000 55 ; CHECK IF THE FIRST CHARACTER AFTER "&" IS "U" OR "Z"
0000 56 ; AND JUMP TO THE APPROPRIATE ENTRY POINT
0000 57 ;
0000 C95A 58 CMP #'Z'
0000 F007 59 BEQ INITZ
0000 C956 60 CMP #'U'
0000 F03D 61 BEQ INITU
0000 4CC9DE 62 JMP ERROR
0000 63 ;
0000 64 ; IT'S A "ZERO" INITIALIZATION
0000 65 ;
0000 66 ; ENTRY POINT: & Z (A$,B%,C)
0000 67 ;
0000 68 INITZ:
0000 69 ;
0000 70 ; CHECK WHETHER THE TEXT POINTER POINTS TO A LEFT PARENTHESIS
0000 71 ;
0000 20B100 72 JSR CHRG
0000 F02F 73 BEQ ERR
0000 C928 74 CMP #'('
0000 D0F7 75 BNE INITZ
0000 76 BEGINZ:
0000 77 ;
0000 78 ; SEARCH THE ADDRESS OF THE FIRST AND THE LAST ARRAY ELEMENTS
0000 79 ;
0000 20B100 80 JSR CHRG
0000 20D9F7 81 JSR GETARR
0000 20C303 82 JSR SEARCH
0000 A000 83 LDY #000
0000 84 LABEL2:
0000 85 ;
0000 86 ; PUT ZERO IN ALL ELEMENTS OF THE ARRAY
0000 87 ;
0000 98 ;
0000 99 ;
0000 20B100 88 JSR CHRG
0000 20D9F7 89 JSR GETARR
0000 E694 90 INC ARRAVH
0000 D002 91 BNE LABEL1
0000 E695 92 INC ARRAVH
```

```
0328 A594 93 LABEL1 LDA ARRAVH
032A C500 94 CMP ENDST
032C D0F1 95 BNE LABEL2
032E A595 96 LDA ARRAVH
0330 C501 97 CMP ENDST+#01
0332 D0EB 98 BNE LABEL2
0334 99 ;
0334 100 ; IS IT ANOTHER ARRAY TO INITIALIZE ?
0334 101 ;
0334 20B700 102 JSR RELOAD
0337 C92C 103 CMP #'/'
0339 F0D9 104 BEQ BEGINZ
033B C929 105 CMP #'>'
033D F003 106 BEQ ENDZ
033F 4CC9DE 107 ERR JMP ERROR
0342 108 ;
0342 109 ;
0342 110 ENDZ:
0342 4CB100 111 JMP CHRG
0345 112 ;
0345 113 ; INITIALIZATION TO A NON-ZERO VALUE
0345 114 ;
0345 115 ; ENTRY POINT: & U ((I$),A$,B$/(I$),A$,B%/(I),A,B)
0345 116 ;
0345 117 ;
0345 118 INITU:
0345 20B100 119 JSR CHRG
0348 F0F5 120 BEQ ERR
034A C928 121 CMP #'('
034C D0F7 122 BNE INITU
034E 123 BEGINU:
034E 124 ;
034E 125 ; SEARCH THE VARIABLE TYPE
034E 126 ;
034E 20B100 127 JSR CHRG
0351 20BDEE 128 JSR CHECKL
0354 20E3DF 129 JSR GETVAR
0357 A581 130 LDA NAME
0359 300A 131 BMI INT
035B A582 132 LDA NAME+#1
035D 100C 133 BPL REAL
035F A903 134 STRING LDA #003
0361 8502 135 STA LEN
0363 D00A 136 BNE LABEL3
0365 A902 137 INT LDA #002
0367 8502 138 STA LEN
0369 D004 139 BNE LABEL3
036B A905 140 REAL LDA #005
036D 8502 141 STA LEN
036F 142 LABEL3:
036F 143 ;
036F 144 ; AND COPY ITS VALUE IN LOCATIONS BEGINNING AT 'ADR' ADDRESS
036F 145 ;
036F A200 146 LDX #000
0371 A002 147 LDY #002
0373 B198 148 STORE LDA (POINTL),Y
0375 9503 149 STA ADR,X
0377 C8 150 INY
0378 E8 151 INX
0379 E482 152 CPX LEN
037B D0F6 153 BNE STORE
037D 154 ;
037D 155 ; NOW SEARCH THE ARRAY POINTERS
037D 156 ;
037D 20BDEE 157 JSR CHECKR
0380 158 LABEL6:
0380 20B100 159 JSR CHRG
0383 20D9F7 160 JSR GETARR
0386 20C303 161 JSR SEARCH
0389 162 LABEL5:
0389 163 ;
0389 164 ; COPY THE VARIABLE VALUE IN EACH ELEMENT OF THE ARRAY
0389 165 ;
0389 A200 166 LDX #000
038B A000 167 LDY #000
038D B503 168 LABEL4 LDA ADR,X
038F 9194 169 STA (ARRAVH),Y
0391 E8 170 INY
0392 C8 171 INX
0393 C482 172 CPY LEN
0395 D0F6 173 BNE LABEL4
0397 1B 174 CLC
0398 A594 175 LDA ARRAVH
039A 6502 176 ADC LEN
039C 8594 177 STA ARRAVH
039E A595 178 LDA ARRAVH
03A0 6900 179 ADC #000
03A2 8595 180 STA ARRAVH
03A4 C501 181 CMP ENDST+#01
03A6 D0E1 182 BNE LABEL5
03A8 A594 183 LDA ARRAVH
03AA C500 184 CMP ENDST
```

continued on page 187

APPLE UTILITIES

Apple Checker 3.0

by Ken McCandless
2747 Inspiration Drive
Colorado Springs, CO 80917

Due to numerous requests by Nibble readers that the APPLE CHECKER program ignore spaces and hidden control characters in BASIC programs, I am providing this new version. My main goal was to revise the original APPLE CHECKER to provide this capability, but this was quickly cast aside as the full complexity of the problem became clear. To all of you who entered the original program and are facing the entry of this one, I offer my apologies.

This new version of APPLE CHECKER is **easier to use** and provides a corrected CHECKSUM routine which will catch systematic transposition errors; i.e., the routine is sensitive to the location of each byte. I wish to thank Ken Wetzel of Ridgecrest, CA for his letter bringing to light a problem of data transposition, and his suggested change to correct it. Thanks again, Ken.

HOW TO RUN APPLE CHECKER 3.0

After you have keyed in APPLE CHECKER 3.0, do a **BSAVE APPLE CHECKER 3.0, A\$803, L\$257**. To execute the program:

1. For BASIC, activate the basic language
2. BRUN CHECK CODE 3.0
3. LOAD (BLOAD) the program to be checked
4. CALL 25 (must immediately follow step #3).
5. After you are finished with the APPLE CHECKER 3.0 program, you should do an FP or INT before continuing on to other things. (This will reset BASIC back to its normal state.)

NOTE: If you are checking a BINARY program, be sure to load it with the address specification option containing a value of at least **\$B00**. This will insure that the BINARY program does not load on top of the APPLE CHECKER program and produce disastrous results. As an example: **BLOAD PROGRAM, A\$B00**.

The APPLE CHECKER program must be initialized before you load your BASIC program. If you are doing multiple consecutive runs of this program and are switching between APPLESOFT and INTEGER BASIC, you must re-initialize the program following each BASIC language switch. Initialization is accomplished when the program is BRUN but can be repeated, if APPLE CHECKER 3.0 is in memory, by doing a CALL 2051 or for APPLESOFT by typing an "&" followed by a RETURN. If no BASIC language switch is required for your multiple runs, just LOAD the program to be checked and do a CALL 25.

If you think you will never have a need for doing multiple consecutive runs of the APPLE CHECKER program, you can eliminate the requirement for doing step 5 by changing the byte at \$A0E to a HEX D3. (BLOAD APPLE CHECKER 3.0, enter the monitor by a CALL -151, type in A0E:D3 followed by a return, and

then resave the program.) This change will not alter the performance of the program but merely causes DOS to do a coldstart of the currently active BASIC language, thereby doing the required cleanup.

HOW IT WORKS

When you BRUN the program all the necessary parameters are set so that neither of the BASICs will load the BASIC program on top of APPLE CHECKER. A jump vector is built at HEX 19 (decimal 25) for easy execution of the program and the AMPERSAND (&) vector is set for easy re-initialization from APPLESOFT.

Once you've LOADED (or BLOADED) the program to be checked and issued a CALL 25, the check code development portion of the program begins. The first thing that the program does is to clear out the work areas, build a table for indirection into DOS and then display the program's identifying header. Next, the program grabs the file type and file name from DOS and displays this information if all is correct. The beginning and ending (length for BINARY files) is picked up from DOS next. Now the work really begins.

For BINARY files, the ending address must be calculated so that the program will know when to stop generating checksums. Once this is accomplished, the program continues into the checksum development loop until it is finished.

For APPLESOFT programs, all of the next line address pointers (the first two bytes of every encoded program line) are zeroed. This is done to eliminate problems with ignoring imbedded spaces and control characters. The program continues by calculating the program's length and ending address, and then branching to the checksum development loop.

For INTEGER BASIC programs, all the line lengths (the first byte of every encoded program line) are zeroed. Again, this is to eliminate problems with ignoring spaces and control characters. The program continues by calculating the program's length and ending address, and then goes to the checksum loop.

This loop gets a byte of the program to be checksummed, and then determines whether it is a BINARY or BASIC file. If it is a BINARY file, it checksums the byte and continues to the end of program check. If it is a BASIC file the program must determine whether it is a space or control character, and if it is either, it will be ignored (with the exception of Control-D). All other characters must be checksummed.

If this is not the end of the program to be checksummed, the checksum development loop continues until finished. The program then outputs the length (number of bytes checked, excluding the ignored spaces and control characters) and the checksum. Finally the program cleans up PAGE ZERO and returns to BASIC. Because of all the manipulations to the BASIC program, your loaded program will not be accessible in memory after a run of APPLE CHECKER 3.0. You must reload it to do any checking or to execute it.

If an error is detected during execution of APPLE CHECKER 3.0, the bell will sound and "ERR" will be displayed. To retry you must

reload the program to be checked and do a CALL 25.

ERROR CHECKING

Just as in the original version, the error checking is quite simple. The program picks up the file type from the DOS file manager and if the type is a zero you will get an error. Doing a CATALOG will set the file type to a zero. Other DOS functions also set this variable. For this reason you must not execute other DOS functions after the load of the program to be checked before execution of APPLE CHECKER 3.0; disastrous results could take place if you do so.

Another condition that could produce an error occurs when you load a BASIC file, and then switch to the other language before executing APPLE CHECKER 3.0, whether you have initialized APPLE CHECKER or not.

The final condition that could produce an error is to execute APPLE CHECKER 3.0 without loading anything. In this case APPLE CHECKER thinks that itself was the last file loaded and will halt.

MODIFICATIONS FOR A 32K SYSTEM

The program is written for a 48K system with DOS residing in its standard place. If you have a 32K system the following changes must be made to make APPLE CHECKER 3.0 work. Also, if you have DOS loading in some location other than either a standard 48K or a 32K system, you must change the same locations below for your system's values.

1. BLOAD APPLE CHECKER 3.0
2. Enter the monitor (CALL -151)
3. 855:75
4. 862:6A
5. A47:6A
6. A4D:6A
7. A53:6A
8. A59:6A
9. BSAVE APPLE CHECKER 3.0, A\$803, L\$257

FINAL NOTE: Remember, REM statements are valid BASIC statements and must be included in your program to obtain the correct checksum. All spaces and control characters, except CONTROL-D, in your BASIC program will be ignored and not included in the checksumming process, so don't worry about the number of spaces in the REMs or any string literals. As an example, the following will produce identical checksums:

10 REM INITIALIZATION ROUTINE

20 A="A B C D E"

AND

10 REMINITIALIZATION ROUTINE

20 A="ABCDE"

Since both of the above will produce identical checksums, it should be noted that line 20 in each program is different and will produce different results if used in any comparison, so be alert to spacing in string assignment statements.



Apple Checker 3.0 (Cont.)

SOURCE FILE: APPLE CHECKER 3.0

```

0000: 1 *****
0000: 2
0000: 3 APPLE CHECKER
0000: 4 VERSION 3.0
0000: 5
0000: 6 ASSEMBLER: APPLE 6502 ASSEMBLER/EDITOR
0000: 7 AUTHOR: KEN MCCANDLESS
0000: 8 DATE: MAY 18, 1982
0000: 9
0000: 10 THIS PROGRAM GENERATES A CHECKSUM FOR APPLESOFT,
0000: 11 INTEGER OR BINARY PROGRAMS. (SPACES AND CONTROL
0000: 12 CHARACTERS; EXCEPT CONTROL-D, ARE IGNORED.)
0000: 13
0000: 14 TO EXECUTE:
0000: 15 1. BRUN CHECK CODE
0000: 16 (OR BLOAD IT AND THEN CALL 2051)
0000: 17 2. (B)LOAD THE PROGRAM FOR CHECKSUMMING
0000: 18 3. CALL 25 (IF IN THE MONITOR DO 196)
0000: 19 *****
0000: 20
0000: 21
0000: 22 COPYRIGHT (C) 1982 BY
0000: 23 MICRO-SPARC, INC.
0000: 24
NEXT OBJECT FILE NAME IS APPLE CHECKER 3.0.OBJ30
0B03: 25 ORG $B03
0B03: 26
0000: 27 CKCD EQU $00 ;CHECK CODE
0001: 28 BYTE EQU $01 ;BEGINNING OF PROGRAM
0001: 29 BEG EQU $01 ; POINTER
0003: 30 END EQU $03 ;END OF PROGRAM POINTER
0005: 31 LEN EQU $05 ;LENGTH OF PROGRAM
0007: 32 FTYPE EQU $07 ;FILE TYPE (0=1,2=A,4=B)
0008: 33 DECVL EQU $08 ;WORK BYTE FOR ASOFT
000A: 34 HLD EQU $0A ;WORK ADDR
0019: 35 ENTRY EQU $19 ;ROUTINE ENTRY VECTOR
0024: 36 CHORIZ EQU $24
004A: 37 INTLOW EQU $4A ;INTEGER BASIC LOMEM
004C: 38 INTH EQU $4C ;INT PGM END ADDR
0067: 39 ASFTL EQU $67 ;A-SOFT PGM START ADDR
00AF: 40 ASFTH EQU $AF ;A-SOFT PGM END ADDR
00CA: 41 INTL EQU $CA ;INT PGM START ADDR
00E0: 42 PBEGL EQU $E0
00E6: 43 PBEGH EQU $E6
00EC: 44 PENDL EQU $EC
00F1: 45 SPEED EQU $F1 ;APPLESOFT SPEED
00F2: 46 PENDH EQU $F2
03D0: 47 RTNBAS EQU $3D0 ;DOS WARMSTART VECTOR
03F6: 48 AMPVEC EQU $3F6 ;AMPERSAND VECTOR ADDRESS
0400: 49 LINEO EQU $400 ;FIRST BYTE OF TEXT SCREEN AREA
0B01: 50 BGNASF EQU $B01 ;NEW LOAD POINT FOR ASOFT
AA60: 51 BLEN EQU $AA60 ;FOR 4BK
AA72: 52 BINL EQU $AA72 ;FOR 4BK
AA75: 53 DOSFN EQU $AA75 ;FILE NAME-4BK
B5F6: 54 DOSFT EQU $B5F6 ;FILE TYPE - 4BK
E000: 55 BASICN EQU $E000
0B03: 56
0B03: 57 * MONITOR ROUTINES
0B03: 58
F941: 59 PRINTX EQU $F941
FC58: 60 HOME EQU $FC58
FD8E: 61 CROUT EQU $FD8E
FDED: 62 COUT EQU $FDED
FDDA: 63 PRBYTE EQU $FDDA
FF2D: 64 PRERR EQU $FF2D ;PRINT "ERR"
0B03: 65
0B03: 66 * SETUP FOR ROUTINE
0B03: 67
0B03: 68 INIT CLD
0B04: A9 01 LDA $1
0B06: B5 4A STA INTLOW ;SET INTEGER BASIC LOMEM
0B08: A9 08 LDA $8 ;BGNASF
0B0A: B5 68 STA ASFTL+1 ;SET ASFT BEG ADDR TO $B01
0B0C: B5 4B STA INTLOW+1 ;SET INT LOMEM
0B0E: A9 00 LDA $0
0B10: B0 00 LDA $BGNASF-1 ;MAKE ASFT HAPPY
0B13: A9 4C LDA $4C ;SET UP JUMP ADDR FOR ROUTINE
0B15: B5 19 STA ENTRY
0B17: A9 2A LDA $2A ;START
0B19: B5 1A STA ENTRY+1
0B1B: A9 08 LDA $8 ;START
0B1D: B5 1B STA ENTRY+2
0B1F: A9 03 LDA $3 ;INIT ;SET UP & FOR RE-INIT
0B21: B0 F6 03 STA AMPVEC
0B24: A9 08 LDA $8 ;INIT
0B26: B0 F7 03 STA AMPVEC+1
0B29: 60 RTS
0B2A: 87
0B2A: 88 * ROUTINE STARTING POINT
0B2A: 89
0B2A: 8B 90 START CLD
0B2B: A2 00 LDA $0
0B2D: A0 08 LDA $8 ;$08
0B2F: 96 00 STX CKCD,Y ;CLEAR WORK AREAS
0B31: B8 94 DEY
0B32: 10 FB BPL LPCLR
0B34: B0 42 0A 96 LPP LDA PDATA,X ;BUILD INDIRECT TABLE
0B37: 95 E0 STA PBEGL,X
0B39: E8 98 INX
0B3A: E0 1A 99 CPX $26
0B3C: D0 F6 100 BNE LPP
0B3E: 101
0B3E: 102 * OUTPUT HEADER
0B3E: 103
0B3E: 20 58 FC JSR HOME
0B41: 20 BE FD JSR CROUT
0B44: A9 0D LDA $13
0B46: B5 24 STA CHORIZ
0B48: A2 15 LDX $21
0B4A: B0 2C 0A 109 LP1 LDA INP,X
0B4D: 20 ED FD JSR COUT
0B50: CA 111 DEX
0B51: 10 F7 112 BPL LP1
0B53: 113
0B53: 114 * DO FILE TYPE AND SET UP PARAMETERS
0B53: 115
0B53: AD F6 B5 116 LDA DOSFT ;GET FILE TYPE
0B56: F0 65 117 BEQ GETAGN ;NOT A VALID TYPE
0B58: 0A 118 ASL A ;DISPOSE OF LOCK BIT
0B59: 4A 119 LSR A
0B5A: 4A 120 LSR A ;CHANGE INTEGER FROM 1 TO 0

```

```

0B5B: 0A 121 ASL A
0B5C: B5 07 122 STA FTYPE ;SAVE IT FOR LATER
0B5E: 123 * OUTPUT FILE NAME
0B5E: A2 00 124 LDX $0 ;OUTPUT FILE NAME
0B60: B0 75 AA 125 FNOUT LDA DOSFN,X
0B63: 20 ED FD 126 JSR COUT
0B66: E8 127 INX
0B67: E0 1E 128 CPX $30
0B69: D0 F5 129 BNE FNOUT
0B6B: 130 * OUTPUT FILE TYPE
0B6B: A2 06 131 LDX $6
0B6D: B0 25 0A 132 OFTYP LDA CTYP,X
0B70: 20 ED FD 133 JSR COUT
0B73: CA 134 DEX
0B74: 10 F7 135 BPL OFTYP
0B76: A5 07 136 LDA FTYPE ;BUILD AND OUTPUT FILE TYPE
0B78: 4A 137 LSR A
0B79: 18 138 CLC
0B7A: 69 C0 139 ADC $C0
0B7C: C9 C0 140 CMP $C0
0B7E: D0 02 141 BNE OTTP
0B80: A9 C9 142 LDA $C9
0B82: 20 ED FD 143 OTTP JSR COUT
0B85: 144 * BUILD INDIRECT ADDRESSES
0B85: A6 07 145 LDX FTYPE
0B87: A1 E0 146 LDA (PBEGL,X) ;GET APPROPRIATE BEG LO ADDR
0B89: B5 01 147 STA BEG
0B8B: A1 E6 148 LDA (PBEGH,X) ;GET BEG HI ADDR
0B8D: B5 02 149 STA BEG+1
0B8F: A1 EC 150 LDA (PENDL,X) ;GET APPROPRIATE END LO ADDR
0B91: B5 03 151 STA END
0B93: A1 F2 152 LDA (PENDH,X) ;GET END HI ADDR
0B95: B5 04 153 STA END+1
0B97: E6 08 154 INC DECVL ;TO REDUCE END ADDR BY 1
0B99: A5 07 155 LDA FTYPE
0B9B: C9 04 156 CMP $4 ;BIN FILE?
0B9D: D0 27 157 BNE BASIC ;NO
0B9F: 158
0B9F: 159 * SET UP FOR BINARY FILE
0B9F: 160
0B9F: A5 02 161 LDA BEG+1 ;IF LOAD ADDR HI IS SAME
0BA1: C9 08 162 CMP $CINIT ;THEN NOT A BINARY FILE!
0BA3: F0 18 163 BEQ GETAGN
0BA5: A5 03 164 LDA END
0BA7: B5 05 165 STA LEN
0BA9: A5 04 166 LDA END+1
0BAB: B5 06 167 STA LEN+1
0BAD: 18 168 CLC
0BAE: A5 01 169 LDA BEG ;COMPUTE END ADDR
0BB0: 65 05 170 ADC LEN ;END=BEG+LEN-1
0BB2: B5 03 171 STA END
0BB4: A5 02 172 LDA BEG+1
0BB6: 65 06 173 ADC LEN+1
0BB8: B5 04 174 STA END+1
0BBA: 4C 3A 09 175 JMP CONT
0BBD: 176
0BBD: 177 * RESTART
0BBD: 178
0BBD: 20 BE FD 179 GETAGN JSR CROUT
0BC0: 20 2D FF 180 JSR PRERR ;OUTPUT "ERR"
0BC3: 4C DC 09 181 JMP CLNUP ;CLEAN UP THE ZERO PAGE MESS & QUIT
0BC6: 182
0BC6: 183 * SET UP FOR BASIC FILE
0BC6: 184
0BC6: A5 07 185 BASIC LDA FTYPE
0BC8: F0 32 186 BEQ IBASIC ;0=INTEGER 2=ASOFT
0BCA: AD 00 E0 187 LDA BASICN ;IS APPLESOFT ACTIVE?
0BCD: C9 4C 188 CMP $4C
0BCE: D0 EC 189 BNE GETAGN
0BD1: A0 00 190 CLRADD LDY $0
0BD3: B1 01 191 LDA (BEG),Y ;GET NEXT LINE ADDR LO
0BD5: B5 0A 192 STA HLD ;SAVE IT
0BD7: C8 193 INY
0BD8: B1 01 194 LDA (BEG),Y ;GET NEXT LINE ADDR HI
0BDA: B5 0B 195 STA HLD+1 ;SAVE IT ALSO
0BDC: A9 00 196 LDA $0 ;TO CLEAR ADDR
0BDE: A8 197 TAY ;RESET Y
0BDF: F1 01 198 STA (BEG),Y ;0 NEXT ADDR PTR LO
0BE1: C8 199 INY
0BE2: 91 01 200 STA (BEG),Y ;0 NEXT ADDR PTR HI
0BE4: A5 0A 201 LDA HLD ;SET UP FOR NEXT ONE
0BE6: B5 01 202 STA BEG
0BE8: A5 0B 203 STA HLD+1
0BEA: B5 02 204 STA BEG+1
0BEC: D0 E3 205 BNE CLRADD
0BEE: A6 07 206 LDX FTYPE
0BF0: A1 E0 207 LDA (PBEGL,X) ;REBUILD BEG ADDR
0BF2: B5 01 208 STA BEG
0BF4: A1 E6 209 LDA (PBEGH,X)
0BF6: B5 02 210 STA BEG+1
0BF8: E6 08 211 INC DECVL ;REDUCE END ADDR BY 2
0BFA: D0 3E 212 BNE CONT
0BFC: 213
0BFC: AD 00 E0 214 IBASIC LDA BASICN ;IS INTEGER ACTIVE?
0BFF: C9 20 215 CMP $20
0901: D0 BA 216 BNE GETAGN
0903: 217 * ZERO OUT INTEGER LINE LENGTHS
0903: A0 00 218 LDY $0
0905: A5 01 219 INTLP LDA BEG
0907: B5 0A 220 STA HLD
0909: A5 02 221 LDA BEG+1
090B: B5 0B 222 STA HLD+1
090D: C5 04 223 CMP END+1 ;CHECK IF FINISHED
090F: 90 17 224 BLT INTCLR ;NOT DONE-CLEAR IT
0911: F0 02 225 BEQ INTLW ;HI ADDR THE SAME-CHECK LO ADDRS
0913: B0 06 226 BGE INTDN ;(JUST FOR MURPHY)
0915: A5 0A 227 INTLW LDA HLD ;CHECK LO ADDR
0917: C5 03 228 CMP END
0919: 90 00 229 BLT INTCLR ;NOT DONE - CLEAR IT
091B: A6 07 230 INTDN LDX FTYPE ;RESET BEG POINTER
091D: A1 E0 231 LDA (PBEGL,X)
091F: B5 01 232 STA BEG
0921: A1 E6 233 LDA (PBEGH,X)

```

continued on page 188

GO — A GREETING PROGRAM

GO — A Greeting Program

by Chris Glenn
96 Chestnut Hill Rd.
Norwalk, CT 06851

APPLE DOS is truly wizardly . . . so fast, so flexible. The trouble is, getting DOS to do its stuff is often an exacting, frustrating chore. Take my case: like thousands of other multi-thumbed APPLE freaks, when faced with entering 30 or 40 characters of code without any mistakes so that DOS can make its magic, I quail. Therein lies the genesis of GO.

GO is a disk greeting program written in APPLESOFT for 48K systems with DOS 3.3. Using GO, the following operations can be performed with only a keystroke or two:

RUN
LOAD
BRUN
BLOAD
EXEC
LOCK
UNLOCK
RENAME
DELETE
VERIFY
CHECK DISK SPACE USED AND FREE
CHECK FILE SIZE IN DISK SECTORS
CHECK BINARY FILE ADDRESS AND LENGTH

Typical of the savings GO makes possible is this common situation: to **DELETE** a **LOCKed** file named "**BAD CHECKS — 1971**" ordinarily would require typing **UNLOCK BAD CHECKS — 1971 <RETURN> DELETE BAD CHECKS — 1971 <RETURN>** . . . a total of 50 keystrokes. But using GO, the same procedure requires only four keystrokes. Similar economies are realized with each of GO's other functions.

GO has some other advantages. For one thing, it's quick to get ready. A power-up boot to GO takes but 11 seconds. With DOS already installed, type **RUN GO <RETURN>** and GO will be at your command just 7.5 seconds later. Another plus is screen clarity: even though you may have 105 files on a given disk (the DOS 3.3 maximum), GO never splits the screen, truncates filenames, or packs the display so tightly it becomes illegible.

HOW TO USE GO

GO is very user-friendly and completely self-guiding. There's only one thing to remember: **you can ALWAYS press <RETURN> to "back up" to the previous command level or to re-CATALOG the disk.** Other than that, it's just a matter of following the prompts.

To get started, **LOAD GO** into memory, and then **SAVE GO** to the disk you want to use . . . either a newly **INITIALIZED** disk or one that is already storing other files. Now **BRUN** the **MASTER CREATE** program found on the **SYSTEM MASTER** disk, and enter **GO** when asked for the greeting program's file name. From now on, your disk will **RUN GO** when it's booted.

In operation, GO begins by displaying a screenful of the disk **CATALOG** and pausing for a keypress (presuming more than 16 files are stored on the disk). A prompt line appears at the top of the screen: **<RETURN>=THIS SCREEN <SPACE>=MORE FILES.** Press [RETURN] if you see the file name you want on the screen, or hit the [SPACE BAR] for more of the **CATALOG**. If there are fewer than 16 files on the disk, screen setup is carried out automatically, without the pause or the prompt.

Now the **DISK TITLE** appears at the top of the screen. You can easily change the **DISK TITLE** to reflect the general nature of the files on the disk (i.e. "**GAMES**", "**UTILITIES**", "**HOME FINANCE**", etc.). Just change **T\$** in line 20 of the GO program. The **DISK TITLE** may be up to 40 characters long, and is self-centering when displayed.

Next, a **LETTER** appears before each file name and you'll see a prompt line below the catalog reading: **FILE CHOICE? (OR #=SPACE X=END)**

- Press a **LETTER** to indicate a **FILE CHOICE**.
- Press [#] to display **DISK SPACE** information, then hit any key to continue.
- Press [X] to **END** the GO program.
- Or press [RETURN] to begin the **CATALOG** again.

After picking a file, a prompt displays a selection of **COMMAND** choices:

CHOSEN FILE'S TYPE:	COMMAND PROMPT READS:			
APPLESOFT or INTEGER	RUN	LOAD	TEND	SIZE
TEXT	EXEC	TEND	SIZE	
BINARY	RUN	LOAD	TEND	ADDRESS SIZE

Just press the first letter of the **COMMAND** you want, or press [RETURN] to go back to the **FILE CHOICE** prompt.

If you choose **TEND**, a prompt offering **LOCK UNLOCK RENAME DELETE VERIFY** will appear. Press the first letter of the function you want, or press [RETURN] to go back to the **COMMAND** prompt.

Some notes on using the **TEND** functions: **VERIFY**, if successful, will return you to the functions prompt without any message. **RENAME** accepts only legal DOS file names . . . press [RETURN] when you've finished entering the new name. You'll be prompted to reconfirm any **RENAME** or **DELETE** command before it's executed . . . answer **Y** for **YES** or **N** for **NO**.

Selecting the **SIZE** command will display the size in disk sectors of the chosen file. Hit any key to return to the **COMMAND** prompt.

If you've selected a **BINARY** file and the command is **ADDRESS**, a prompt will ask you to reconfirm (Y/N) before attempting the find. The **ADDRESS** routine may fail with some **BINARY** files, causing the system to hang. The reconfirmation prompt reminds you to boot the disk if this happens. Even a successful find must be followed by a boot because the **ADDRESS** routine clobbers GO . . . but a good find will give you the flashing cursor, and pressing any key should immediately boot you back to GO.

Should an **ERROR** message appear, you've most likely left the drive door open or some-

how gotten a different disk in the drive since last running GO. Or perhaps you called an **INTEGER BASIC** program, but don't have the language available. An unsuccessful **VERIFY** will also generate a message. Whatever, correct the condition and hit any key to **RUN GO** again.

REMEMBER: [RETURN] is GO's "failsafe" key...you can **ALWAYS** press it to "back up" to the previous prompt or to begin the **CATALOG** again.

GO: THE WAY IT GOES

Here's a general description of what happens within the GO program.

SCREEN TITLE — LINE 20

T\$ stores the **DISK TITLE** which is displayed on the top screen line. Change it to suit. This option is placed first in the program for easy access. GO now jumps ahead over the program subroutines, which will be discussed later.

SETUP — LINES 270-340

(280-300) **POKE** in a routine permitting the DOS **CATALOG** function to be interrupted by pressing **RETURN**. Perform some **POKES** restricting the number of files that can be **CATALOGed** per screen. **DIMension** the arrays used in the file **SIZE** and **RENAME** **INPUT** routines. Bait the **ERRor** trap.

(310-320) **GOSUB 230** to initialize variables. Normalize the screen with **NOMON C,I,O** and **TEXT** commands. Display **<RE-**

TURN>=THIS SCREEN, etc. prompt on screen line 1.

(330) Display **CATALOG**. (340) Display **DISK TITLE**.

(350) Read the screen and store the file size information for each file in array variables, so the data will be available if called by GO's **SIZE** command later.

(360) Now blank out the size data on the screen and display GO's file indicator letters in those spaces. Determine last file indicator letter on display and screen vertical position of prompt line.

FILE NAME/TYPE — LINES 370-470

(380) Setup to determine the selected file's name and type. (390) Display **FILE CHOICE?**, etc. prompt. Get selection and (400-420) operate accordingly. (430) Find the screen line containing the selected filename and get the address of the first space on that line. (440) Determine the chosen file's type (A, I, B or T only). (450) Find the position of the last character of the chosen filename. (470) Read the chosen filename, from first character to the just-determined last character. Highlight the chosen filename in position on the screen.

COMMAND — LINES 480-650

(490-520) Determine and display the proper combination of command options indicated by the chosen file's type from among **RUN**, **LOAD**, **EXEC**, **TEND**, **ADDRESS** and **SIZE**. (530-610) Get the user's command

GO — A Greeting Program (Cont.)

choice. (620) Do some screen-and-machine normalizing. Set for BRUN or BLOAD, if indicated. (630) Display and execute selected command. (640) Set vertical screen position for ENDING the GO program.

DISK SPACE — LINES 650-690

(660) Clear the prompt line and print DISK SPACE:, then (670-680) READ DATA and POKE in values used by the routine. (690) Determine the sectors free and used and display the information on the prompt line. Await a keypress then GOSUB 300 to reset GO variables the DISK SPACE routine wipes out. RETURN to the main program at line 310 with a fresh CATALOG display.

SIZE — LINES 700-720

(710) Clear the prompt line and display the file SIZE information retrieved from storage in the array initialized earlier. (720) Await a keypress and return to the command prompt at line 490.

FILE TENDING — LINES 730-1040

LINES 740-800 — Set up for file tending and offer LOCK UNLOCK RENAME DELETE VERIFY prompt. Get the chosen command and go to the proper line to execute it.

LINES 810-820 — VERIFY — If VERIFY is successful, return to the file tending prompt line without any message. If VERIFY fails, go to the ERROR handling routine.

LINES 830-880 — LOCK and UNLOCK — Reject attempts to LOCK an already LOCKed file, and vice-versa. LOCK or UNLOCK the selected file, and either add or remove the FILE LOCKED indicator asterisk from the screen display, as the case may be.

LINES 890-970 — RENAME — (900) POKE in a flashing FILE LOCKED asterisk if the file that's to be RENAMED is LOCKed. (910) Offer the RENAME=prompt, and pass control to the INPUT subroutine at line 40. (More on the INPUT routine later). (920-940) As control returns from the INPUT routine, if the new filename is the same as the old, refuse to execute the RENAME command. Double-check if new filename is OK?, getting a YES or NO answer. (950-970) Execute RENAME, UNLOCKing and then re-LOCKing the file if necessary. Display the highlighted new filename in position on the screen.

LINES 980-1040 — DELETE — (990-100)

Clear the prompt line, display the DELETE confirmation prompt, and POKE in a flashing asterisk if the file to be DELETED is LOCKed. If it is LOCKed, add LOCKED FILE to the DELETE confirmation prompt. (1010-1020) Get a Y/N answer, returning to the file tending functions prompt if it's N. (1030-1040) Execute DELETE, UNLOCKing the file first if necessary. Then re-CATALOG the disk as command passes back to line 350.

ERRORS — LINES 1050-1120

(1060-1070) POKE in the standard APPLE-SOFT ERROR-handling routine. (1080-1110) Display the ERROR code on the prompt line, usually along with a message ... LANGUAGE NOT AVAILABLE (no INTEGER BASIC card?) or I/O (drive door open?) or FILE NOT FOUND (wrong disk in drive?). A bad VERIFY also kicks over to the ERROR routine, causing the message UNABLE TO VERIFY to be displayed. (1120) Whatever the error, wait for a key press, then RUN the GO main program again.

ADDRESS — LINES 1130-1230

(1140-1170) Display the ADDRESS/LENGTH, etc. prompt and await the answer (Y/N). Redirect the ERROR handling routine to a dummy destination. (1180) BLOAD the chosen BINARY file. (1190) Reset LOMEM: to just below HIMEM: and set the high-end variable H used by the routine. This is necessary, but unfortunately destroys the GO program and sometimes DOS. (1200-1210) Find the BLOAD ADDRESS and LENGTH of the chosen file, and display the ADDRESS/LENGTH information on the prompt line. (1220-1230) Wait for a keypress, then execute a PR#6 to restore DOS and GO.

SUBROUTINES — LINES 40-260

LINES 50-210 — INPUT — This routine is used by the RENAME file tending function. It accepts only legal DOS filenames. (50) Set up to receive INPUT by nulling out load information from any previous RENAME. (60) Prevent "backing out" of the screen space allotted for entering the new name. (70) Place the cursor in the current horizontal position.

(80) Get a character. If the first character is RETURN, abandon RENAME and go back to the file tending command prompt. (90) If a comma is entered, reject it. (100) If a forward arrow is entered AND the first character is a letter, advance the horizontal cursor position. (110) If the first character entered is not a letter, reject it.

(120) If RETURN is entered, it signals the end of the new name INPUT and control passes to line 190. (130) If a back arrow is entered, store the affected character in a temporary string array element, since it may be needed again later. (140) If the maximum legal filename length is reached, refuse to accept any more characters. (150-160) These lines concern entry of forward arrows. If the affected character is null (i.e., "") set it to a true space. Retrieve from the temporary array element corresponding to the affected location any character which may be stored there. (170) Reject any attempt to enter a control character. (180) Display the character entered in position on the prompt line and GOTO 60 to fetch another.

(190-200) After a RETURN is encountered, check the entire INPUT line backward to find the first non-blank space (i.e., the last character of the new filename). (210) Read the input space forward, compiling the new filename from the characters stored in each string array element. RETURN with the new filename to the main RENAME routine at line 910.

LINES 220-260 — VARIABLES — the variables GO uses are assigned from a subroutine because some of them have to be reset after being clobbered by the DISK SPACE routine. GO also uses variables freely to speed up execution in FOR...NEXT loops, the RENAME INPUT routine and elsewhere. The two defined functions (250) are these: FN S(R) reads the screen columns containing the DOS CATALOG file size figures before they are wiped out by GO's file selection letters. And FN T(A) is used to center the user DISK TITLE, whatever it may be, on the top line of the screen.

OTHER CONSIDERATIONS

Some cautions: using GO's DISK SPACE function may clobber programs like APPLE-SOFT PROGRAMMER'S ASSISTANT which co-reside in memory. Similarly, using GO's ADDRESS function when another program is co-resident could result in an inaccurate find.

Although GO was written for DOS 3.3, it will also work fine with DOS 3.2...EXCEPT, of course, DISK SPACE information will be inaccurate. Perhaps the clever programmer can rework the values in the DISK SPACE routine to make it work for DOS 3.2. I've also thought that it would be handy to have GO display a text screen disk map as part of the DISK SPACE function. Another project might be to make GO work on systems with less than 48K RAM.

LIST

```

1 REM *****
2 REM      GO
3 REM      BY CHRIS GLENN
4 REM      COPYRIGHT (C) 1982
5 REM      BY MICRO-SPARC INC
6 REM      LINCOLN, NE. 01773
7 REM *****
20 T$ = "GO - DEMONSTRATION
30 GOTO 280
40 REM INPUT
50 C = H:I$ = "": FOR X = H TO H + L:C$(X) = "":T$(X) = ""
: NEXT
60 IF C = H - I THEN C = H
70 HTAB C
80 GET C$(C): IF C$(H) = R$ THEN HTAB I: CALL - B6B: INVERSE
: POP: GOTO 740
90 IF C$(C) = C$ THEN C$(C) = "": GOTO 70
100 IF C$(C) = F$ AND T$(C) > = A$ THEN 120
110 IF C$(H) < A$ OR C$(H) > Z$ THEN 70
120 IF C$(C) = R$ THEN CALL - B6B: GOTO 190
130 IF C$(C) = B$ THEN T$(C - I) = C$(C - I):C = C - I: GOTO
60
140 IF C = H + L + I THEN PRINT B$: GOTO 80
150 IF C$(C) = F$ AND T$(C) = "" THEN T$(C) = S$
160 IF C$(C) = F$ THEN C$(C) = T$(C):C = C + I: GOTO 60
170 IF ASC (C$(C)) < 32 THEN 60
180 VTAB V: HTAB C: PRINT C$(C):C = C + I: GOTO 60

```

```

190 FOR X = C - I TO H STEP - I: IF ASC (C$(X)) + Z < >
B THEN C = X: GOTO 210
200 NEXT
210 FOR X = H TO C:I$ = I$ + CHR$ ( ASC (C$(X)) + Z): NEXT
: RETURN
220 REM VARIABLES (NOTE: B3$ = 3 BLANKS, B40$ = 40 BLANKS
)
230 C = 0:F0 = 40:F1 = 41:F2 = 256:I = 1:P = 3:Q = 4:R = 5
:B = 160:Y = 62:H = 8:L = 29:F = 7:Z = 12B
240 A$ = "A":Z$ = "Z":B3$ = " ":B40$ = " "
250 DEF FN S(R) = PEEK (C + PEEK (F0) + PEEK (F1) * F
2): DEF FN T(C) = INT ((F1 - LEN (T$)) / 2 + .5)
260 D$ = CHR$ (4):R$ = CHR$ (13):F$ = CHR$ (21):B$ = CHR$
(8):S$ = CHR$ (32):C$ = CHR$ (44):B$ = CHR$ (7): RETURN
270 REM SETUP
280 POKE 768,32: POKE 769,12: POKE 770,253: POKE 771,201:
POKE 772,141: POKE 773,208: POKE 774,3: POKE 775,76
290 POKE 776,44: POKE 777,174: POKE 778,76: POKE 779,60: POKE
780,174: POKE 781,0: POKE 44602,0: POKE 44603,3
300 POKE 44452,20: POKE 44605,19: DIM A$(22,6),C$(38),T$(
38): ONERR GOTO 1060
310 GOSUB 230: PRINT D$"NOMON C,I,O": TEXT : HOME
320 INVERSE : PRINT B40$: VTAB I: HTAB I: PRINT "<RETURN
>THIS SCREEN <SPACE>MORE FILES": POKE 34,2
330 HOME : NORMAL : PRINT R$: VTAB I: PRINT D$"CATALOG":V
= PEEK (37) - 2: VTAB 24: PRINT CHR$ (10)

```

continued on page 189

QUICK SORT

Quick Sort

by C. Bongers
Erasmus University
Postbox 1738
3000 DR Rotterdam
The Netherlands

INTRODUCTION

In a simulation study I was working on recently, it was necessary to sort about 1000 real numbers at each trial. It readily became apparent that the easiest solution, i.e. a BASIC bubblesort subroutine, was totally unacceptable because of its tremendous consumption of CPU time. **Using bubblesort, it took more than 2 1/4 hours to sort 1000 reals.** Next I tried a BASIC sort routine based on quick sort, but although CPU time was cut down to 200 secs, this still seemed rather long.

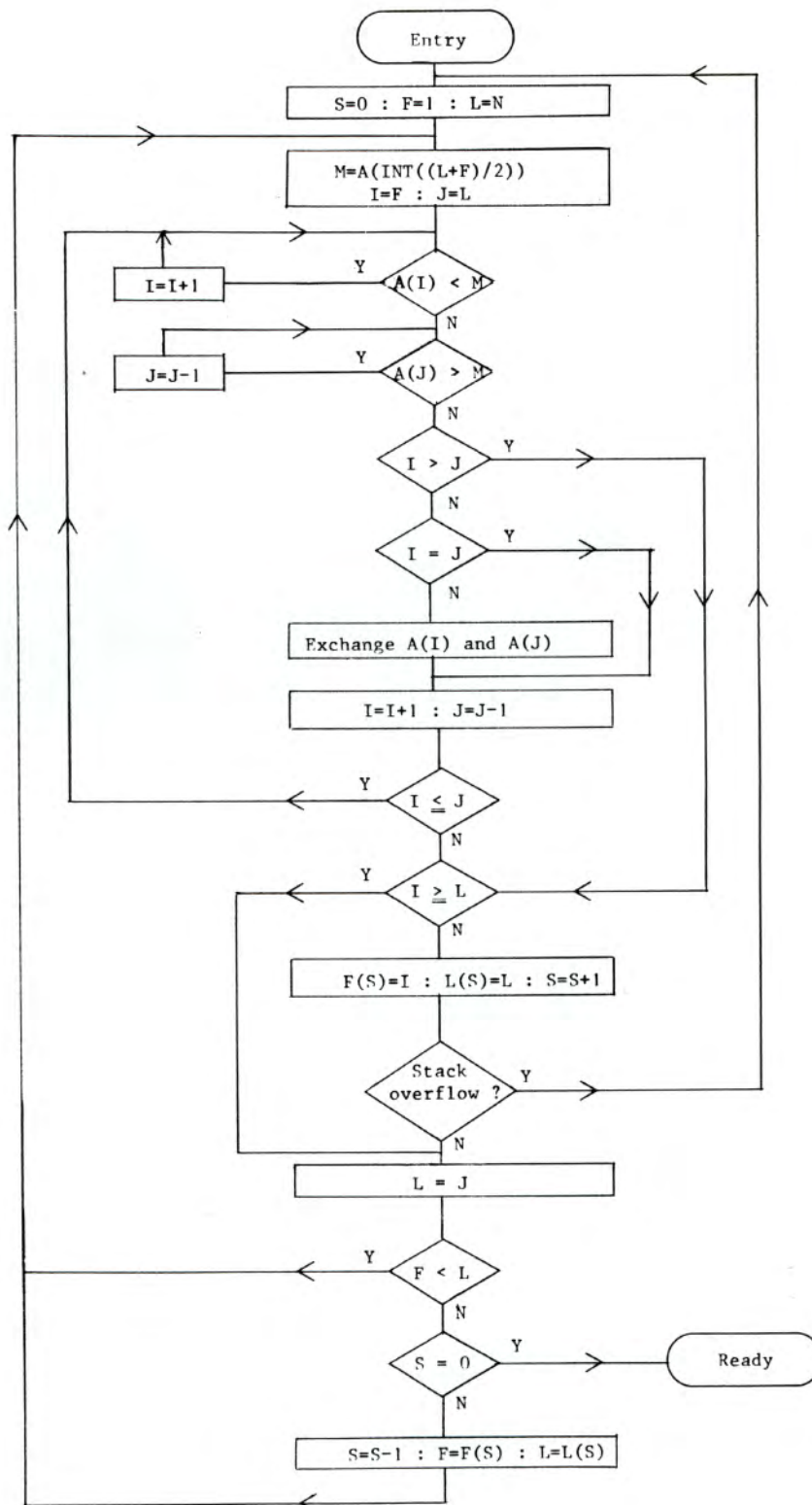
The obvious next step is to get out your old MICRO's, NIBBLES etc. and see if someone, somewhere has published a good machine language sorting routine. I found one in MICRO (July 1979), based on bubblesort, which worked fine. The time needed to sort 1000 reals was approximately 130 secs.

However, in BASIC, CPU time was reduced by a **factor of 40** by using quick sort instead of bubblesort, so I wondered whether the same factor could be gained by applying quick sort, rather than bubblesort, in a machine language program. Table 1 shows that this was indeed the case.

QUICK SORT

Quick sort is a sorting method which, as its name implies, is able to sort large quantities of numbers and strings in a surprisingly short time. **Figure 1** outlines the method in the form of a flow diagram. In essence, quick sort is a recursive algorithm which works as follows: In the first iteration, an element is chosen from the array to be sorted. This can be any element, but the middle element usually gives the best results. Next, two pointers, say **F** and **L**, are initialized. **F points to the first element of the array and L to the last.** Denoting the middle element by **M**, the element pointed to by **F** is compared with **M**. If this element is smaller than **M**, **F** is increased and the second element (i.e. the element now pointed to by **F**) is compared with **M**. This process continues until **F** points to an element that is larger than or equal to **M**. Having found such an element, we start at the other end of the array with the pointer **L**. If the element pointed to by **L** is larger than **M**, **L** is decreased and the one but last element (i.e. the element now pointed to by **L**) is compared with **M**. This process continues until **L** points to an element that is smaller than or equal to **M**. Next, the elements pointed to by **F** and **L** are exchanged and **F** is increased and **L** is decreased. If **F** is not larger than **L**, we start again by comparing the element pointed to by **F** with **M**, and the process continues.

As soon as F is larger than L, the array can be split up into two parts. The first part (i.e. the first element through the element pointed to by **L**) consists of elements smaller than or equal to **M**, and the second part (i.e. the element pointed to by **F** through the last element) consists of elements larger than or equal to **M**. Hence the array can be sorted just by sorting these two sub-arrays separately.



Quick Sort (Cont.)

Each of the sub-arrays can be sorted by repeatedly applying the method described above (note that this makes the method recursive), but since only one sub-array can be sorted at a time, the starting- and ending-point of the second sub-array have to be stored so that it can be sorted later. In BASIC the natural way to do this is to dimension **two arrays**, say **F** and **L**, and to define a **pointer**, say **S**, which is initialized to zero. The starting and ending points of sub-arrays that still have to be sorted can now be stored by the instructions: **F(S)=starting-point: L(S)=ending-point :S=S+1**. If a sub-array has been sorted, which is the case if the ending-point is smaller than or equal to the starting-point, the next sub-array to be sorted can be found by the instructions: **S=S-1: starting-point=F(S): ending-point=L(S)**. Before these instructions are executed, **S** must be checked, for if **S=S-1** leads to a negative result, the sorting is finished.

To avoid allocating working storage and to increase sorting speed, I have used a slightly different method in the machine language program, which consists of pushing the ending and starting points of the sub-arrays that still have to be sorted onto the stack. To obtain enough room in the stack, the BASIC stack is temporarily saved in the input buffer before the sorting begins. Usually, the 256 byte stack is large enough to contain all the starting and ending points generated by the sort. Some specific series, however, may cause stack overflow. I have dealt with this problem by letting the whole sorting process restart on stack overflow. It appeared that this rather crude solution works reasonably well.

SYNTAX OF THE SORT STATEMENT

The syntax of the sort statement is:

& SQR arrayname

The **&** causes an unconditional jump to the machine language sort routine. In this routine the presence of (the token for) **SQR** is checked first, and next the arrayname is evaluated. **The array must be dimensioned before execution of the & SQR statement, or else you get an OUT OF DATA error.**

In the simplest case, the array has one dimension, which must be larger than zero. The sort is then executed on the elements **1** through **N**, where **N** denotes the size of the dimension. The **0**th element of the array is thus neglected by the sort. Figure 2 shows an example.

```
10 DIM A(10)
20 FOR I = 1 TO 10: A(I) = RND (1): NEXT I
30 & SQR A
40 FOR I = 1 TO 10: PRINT I, A(I): NEXT I
50 END
```

JRUN	
1	.0536633605
2	.111870206
3	.302311457
4	.414951005
5	.431383824
6	.559222743
7	.59080247
8	.723385026
9	.922644368
10	.988774152

Figure 2

TWO DIMENSIONAL ARRAYS

The sort routine also accepts arrays (matrices) with **two dimensions**. In this case the **first dimension must be larger than zero and the second dimension (i.e. number of columns-1) must equal one**. If these conditions are not met, the error message **BAD SUBSCRIPT** will be generated.

When a two-dimensional array is mentioned in the **& SQR** statement, the sort is executed on the elements of the second column of the array (except the **0**th element). Thus, if an array **A**, dimensioned with the statement **DIM A(10,1)**, is sorted, the sort is executed on the elements **A(1,1)** through **A(10,1)**. All other elements (i.e. **A(0,0)** to **A(10,0)** and **A(0,1)**) are left in place. See Figure 3 for an example.

```
10 DIM A%(10,1)
20 FOR I = 1 TO 10: A%(I,1) = 5 - RND (1): NEXT I
30 & SQR A%
40 FOR I = 1 TO 10: PRINT I, A%(I,1): NEXT I
50 END
```

JRUN	
1	0
2	0
3	0
4	1
5	1
6	1
7	2
8	2
9	2
10	3

Figure 3

Although the sorting capabilities described above will be sufficient for many purposes, I readily discovered that in some cases the user runs into trouble. For instance, when the ranks of a series of numbers have to be determined, one has to find out (after sorting the numbers) where each number was originally stored in the array, in order to give it its rank.

Although this can easily be done in BASIC, much of the time gain of the machine language sort would be lost.

As another example, suppose that one is working with multiple string arrays, each containing a part of a record. Let's say that the array **AS** contains names and the array **BS** the corresponding telephone numbers. Thus if **AS(I,1) = "MARJO"**, then **BS(I,1)** equals the telephone number of MARJO.

Clearly, if **AS** is sorted, the link with **BS** will be destroyed and we end up with a worthless file.

THE INDEX OPTION

To meet these problems, I have added an **extra option** to the sorting routine. This option can be used with two-dimensional arrays only and is invoked by adding the text **",I"** behind the array name:

&SQR arrayname, I

The **I** stands for Index and the option has the effect that **every time two elements in the second column are exchanged, the two corresponding elements in the first column will be exchanged too**. Hence, by initializing the first column of the array with the numbers **1** to **N** (or in the case of a string array with the strings **STR\$(1)** to **STR\$(N)**), the original place of each element **A(I,1)** can — after the sorting — be found in **A(I,0)**. It must be remembered that the user him/herself is responsible for the initialization of the indi-

ces. This was purposely not done in the program since it sometimes can be useful to use, instead of the series **1** to **N**, another series.

Figure 4 shows an example of a sort with the index option.

```
10 DIM AS(5,1)
20 DATA WIM,JAAP,GERARD,KEES,MICHEL
30 FOR I = 1 TO 5: READ AS(I,1): AS(I,0) = STR$(I): NEXT I
40 & SQR AS,I
50 FOR I = 1 TO 5: PRINT I: TAB( 10): AS(I,0), AS*(I,1): NEXT I
60 END
```

JRUN			
1	3	GERARD	
2	2	JAAP	
3	4	KEES	
4	5	MICHEL	
5	1	WIM	

Figure 4

THE PERFORMANCE OF QUICK SORT

The time needed to sort an array of reals containing random numbers is for a large **N** approximately proportional to **N*LOG(N)**, where the logarithm has base 2. This formula can easily be verified by observing that in the first iteration, **N** elements have to be compared with the middle element.

Denoting the total number of comparisons needed to sort the array by **F(N)** and assuming that the two sub-arrays left after the first iteration are each of length **N/2**, we have,

$$F(N) = N + 2 * F(N/2)$$

Substituting **N*LOG(N)** for **F(N)**, we see that this formula satisfies the equation, i.e.

$$N * LOG(N) = N + 2 * (N/2) * LOG(N/2) = N + N * LOG(N) - N$$

Of course the formula gives only an approximation to the sorting time (because in general the sub-arrays will not be of equal length and because the sorting time will depend on the number of exchanges as well, but the approximation is usually surprisingly accurate).

It is interesting to note that the time needed to sort an array using **bubblesort** varies approximately proportionally with **N*N/2**. This means that on the average, **quick sort sorts an array N/(2*LOG(N))*100% faster than bubblesort**. For **N=1000** this means about a factor of **50** faster and for **N=4000** about a factor of **170**.

Table 1 shows some specific sorting results for reals. The second column of the table gives the time (in seconds) needed to sort 1000, ..., 8000 reals. The least squares line **sorting time = 3.6 E-04 * N * LOG(N)** fits these data very well.

The third and the fourth column give the results if the input of the sort routine is an array that is already sorted, ascending and descending, respectively. The fifth column gives an indication about the price that has to be paid if the index option is used. As can be seen, the sorting time for randoms increases by approximately 30%.

The sixth column requires some explanation. Due to the method of sorting used, there exists some series that result in a bad performance. An example of such a series is given below.

02 4 6 8 10 12 14 16 18 20 13 5 15 4 17 3 19 2 21 1

continued on next page

Quick Sort (Cont.)

TABLE 1: SORTING TIMES (INSECS) FOR REALS

By sorting this series by hand, using the flow diagram, it can be seen that (in the beginning) each row of length K splits up in a row of length 2 and a row of length K-2. This leads to a very low exchange/comparison ratio and it also causes stack overflow for moderate values of N. However, the sorting times are still reasonable and do not seem to justify corrective programming action.

The last column shows the performance of bubblesort. This column needs no further explanation.

dim of array	quick sort				bubblesort	
	randoms	sorted ascend.	sorted descend.	randoms indices	special series	randoms
1000	3	2	2	3	9	129
2000	6	4	4	8	20	513
4000	12	8	8	16	38	2044
6000	18	12	13	-	57	-
8000	26	16	18	-	-	-

(-) means out of memory or out of patience

THE MACHINE LANGUAGE PROGRAM

The machine language program is listed in Figure 5. It has a length of 574 bytes (\$23E Hex) and starts at \$81C0. After assembling the text file and storing it to disk, the program can be installed by : BRUN programname.

This command executes an initialization routine which sets HIMEM TO \$81C0 and installs the & vector. The program makes use of some Applesoft routines in ROM; thus, if the RAM version of Applesoft is used, the relevant subroutine calls must be adjusted.

HOW TO ENTER QUICK SORT

The QUICK SORT was assembled using an Assembler program. You do not need an Assembler to load it into your Apple, however.

The program can be directly entered into memory using the Hexadecimal Codes which immediately follow the memory locations. To do this, enter the monitor by typing CALL -151. The asterisk should appear on your screen.

Now you can begin entering the program at memory location \$81C0 by typing pairs of hexadecimal characters:

81C0:A9 C0 85 73 A9 81 85 74 A9 D3 etc.

Consult page 44 of your Apple Reference Manual for more information on entering Hex codes.

When you have finished entering the program, reenter Basic by typing: CTRL C. Then type:

BSAVE QUICK SORT, A\$81C0, L\$23E

That's all there is to it.



--- END PASS # 1 ---

Figure 5

QUICK SORT

(In Machine Language)

by C. Bongers

Copyright (c) 1982

by Micro-SPARC Inc

Lincoln, MA 01773

ORG \$81C0

```

IC EQU $49      I (CHAR)
RFLAG EQU $06   REALS/INT/STR FLAG
PAR EQU $07     MASK FOR INTEGERS
HFL EQU $08     ELEMENT TUNER
FLAG EQU $64    FLAG/COUNTER
L EQU $17       POINTER TO N TH ELEMENT
F EQU $19       POINTER TO 1 TH ELEMENT
J EQU $1B       TEMP FOR L
I EQU $1D       TEMP FOR F
SPT EQU $1F     STACK POINTER STORAGE
HIM EQU $73     HIMEM
SARA EQU $9B    GENERAL
IST EQU $9D     GENERAL
AGEB EQU $A0    GENERAL
IFL EQU $A5     INDEX FLAG
IN1 EQU $A6     GENERAL
IN2 EQU $A8     GENERAL
MULT EQU $AD    COLUMN LENGTH
SAVE EQU $FA    BACK UP FOR L AND F
STACK EQU $100  STACK
IBUF EQU $200   INPUT BUFFER
BJP EQU $3F5    & VECTOR
TMFP EQU $EAF9  PULL NO IN MFP
M2BT EQU $E2B6  2 BYTE MULTIPLY
CMP EQU $EBB2   COMPARE MFP WITH MEMORY
GNAME EQU $F7D9 EVALUATE ARRAY NAME
CKOM EQU $DEBE  CHECK ON COMMA
CHNW EQU $B1    GET NEXT CHARACTER
CHOLD EQU $B7   GET OLD CHARACTER
BDSB EQU $E196  BAD SUBSCRIPT
SYN EQU $DEC9   SYNTAX ERROR

```

* INITIALIZATION

```

*
81C0-A9C0 BEGIN LDA #BEGIN
81C2-8573 STA HIM SET HIMEM
81C4-A9B1 LDA #BEGIN/256
81C6-8574 STA HIM+1
81C8-A9D3 LDA #START
81CA-8DF603 STA BJP+1 SET & VECTOR
81CD-A981 LDA #START/256
81CF-8DF703 STA BJP+2
81D2-60 RTS

```

* MAIN PROGRAM

```

*
81D3-C9DA START CMP #218 TOKEN FOR SQR ?
81D5-F003 BEQ NSYN BRANCH IF SO
81D7-4CC9DE JMP SYN SYNTAX ERROR (JMP STORE)
81DA-20B100 NSYN JSR CHNW ADVANCE TEXTPOINTER

```

```

81DD-20D9F7 JSR GNAME EVALUATE ARRAYNAME
81E0-A000 LDY #900
81E2-84A5 STY IFL CLEAR INDEX FLAG
81E4-20B700 JSR CHOLD GET OLD CHARACTER
81E7-F00C BEQ NIN NO INDICES IF Z.F. SET
81E9-20BEDE JSR CKOM CHECK ON COMMA
81EC-C949 CMP #IC COMPARE WITH 'I'
81EE-D042 BNE TYP ERROR IF NOT PRESENT
81F0-C6A5 DEC IFL SET INDEX FLAG
81F2-20B100 JSR CHNW ADVANCE TEXT POINTER
81F5-A202 NIN LDX #902
81F7-98 TYA ACCU=0
81F8-2412 BIT #12
81FA-300C BMI STFL BRANCH IF INTEGERS
81FC-E8 INX
81FD-A002 LDY #902
81FF-A511 LDA #11
8201-4980 EOR #80
8203-1003 BPL STFL BRANCH IF STRINGS
8205-A205 LTX #905
8207-C8 INY
8208-8506 STFL STA RFLAG INIT REALS/INT/STR FLAG
820A-8664 STX FLAG
820C-8408 STY HFL
820E-A002 LDY #902
8210-18 CLC
8211-B19B LDA (SARA),Y
8213-659B ADC SARA CALCULATE POINTER TO
8215-0B PHP LAST ELEMENT OF ARRAY
8216-38 SEC
8217-E564 SBC FLAG
8219-8517 STA L
821B-C8 INY
821C-B19B LDA (SARA),Y
821E-E900 SBC #900
8220-28 PLP
8221-659C ADC SARA+1
8223-8518 STA L+1
8225-C8 INY
8226-B19B LDA (SARA),Y NO OF DIMENSIONS
8228-C902 CMP #902 MUST BE <3
822A-F009 BEQ EAL BRANCH IF 2 DIMENSIONS
822C-A5A5 LDA IFL NO INDICES PERMITTED
822E-3002 BMI TYP IF 1 DIMENSION
8230-900F BCC ONLY BRANCH IF 1 DIMENSION
8232-4C96E1 JMP BDSB ELSE BAD SUBSCRIPT ERROR
8235-C8 INY
8236-B19B LDA (SARA),Y
8238-D0F8 BNE TYP SECOND DIMENSION
823A-C8 INY MUST BE 1 WHEN PRESENT
823B-B19B LDA (SARA),Y
823D-C902 CMP #902
823F-D0F1 BNE TYP
8241-C8 ONLY INY
8242-C8 INY
8243-38 SEC
8244-B19B LDA (SARA),Y
8246-E902 SBC #902 PREPARE MULTIPLICATION
8248-85AD STA MULT FIRST DIMENSION-2
824A-88 DEY WITH FLAG

```


Quick Sort (Cont.)

```

824B-B19B LDA (SARA),Y
824D-E900 SBC #900
824F-B5AE STA MULT+1
8251-90DF BCC TYP FIRST DIM MUST BE >0
8253-BA TSX
8254-861F STX SPT SAVE STACK POINTER
8256-BD0001 D1 LDA STACK,X
8259-9D0002 STA IBUF,X SAVE STACK IN INPUT BUFFER
825C-EB INX
825D-D0F7 BNE D1
825F-BA TXA
8260-9A TXS INIT STACK POINTER
8261-20B6E2 JSR M2BT EXECUTE MULTIPLICATION
8264-86AD STX MULT STORE RESULT IN MULT
8266-84AE STY MULT+1
8268-38 SEC
8269-A2FE LDX #9FE
826B-B519 D1LL LDA L+2,X CALCULATE POINTER TO
826D-95FC STA SAVE+2,X START OF COLUMN AND
826F-F5AF SBC MULT+2,X STORE IN F
8271-9518 STA F+2,X
8273-95FE STA SAVE+4,X BACK UP L AND F
8275-EB INX IN SAVE
8276-D0F3 BNE D1LL
8278-A564 LDA FLAG
827A-0A ASL A ACCU=2*FLAG
827B-65AD ADC MULT
827D-B5AD STA MULT MULT=COLUMN LENGTH
827F-9002 BCC DECF
8281-E6AE INC MULT+1
8283-CA DECF DEX
8284-A517 LDA L
8286-6A ROR A
8287-B001 BCS MFL1
8289-CA DEX
828A-8607 MFL1 STX PAR SET PAR FOR INTEGERS
828C-C664 DEC FLAG ADJUST FLAG
828E-1047 BPL MFL ALWAYS
8290-38 ADD5 SEC
8291-A564 LDA FLAG I=I+2/3/5
8293-651D ADC I
8295-851D STA I
8297-9002 BCC RTS1
8299-E61E INC I+1
829B-60 RTS1 RTS
829C-18 SBC5 CLC
829D-A51B LDA J J=J-2/3/5
829F-E564 SBC FLAG
82A1-8518 STA J
82A3-B002 BCS YRDY
82A5-C61C DEC J+1
82A7-60 YRDY RTS
82A8-A51C CMJ CMJ
82AA-C51E CMP I+1
82AC-9006 BCC RTS2
82AE-D004 BNE RTS2
82B0-A518 LDA J
82B2-C51D CMP I
82B4-60 RTS2 RTS
82B5-A519 CMJ1 LDA F
82B7-C517 CMP L
82B9-A51A LDA F+1
82BB-E518 SBC L+1
82BD-9018 BCC MFL
82BF-BA TSX
82C0-D00D BNE EMOS
82C2-A61F LDX SPT
82C4-9A TXS
82C5-BD0002 D2 LDA IBUF,X
82C8-9D0001 STA STACK,X
82CB-E8 INX
82CC-D0F7 BNE D2
82CE-60 RTS
82CF-A203 EMOS LDX #903
82D1-68 DAL PLA
82D2-9517 STA L,X
82D4-CA DEX
82D5-10FA BPL BAL
82D7-A203 MFL LDX #903
82D9-B517 DAL LDA L,X
82DB-9518 STA J,X
82DD-CA DEX
82DE-10F9 BPL DAAL
82E0-18 CLC
82E1-A517 LDA L
82E3-6519 ADC F
82E5-AA TAX
82E6-A518 LDA L+1
82E8-651A ADC F+1
82EA-6A ROR A
82EB-A8 TAY
82EC-8A TXA
82ED-6A ROR A
82EE-9006 BCC NPR
82F0-18 CLC

```

```

82F1-6508 ADC HFL ADJUST IF RESULT NOT EVEN
82F3-9001 BCC NPR
82F5-C8 INY
82F6-2406 NPR BIT RFLAG REALS ?
82F8-3016 BMI PNFP BRANCH IF SO
82FA-7004 BVS SSAR BRANCH IF STRINGS
82FC-0901 ORA #901 ADJUST ACCU IN CASE
82FE-2507 AND PAR OF INTEGERS
8300-859B SSAR STA SARA
8302-849C STY SARA+1
8304-A464 LDY FLAG
8306-B19B DNI LDA (SARA),Y STORE INTEGER OR STRING
8308-999D00 STA IST,Y INFO IN IST
830B-88 DEY
830C-10F8 BPL DNI
830E-3003 BMI DNE ALWAYS
8310-20F9EA PNFP JSR TMFP PULL NO IN MFP
8313-A51D DNE LDA I
8315-A41E GNN LDY I+1
8317-20AB83 JSR CMP2 COMPARE NO WITH A(I)
831A-F00A BEQ GAJ
831C-3008 BMI GAJ BRANCH IF NO<=A(I)
831E-209082 JSR ADD5 I=I+2/3/5
8321-D0F2 BNE GNN ALWAYS
8323-209C82 GNN JSR SBC5 J=J-2/3/5
8326-A51B GAJ LDA J
8328-A41C LDY J+1
832A-20AB83 JSR CMP2 COMPARE NO WITH A(J)
832D-30F4 BMI GNN BRANCH IF NO<A(J)
832F-20AB82 JSR CMJ COMPARE I WITH J
8332-9044 BCC POS BRANCH IF J<I
8334-F037 BEQ NCH DON'T EXCHANGE IF I=J
8336-A464 LDY FLAG
8338-B11D EXIJ LDA (I),Y EXCHANGE A(I) AND A(J)
833A-AA TAX
833B-B11B LDA (J),Y
833D-911D STA (I),Y
833F-8A TXA
8340-911B STA (J),Y
8342-88 DEY
8343-10F3 BPL EXIJ
8345-A5A5 LDA IFL INDEX FLAG SET ?
8347-F024 BEQ NCH BRANCH IF NOT
8349-88 DEY
834A-A200 LDX #900
834C-38 SEC
834D-B518 NYSB LDA J,X
834F-E5AD SBC MULT IN1=J-COLUMN LENGTH
8351-95A6 STA IN1,X
8353-851C LDA J+1,X
8355-E5AE SBC MULT+1 IN2=I-COLUMN LENGTH
8357-95A7 STA IN1+1,X
8359-A202 LDX #902
835B-C8 INY
835C-D0EF BNE NYSB
835E-A464 LDY FLAG
8360-B1A6 EXIN LDA (IN1),Y EXCHANGE INDICES
8362-AA TAX
8363-B1A8 LDA (IN2),Y
8365-91A6 STA (IN1),Y
8367-8A TXA
8368-91A8 STA (IN2),Y
836A-88 DEY
836B-10F3 BPL EXIN
836D-209082 NCH JSR ADD5 I=I+2/3/5
8370-209C82 JSR SBC5 J=J-2/3/5
8373-20AB82 JSR CMJ COMPARE I WITH J
8376-B09B BCS DNE BRANCH IF J>=I
837B-A51D POS LDA I
837A-C517 CMP L I>=L ?
837C-A51E LDA I+1
837E-E518 SBC L+1
8380-B018 BCS DNOT BRANCH IF SO
8382-A517 POS1 LDA L
8384-48 PHA
8385-A518 LDA L+1
8387-48 PHA
8388-A51D LDA I
838A-48 PHA
838B-A51E LDA I+1
838D-48 PHA
838E-BA TSX
838F-D00C BNE DNOT CHECK ON STACK OVERFLOW
8391-A203 LDX #903 BRANCH IF OK
8393-B5FA SNP1 LDA SAVE,X STACK OVERFLOW !
8395-9517 STA L,X RESTORE L AND F
8397-CA DEX
8398-10F9 BPL SNP1
839A-4CD782 JMP MFL AND TRY IT AGAIN
839D-A51B DNOT LDA J
839F-8517 STA L
83A1-A51C LDA J+1
83A3-8518 STA L+1
83A5-4CB582 JMP CMJ1 SORT NEXT SUB-COLUMN
83A8-4CB2EB CRLS JMP CMP COMPARE NO WITH MEMORY

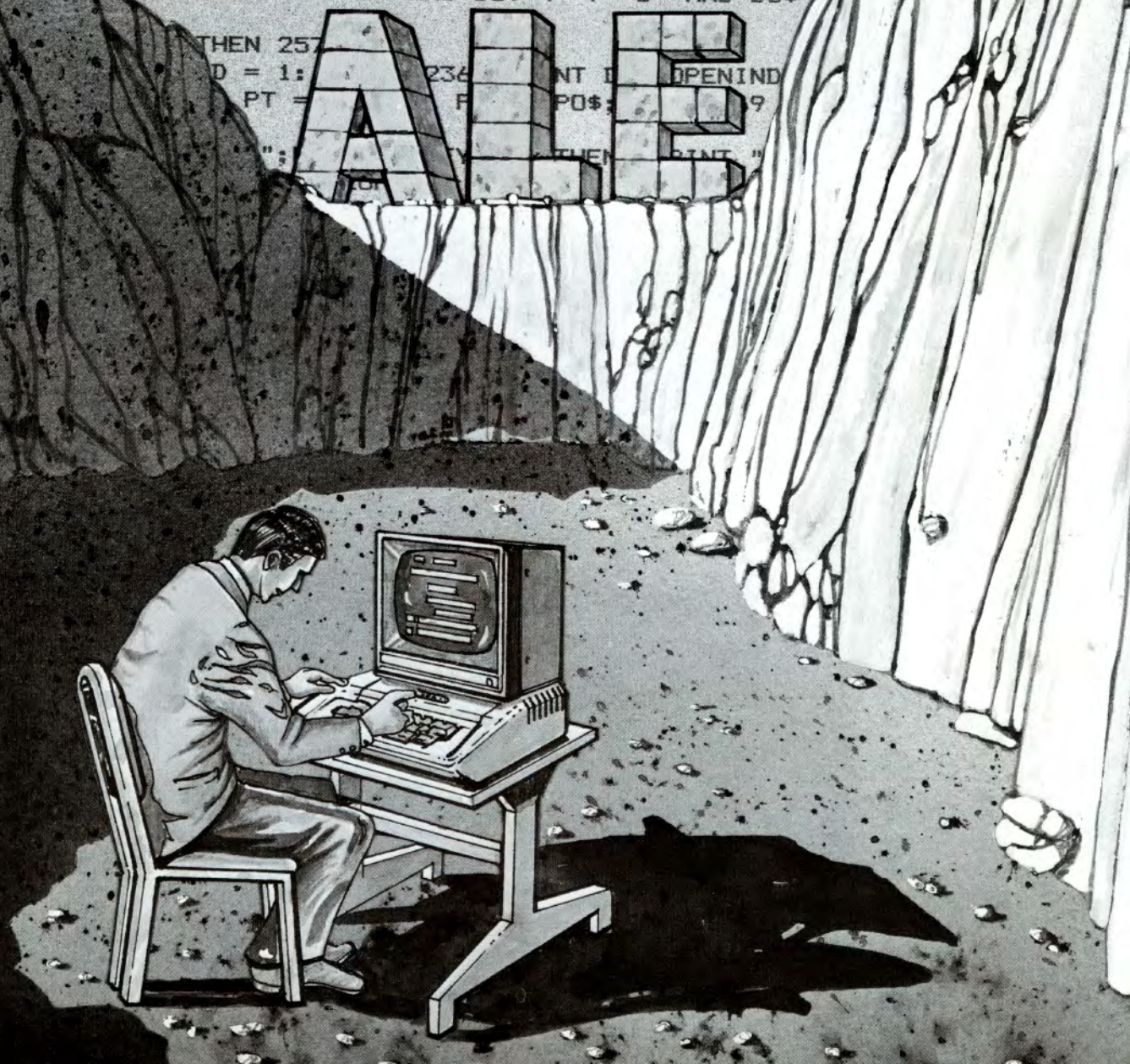
```

continued on page 189


```

51 VTAB 15: HTAB 2: GET K2$: PRINT K2$: IF K2$ = CHR$ (13) THEN VC$ =
   "0" + K1$: GOTO 55
52 IF K1$ = "A" AND (K2$ < "A" OR K2$ > "Z") THEN VTAB 17: PRINT "INVALID
   ENTRY... USE A-Z": GOTO 51
53 IF K1$ = "B" AND (K2$ < "A" OR K2$ > "K") THEN VTAB 17: PRINT "INVALID
   ENTRY... USE A-K": GOTO 51
54 VC$ = K1$ + K2$
55 ONERR GOTO 102
56 VC = LEN (VC$): FOR O = 1 TO 500: NEXT : HOME :ZZ = 0: PRINT D$:AA =
   J:BB = K:CC = 1: PRINT D$;"OPEN INDEX,L";RL:I = AA: IF VC$ < >
   "" THEN 59
57 YY = YY + 1:ZZ = ZZ + 1: GOSUB 3: PRINT YY;")";B$: IF ZZ = 21 THEN
   PRINT D$:: PRINT "PRESS ANY KEY TO CONTINUE ";; GET K$: PRINT K$:
   ZZ = 0
58 GOTO 61
59 IF VC$ = LEFT$ (C$(I),VC) THEN GOSUB 3:YY = YY + 1:ZZ = ZZ + 1:
   PRINT YY;")";B$
60 IF ZZ = 21 THEN PRINT D$: PRINT "PRESS ANY KEY TO CONTINUE ";; GET
   K$: PRINT K$:ZZ = 0: HOME
61 I = I + CC: IF I < 0 OR I > BB THEN 64
62 IF VC$ = "" THEN 57
   IF VC$ < > "" THEN 59
   PRINT "CLOSE INDEX":VC$ = "":VC = 0:I = 0:Y = 0:YY = 0: PR
   KEY FOR MENU";: GET K$: PRINT K$: GOTO 257
   10: PRINT "S)INGLE.. D)OUBLE COL PRINT. RTN=ME
   IF DC$ < > "S" AND DC$ < > "D" AND DC$

```



APPLESOFT LINE EDITOR (A.L.E.)

Applesoft Line Editor

by Joe Brooks
521 Moody
Athens, TX 75751

The Applesoft Line Editor (A.L.E.) is a full-featured editing program that adds powerful editing commands and expanded cursor control to the 48K APPLE II. It is written in assembly language, and designed for use with Applesoft BASIC in ROM. With a touch of the Ampersand key, A.L.E. lists the program lines using the full 40 character width of the screen and prints control characters in the inverse mode. Some of its other features are Auto-Line Numbering, Insert & Delete, and Auto Cursor Forward & Backward.

To save A.L.E. to disk or tape, first type the program into memory. This can be done using an assembler (I used the S-C Assembler II, Version 4.0), or you may use direct memory entry. If you are not sure how this is done, refer to pages 43 & 44 of the APPLE II Reference Manual. For direct memory entry, first enter the Monitor by typing **CALL-151**. Then begin entering the machine language code by Typing. **9060:A9 4C 8D F5 03** etc. To save to disk, type **BSAVE A.L.E., A\$9010,L\$3EF**. Type **9010.93FEW** to save to tape.

To use A.L.E. from tape, type **CALL -151**, then **9010.93FER**. After the program is in memory, type **9010G** to initialize it, then **CTRL C** to return to Applesoft. To use from disk, type **BRUN A.L.E.** Normally, any program previously in memory will not be altered.

The program is now ready to use. To edit your first line, type:

&<LINE NUMBER>

If the line number is in memory, the entire line will first be printed on the screen, then the cursor will be positioned following the line number. If it is not in memory, then only the line number is printed, followed by the cursor. A.L.E. is now in NORMAL mode, awaiting input.

In the NORMAL mode, all keys will perform normally, except for the following:

- <RIGHT ARROW> : Operates normally, except that the cursor will advance only one space past the last character in the line.
- <LEFT ARROW> : Operates normally, except the cursor will not backspace beyond the first character of the line number.
- <CTRL-CHARACTERS> : Printed in inverse text.
- <ESC> : Sets program to EDIT mode.

In the EDIT mode, the following options are available:

- I : Insert a character
- D : Delete a character

- F : Auto Cursor Forward
- B : Auto Cursor Backward
- <ANY KEY> : Stop Auto Cursor
- <LEFT ARROW> : Cursor up
- <RIGHT ARROW> : Cursor down
- A : Accept as is
- <RETURN> : Accept up to cursor
- N : Accept as is, then go to the next line number
- R : Accept up to cursor, then go to the next line number
- X : Exit to Applesoft
- M : Left Bracket ([)
- / : Backslash(\)
- : Underline (_)
- <SPACE BAR> : Enter NORMAL mode

INSERT & DELETE

The Insert command (I key) will shift all characters one space to the right, beginning at the cursor position. A space is left at the cursor.

The Delete command (D key) deletes the character under the cursor, then shifts the remaining characters, from the cursor to the end of the line, one space to the left.

These commands do not exit the EDIT mode. This allows, for example, pressing the I key three times to make room for three characters. Pressing the SPACE BAR will enter the NORMAL mode.

AUTO CURSOR

This function will move the cursor automatically forward (F key) or backward (B key). It will continue moving until any other key is pressed, or until it reaches the end or beginning of the line. By pressing the SPACE BAR, movement will stop, and A.L.E. will enter NORMAL mode. The cursor speed can be changed by typing **POKE 30,<SPEED>** from Applesoft, where <SPEED> can be any integer ranging from 0 to 255. 1 is fast, and 255 is slow. It is initially set to 150.

CURSOR UP & DOWN

In the EDIT mode, the LEFT ARROW performs as an up arrow, and the RIGHT ARROW becomes the down arrow. The cursor will not go below the bottom line or above the top line. A.L.E. remains in EDIT mode.

ACCEPT & EXIT COMMANDS

By pressing the A key, A.L.E. will accept the entire line, regardless of cursor position, and then exit to Applesoft. Pressing the RETURN key performs normally, and accepts the line up to the cursor. Then it exits to Applesoft.

The N key performs as the A key, except, instead of exiting to Applesoft, the Auto-Line Number function is activated. Also, the R key will accept up to the cursor, and then perform the Auto-Line Number function.

The Auto-Line Number function increments the line number to the next actual line in memory, then, as when A.L.E. was first activated, the new line is printed on the screen, with the cursor positioned after the line number. If this was the last line in memory, A.L.E. will exit to Applesoft.

Sometimes, perhaps when typing in a new program, you may want the Auto-Line Number function to have a pre-defined increment. No problem!! The line number increment may be set by typing **POKE 31,<INCREMENT>** from Applesoft, where <INCREMENT> may be any integer ranging from 0 to 255. If the increment is set to 0, (Default value), the Auto-Line Number function will perform as described above, locating the next actual line in memory.

To exit from A.L.E. to APPLESOFT without changing the line in memory, type the X key (this is the same as CTRL-X from APPLESOFT).

The ACCEPT command removes all unnecessary spaces from the edited or newly typed line, before utilizing the Applesoft ROM routines to enter the line into memory. This will allow typing in and editing long program lines without worrying about the spaces. (Applesoft allows 239 characters maximum.)

SPECIAL CHARACTERS

As most know, Applesoft will print the RIGHT BRACKET symbol (]) by typing SHIFT-M. A.L.E. does this in NORMAL mode. Additional special characters may be obtained from the EDIT mode by typing the M key for the left bracket ([), the DIVIDE key for the backslash(/), and the MINUS key for the underline character (_). These characters set the program to NORMAL mode.

HOW IT WORKS

A.L.E. resides in memory starting at \$9010. The actual program is 1006 bytes in length. In addition, it uses 512 bytes for buffer space, which is located above the program, starting at \$9400.

The program consists of five sections. The first (INITIALIZE) initializes the system. The second (LINE NUMBER INPUT) analyzes the input for proper syntax. The third section (PRINT LINE & STORE IN BUFFER) looks for the line number which is entered, then it prints the entire line if it is found, or only the line number if not. As the characters are printed on the screen, they also are stored in A.L.E.'s buffer. The fourth section (EDIT BUFFER ROUTINE) modifies the contents of the buffer (and the characters on the screen) as desired. The final section (PUT BUFFER INTO APPLESOFT) transfers the programs' buffer into the keyboard buffer, and then lets Applesoft ROM store it into memory. If Auto Line Number is in effect, this section also "types" the new line number into the keyboard buffer, and then jumps to the LINE NUMBER INPUT section. Each of these sections will now be described in more detail. Reference should be made to the assembly listing.

INITIALIZE

When A.L.E. is loaded into memory, it must be initialized before it may be used. This is done by BRUN from disk, or 9010G from tape.

First, the Ampersand (&) vector (this is the address location (\$3F5) that Applesoft jumps to when it encounters the &) is set to the starting address of the program. Next, HIMEM is moved down below the program to 36880 (\$9010). This protects the program from Apple-

continued on next page

Applesoft Line Editor (A.L.E.) (Cont.)

soft. Also, the **DELAY** (this determines the speed for the Auto Cursor function) and the Auto Line Number increment **INCR** are set. This section of the program is not used again, until the next time A.L.E. is loaded.

LINE NUMBER INPUT

This is the actual start of the program. The characters typed after the **&** symbol are checked to see if they are numeric. If not, a jump is made to **SNTXER (\$DEC9)**, which is the Applesoft ROM routine to print "SYNTAX ERROR". If input is numeric, Applesoft routines **CHRGOT (\$B7)** and **(LINGET (\$DA0C))** are used to store the line number into **LINNUM (\$50,\$51)**. **LINGET** will exit to **SNTXER** if the line number is greater than **63999**.

PRINT LINE & STORE IN BUFFER

After the text pointer **TXTPTR (\$B8)**, the vertical tab counter **VCOUNT**, and the program buffer are zeroed, the **CHRGOT** routine (**\$B1**) fetches the line number (that was typed in) one character at a time. Each time, the program goes to the subroutine **PSII**. This subroutine prints the character (inverse if it is a control character), stores the character into the program buffer, increments the buffer pointer **BUFPTR**, and then increments the vertical tab counter **VCOUNT**, if necessary. After the line number is printed and stored, the subroutine **SPSII** is used to print and store a space. Next, the horizontal position of the cursor is stored so that it may be later returned to its present position. The **APPLESOFT** routine **FNDLIN (\$D61A)** is used to see if the line number (stored in **LINNUM**) is in memory. If it is not, a branch is made to the next section. If the line is in memory, **LOWTR (\$9B)**, is set to its link field (line pointer) address. The next step is to print the line. This is done one character at a time using **LOWTR** and the **PSII** subroutine. The exception is the fact that **Applesoft stores its commands (keywords) as tokens**. These tokens, when encountered, must be converted to characters, and then each character printed and stored. This is done at **TOKEN**.

The keywords are listed on page 121 of the Applesoft manual. They are stored in the token table (ASCII code) in that exact order, with the last character of each keyword marked by having bit 7 (MSB) set.

For example:

Hex address-	D0D0	D0D1	D0D2	D0D3	D0D4	D0D5
ASCII code-	45	4E	C4(44)	46	4F	D2(52)
Character-	E	N	D	F	O	R

By subtracting **127 (\$7F)** from the token number, we can find the location of the keyword. For example, the keyword **FOR: 129 (\$81) — 127 (\$7F) = 2. FOR is the 2nd keyword in the table.**

After finding the keyword, it is printed and stored using the subroutine **PSII** at **PRTTOK**. A space is printed before and after each keyword using the subroutine **SPSII**. Note the Applesoft routine **INCLD (\$D72C)**. This increments the token table pointer **TOKPTR**

and then loads the **A register** (accumulator) with the character pointed to by **TOKPTR**.

EDIT BUFFER ROUTINE

After the line is printed and stored in the buffer, **BUFPTR** and the cursor position are reset to the values stored in the previous section. We are now ready to edit the buffer.

The various routines in this section edit the line printed on the screen, and also, the program buffer, which contains a "duplicate image" of the line in ASCII code. The program buffer does not contain any tokens. The keywords will be tokenized in the next section of A.L.E. It is important that any changes made on the screen also be made at the appropriate address inside the buffer. This is done by incrementing or decrementing **BUFPTR** (which points at the buffer address to be changed) by the distance that the cursor has traveled.

The **EDIT** commands use several program and ROM routines. **KEYIN** is the keyboard input routine used in **NORMAL** mode, and **CMD** is the input routine used in **EDIT** mode. These routines use the Applesoft ROM routine **INCHR (\$D553)**, which is the character input routine. The ASCII character input is then compared to ASCII values representing specific keys, which in turn determines the routine which the program is to execute.

BSPACE (\$FC10), used in **BCKARR**, **AUTBCK**, and the subroutine **FLASH**, is a MONITOR ROM routine to backspace the cursor one position. It is followed by either the subroutine **DECBUF**, which decrements **BUFPTR**, or **PRTCHR**, which moves the cursor back to its original position, thus requiring no change to **BUFPTR**. **PRTCHR** uses **COUT1 (\$FDF0)**, the MONITOR ROM routine to output a character to the screen. It sets bit 7 (of register A) for normal video output. For control characters, bit 7 is not set, and **COUT1** prints inverse video.

Another useful MONITOR routine is **VERTAB (\$FC22)**, which positions the cursor vertically to the value stored in **VPOS (\$25)**.

The **UPARR** (UP ARROW) routine moves the cursor up one vertical position, then decrements **BUFPTR** by 40. A check is first made to see if the cursor is on the top line (**VCOUNT = 0**), and if so, no action is taken and a jump is made to **CMD**. The **DNARR** (DOWN ARROW) routine is similar, moving the cursor down and incrementing **BUFPTR** by 40. A check is first made for the bottom line by incrementing **BUFPTR**, then checking for a "00" where **BUFPTR** points inside the

buffer. If it is a "00", the program aborts by resetting **BUFPTR**, and then jumping to **CMD**.

INSERT & DELETE first store the present cursor position and **BUFPTR** at the subroutine **STOR**. After "shifting" the characters on the screen and inside the buffer, the cursor and **BUFPTR** are reset to the stored values. **VCOUNT** is used by **INSERT** to update the vertical position **VPOS** in the event that the screen was scrolled upward.

The **AUTFWD** (AUTO CURSOR FOR-

WARD) and **AUTBCK** (AUTO CURSOR BACKWARD) use the subroutine **FLASH**, which prints the underline character on the screen, and then reprints the original character. **AUTFWD** prints the character to move the cursor forward. **AUTBCK** uses **BSPACE** (described above).

PUT BUFFER INTO APPLESOFT

The edited buffer will now be stored into memory (the BASIC program). Since the buffer was originally zeroed, it ends with a "00". This is used to mark the end of the buffer (End Of Line). If the **accept-to-cursor options** are ("R" and "RETURN"), a "00" is placed at the present cursor position.

The next step is to **transfer the buffer into the keyboard buffer**. This is done by the subroutine **TRNSFR**. During this transfer, all spaces are removed that are not between quotation marks and after **REM** or **DATA** statements. Also, one space is removed after a **REM** or **DATA** statement to compensate for the one Applesoft puts in. Let me note here, that it is possible if one wishes, to have A.L.E. remove all of the spaces after a **REM** or **DATA** statement by deleting the space(s) immediately following the **REM** or **DATA** statement. For example,

1000 DATA20,30,4 0,50,60

will remove the extra spaces.

The transfer complete, the **X register** is now equal to the number of characters in the keyboard buffer. **GDBUF1 (\$D533)** will set **X = 239 (\$EF)** if it is greater than 239. Next it places the **EOL** marker "00" and prepares the registers and the keyboard buffer for the next routine, **PUTMEM (\$D444)**.

PUTMEM is the Applesoft ROM routine which stores the keyboard buffer into memory. It normally exits to Applesoft, with the prompt character "J" and the flashing cursor. For the Auto Line Number function, the **PUTMEM** routine must be "intercepted" before it exits to Applesoft. This is done by resetting the "Character output Switch" (**CSW**) to the proper address inside A.L.E. (the **CSW** is described in detail on page 83 & 84 of the APPLE II REFERENCE MANUAL). Now when the **PUTMEM** routine goes to print the prompt character, it instead jumps back to A.L.E. at **INTCEP**.

INTCEP resets **CSW** to its original value, and then increments the current line number by the value stored in **INCR (\$1F)**. If **INCR** is 0, then the line number is set to the next actual line in memory (if no more lines exist, the program exits to Applesoft). The Applesoft ROM routines **CHNG (\$EBA0)**, **FOUT (\$ED34)**, **STRLIT (\$E3E7)**, and **FREFAC (\$E600)**, convert the new line number (now in hex) to its decimal equivalent. It is then transferred to the keyboard buffer, and the text pointer (**TXTPTR**) is set to the start of the line number. This is the same condition as when **& <LINE NUMBER>** is typed in. A jump is now made to **START**.

MORE ABOUT THE PROGRAM

When A.L.E. is initialized, the cursor speed is set to **150** and the Auto Line Number incremented to **0**. These values may be changed to your preference, within the range of 0 to 255, by Poking in the new values, and then resaving the program. The cursor speed is at **36904 (\$9028)**, and the line number increment at **36908 (\$902C)**.

When using the AUTO CURSOR function, A.L.E. normally remains in the EDIT mode. To modify the program to **switch to NORMAL** mode when the cursor reaches the **end of the line** (AUTO CURSOR FORWARD), **POKE** location **37418 (\$922A)** with **26 (\$1A)**. For the **beginning of the line** (AUTO CURSOR BACKWARD), **POKE 37457 (\$9251)** with **243 (\$F3)**. Also, the underline symbol that is used during movement may be changed by Poking the ASCII value of the character of your prefer-

The modified AMPER FIND may now be loaded in at any time and used along with A.L.E. A.L.E. must be initialized first, and then AMPER FIND.

I think you'll find that A.L.E. will greatly ease the job of editing and modifying your Applesoft programs.



continued on next page

Applesoft Line Editor (A.L.E.) (Cont.)

```

90C9- 20 9C FC 2650 EDBUF JSR CLREOL CLEAR TO EOL
90CC- A9 94 2670 LDA /BUFFER RESET BUFFER
90CE- 85 1B 2680 STA BUFPTR+1 POINTER AND
90D0- A5 1B 2690 LDA HSTD POSITION THE
90D2- 85 24 2700 STA HPOS CURSOR
90D4- 85 1A 2710 STA BUFPTR
90D6- A5 25 2720 LDA VPOS
90D8- 3B 2730 SEC
90D9- E5 19 2740 SBC VCOUNT
90DB- 85 25 2750 STA VPOS
90DD- 20 22 FC 2760 JSR VERTAB
90E0- A0 00 2770 LDY #00 RESET
90E2- 84 19 2780 STY VCOUNT VTAB COUNTER
90E4- 20 53 D5 2800 KEYIN JSR INCHR KEYBOARD INPUT
90E7- C9 15 2810 CMP #15 <-?
90E9- F0 15 2820 BEQ FWDARR <-?
90EB- C9 0B 2830 CMP #0B <-?
90ED- F0 19 2840 BEQ BCKARR RETURN?
90EF- C9 0D 2850 CMP #0D RETURN?
90F1- F0 0A 2860 BEQ JRET ESC?
90F3- C9 1B 2870 CMP #1B ESC?
90F5- F0 22 2880 BEQ CMD PRINT & STORE CHARACTER
90F7- 20 81 92 2890 STEP12 JSR PSII
90FA- 4C E4 90 2900 JMP KEYIN
90FD- 4C D4 92 2910 JRET
9100- A0 00 2930 FWDARR LDY #00 FORWARD ARROW
9102- B1 1A 2940 LDA (BUFPTR),Y IF EOL THEN BRANCH
9104- F0 DE 2950 BEQ KEYIN PRINT CHARACTER
9106- D0 EF 2960 BNE STEP12
9108- A5 24 2980 BCKARR LDA HPOS BACKWARD ARROW
910A- D0 04 2990 BNE STEP13
910C- A5 19 3000 LDA VCOUNT
910E- F0 D4 3010 BEQ KEYIN START OF LINE?
9110- 20 10 FC 3020 STEP13 JSR BSPACE YES!->BRANCH
9112- 20 A5 92 3030 JSR DECBUF BACKSPACE CURSOR
9114- 4C E4 90 3040 JMP KEYIN
9116- 3050
9118- 20 53 D5 3060 CMD JSR INCHR KEYBOARD INPUT
911C- C9 0B 3070 CMP #0B <-?
911E- F0 62 3080 BEQ UPARR <-?
9120- C9 15 3090 CMP #15 <-?
9122- F0 79 3100 BEQ DNARR "D"?
9124- C9 44 3110 CMP #44 "I"?
9126- F0 32 3120 BEQ JDEL "F"?
9128- C9 49 3130 CMP #49 "B"?
912A- F0 31 3140 BEQ JINS "A"?
912C- C9 46 3150 CMP #46 "R"?
912E- F0 30 3160 BEQ JAUTF "N"?
9130- C9 42 3170 CMP #42 "X"?
9132- F0 2F 3180 BEQ JAUTB "M"?
9134- C9 41 3190 CMP #41 "/"?
9136- F0 2E 3200 BEQ JACC "-"?
9138- C9 52 3210 CMP #52 RETURN?
913A- F0 2D 3220 BEQ JRETAU SPACE?
913C- C9 4E 3230 CMP #4E
913E- F0 2C 3240 BEQ JAUTO
9140- C9 5B 3250 CMP #5B
9142- F0 2B 3260 BEQ JBASIC
9144- C9 4D 3270 CMP #4D
9146- F0 2A 3280 BEQ LBRAK
9148- C9 2F 3290 CMP #2F
914A- F0 2A 3300 BEQ BCKSLH
914C- C9 2D 3310 CMP #2D
914E- F0 2A 3320 BEQ UNDLIN
9150- C9 0A 3330 CMP #0A
9152- F0 A9 3340 BEQ JRET
9154- C9 20 3350 CMP #20
9156- F0 8C 3360 BEQ KEYIN
9158- D0 BF 3370 BNE CMD
915A- 4C C2 91 3380 JDEL JMP DELETE
915D- 4C FB 91 3390 JINS JMP INSERT
9160- 4C 25 92 3400 JAUTF JMP AUTFWD
9163- 4C 4B 92 3410 JAUTB JMP AUTOBCK
9166- 4C D9 92 3420 JACC JMP ACCEPT
9169- 4C E2 92 3430 JRETAU JMP RETAUT
916C- 4C E7 92 3440 JAUTO JMP AUTO
916F- 4C 3C D4 3450 JBASIC JMP BASIC
9170- 3460
9172- A9 5B 3470 LBRAK LDA #5B LEFT BRACKET
9174- D0 06 3480 BNE STEP14
9176- A9 5C 3490 BCKSLH LDA #5C BACK-SLASH
9178- D0 02 3500 BNE STEP14
917A- A9 5F 3510 UNDLIN LDA #5F UNDERLINE
917C- 20 81 92 3520 STEP14 JSR PSII PRINT & STORE CHARACTER
917F- 4C E4 90 3530 JMP KEYIN
9180- 3540
9182- A5 19 3550 UPARR LDA VCOUNT UP ARROW
9184- F0 93 3560 BEQ CMD <-TOP OF LINE
9186- C6 19 3570 DEC VCOUNT MOVE
9188- C6 25 3580 DEC VPOS CURSOR UP
918A- 20 22 FC 3590 JSR VERTAB
918D- A5 1A 3600 ADJ LDA BUFPTR ADJUST
918F- 3B 3610 SEC BUFFER POINTER
9190- E9 2B 3620 SBC #40
9192- 85 1A 3630 STA BUFPTR
9194- A5 1B 3640 LDA BUFPTR+1
9196- E9 00 3650 SBC #00
9198- 85 1B 3660 STA BUFPTR+1
919A- 4C 19 91 3670 JMP CMD
919C- 3680
919D- A5 1A 3690 DNARR LDA BUFPTR DOWN ARROW
919F- 1B 3700 CLC ADJUST
91A0- 69 27 3710 ADC #39 BUFFER POINTER

```

```

91A2- 85 1A 3720 STA BUFPTR
91A4- A5 1B 3730 LDA BUFPTR+1
91A6- 69 00 3740 ADC #00
91AB- 85 1B 3750 STA BUFPTR+1
91AA- A0 00 3760 LDY #00 BOTTOM OF LINE?
91AC- B1 1A 3770 LDA (BUFPTR),Y
91AE- D0 05 3780 BNE STEP15 NO!->BRANCH
91B0- 20 92 92 3790 JSR INCBUF YES! READJUST
91B3- D0 DB 3800 BNE ADJ BUFFER POINTER
91B5- 20 92 92 3810 STEP15 JSR INCBUF
91B8- E6 19 3820 INC VCOUNT MOVE
91BA- E6 25 3830 INC VPOS CURSOR DOWN
91BC- 20 22 FC 3840 JSR VERTAB
91BF- 4C 19 91 3850 JMP CMD
91C2- 20 BB 92 3860
91C5- A5 25 3870 DELETE JSR STOR <-BUFFER & CURSOR
91C7- 4B 3880 LDA VPOS
91C8- A0 01 3890 PHA
91CA- B1 1A 3910 LDY #01 GET CHARACTER
91CC- 0B 3920 LDA (BUFPTR),Y NEXT TO CURSOR
91CD- A0 00 3930 PHA STORE IT
91CF- 91 1A 3940 LDY #00 AT CURSOR
91D1- 2B 3950 STA (BUFPTR),Y
91D2- F0 0B 3960 PLP
91D4- 20 C8 92 3970 BEQ STEP17 IF EOL THEN BRANCH
91D7- 20 92 92 3980 JSR PRCHR PRINT CHARACTER
91DA- D0 EC 3990 BNE STEP16 <-NEXT CHARACTER
91DC- A9 20 4000 STEP17 LDA #20 SPACE
91DE- 20 C8 92 4010 JSR PRCHR PRINT IT
91E1- A9 00 4020 LDA #00 PLACE "00" (EOL)
91E3- AB 4030 TAY AT END OF BUFFER
91E4- 91 1A 4040 STA (BUFPTR),Y
91E6- 6B 4050 PLA RESET CURSOR
91E7- 85 25 4060 STA VPOS POSITION
91E9- 20 22 FC 4070 STEP18 JSR VERTAB
91EC- A5 1B 4080 LDA HSTD
91EE- 85 24 4090 STA HPOS
91F0- A5 06 4100 LDA TEMP RESET BUFFER
91F2- 85 1A 4110 STA BUFPTR POSITION
91F4- A5 07 4120 LDA TEMP1
91F6- 85 1B 4130 STA BUFPTR+1
91F8- 4C 19 91 4140 JCMD JMP CMD
91FB- 20 BB 92 4150
91FE- A5 19 4170 INSERT JSR STOR <-BUFFER & CURSOR
9200- 4B 4180 LDA VCOUNT
9201- A0 00 4190 PHA
9203- B1 1A 4200 LDY #00 GET CHARACTER
9205- 84 19 4210 STY VCOUNT
9207- 4B 4220 PHA STORE IT
9208- 20 7F 92 4230 JSR SPSII PRINT SPACE
920B- B1 1A 4240 STEP19 LDA (BUFPTR),Y GET NEXT CHARACTER
920D- AA 4250 TAX STORE IT
920E- 6B 4260 PLA PRINT PREVIOUSLY
920F- 20 B1 92 4270 JSR PSII STORED CHARACTER
9212- BA 4280 TXA
9213- F0 03 4290 BEQ STEP20 IF EOL THEN BRANCH
9215- 4B 4300 PHA STORE NEW CHARACTER
9216- D0 F3 4310 BNE STEP19 <-NEXT CHARACTER
9218- A5 25 4320 STEP20 LDA VPOS RESET VERTICAL
921A- 3B 4330 SEC CURSOR POSITION
921B- E5 19 4340 SBC VCOUNT
921D- 85 25 4350 STA VPOS
921F- 6B 4360 PLA
9220- 85 19 4370 STA VCOUNT
9222- 4C E7 91 4380 JMP STEP18
9225- A0 00 4390
9227- B1 1A 4400 AUTFWD LDY #00 GET CHARACTER
9229- F0 CD 4410 LDA (BUFPTR),Y
922B- 20 65 92 4420 BEQ JCMD IF EOL THEN BRANCH
922E- 20 BB 92 4430 JSR FLASH FLASH CURSOR
9231- 20 92 92 4440 JSR INCUR
9234- 2C 00 C0 4450 JSR INCBUF
9237- 10 EC 4470 BIT KBD KEY PUSHED?
9239- AD 00 C0 4480 STEP21 BPL AUTFWD NO!->BRANCH
923C- 2C 10 C0 4490 LDA KBD YES!
923F- C9 A0 4500 BIT STRB RESET STROBE
9241- F0 02 4510 CMP #A0 SPACE?
9243- D0 B3 4520 BEQ JKEY YES!->BRANCH
9245- 4C E4 90 4530 JKEY BNE JCMD NO!->BRANCH
9248- A5 1B 4540 JMP KEYIN
924A- C9 94 4550 AUTBCK LDA BUFPTR+1 START OF LINE?
924C- D0 04 4560 CMP /BUFFER
924E- A5 1A 4570 BNE STEP22 NO!->BRANCH
9250- F0 A6 4580 LDA BUFPTR
9252- 20 10 FC 4590 BEQ JCMD YES!->BRANCH
9255- 20 A5 92 4600 STEP22 JSR BSPACE BACKSPACE CURSOR
9258- 20 65 92 4610 JSR DECBUF
925B- 20 10 FC 4620 JSR FLASH FLASH CURSOR
925E- 2C 00 C0 4630 JSR BSPACE BACKSPACE CURSOR
9261- 10 E5 4640 BIT KBD KEY PUSHED?
9263- 30 D4 4650 BPL AUTOBCK NO!->BRANCH
9266- 4660 BMI STEP21 <-YES!
9267- 4670
9268- 4680 SUBROUTINES
9269- 4690
9265- A9 5F 4700 FLASH LDA #5F UNDERLINE
9267- 20 C8 92 4710 JSR PRCHR PRINT IT
926A- A5 1E 4720 LDA DELAY
926C- 20 AB FC 4730 JSR WAIT
926F- 20 10 FC 4740 JSR BSPACE BACKSPACE CURSOR
9272- A0 00 4750 LDY #00
9274- B1 1A 4760 LDA (BUFPTR),Y GET CHARACTER
9276- 20 C8 92 4770 JSR PRCHR PRINT IT
9279- A5 1E 4780 LDA DELAY
927B- 20 AB FC 4790 JSR WAIT
927E- 60 4800 RTS
927F- A9 20 4810
9281- 20 C8 92 4820 SPSII LDA #20 SPACE
9284- A0 00 4830 PSII JSR PRCHR PRINT CHARACTER
9287- 4840 LDY #00

```


9286-	91	1A	4850	STA (BUFPTR),Y	STORE IN BUFFER				
9288-	20	92	4860	JSR INCBUF	INCREMENT BUFFER POINTER				
928B-	A5	24	4870	LDA HPOS	INCREMENT				
928D-	D0	02	4880	BNE RET1	VTAB COUNTER				
928F-	E6	19	4890	INC VCOUNT					
9291-	60		4900	RTS					
			4910						
9292-	E6	1A	4920	INC BUFPTR	INCREMENT				
9294-	D0	0E	4930	BNE RET2	BUFFER POINTER				
9296-	E6	1B	4940	INC BUFPTR+1					
9298-	A5	1B	4950	LDA BUFPTR+1					
929A-	C9	96	4960	CHP /BUFEND	END OF BUFFER?				
929C-	D0	06	4970	BNE RET2	NO!->BRANCH				
929E-	20	3A	4980	JSR BELL	SOUND BELL				
92A1-	4C	3C	4990	JMP BASIC	->APPLESOFT				
92A4-	60		5000	RTS					
			5010						
92A5-	A5	1A	5020	DECBUF	DECREMENT				
92A7-	38		5030	SEC	BUFFER POINTER				
92A8-	E9	01	5040	SBC #01					
92AA-	B5	1A	5050	STA BUFPTR					
92AC-	A5	1B	5060	LDA BUFPTR+1					
92AE-	E9	00	5070	SBC #00					
92B0-	B5	1B	5080	STA BUFPTR+1					
92B2-	A5	24	5090	LDA HPOS	DECREMENT				
92B4-	C9	27	5100	CHP #39	VTAB COUNTER				
92B6-	D0	02	5110	BNE RET3					
92B8-	C6	19	5120	DEC VCOUNT					
92BA-	60		5130	RTS					
			5140						
92BB-	A5	24	5150	STOR	STORE CURSOR				
92BD-	B5	1B	5160	STA HSTO	POSITION				
92BF-	A5	1A	5170	LDA BUFPTR	STORE BUFFER				
92C1-	B5	06	5180	STA TEMP	POINTER				
92C3-	A5	1B	5190	LDA BUFPTR+1					
92C5-	B5	07	5200	STA TEMP1					
92C7-	60		5210	RTS					
			5220						
92C8-	C9	20	5230	PRCHR	CONTROL CHARACTER?				
92CA-	90	02	5240	BCC STEP23	YES!->BRANCH				
92CC-	09	80	5250	ORA #80					
92CE-	20	F0	5260	JSR COUT1	PRINT CHARACTER				
92D1-	29	7F	5270	AND #97F					
92D3-	60		5280	RTS					
			5290						
			5300		PUT BUFFER INTO APPLESOFT				
			5310						
92D4-	A9	00	5320	RETURN	LDA #00	PLACE "00" (EOL)			
92D6-	A8		5330	TAY	AT PRESENT CURSOR				
92D7-	91	1A	5340	STA (BUFPTR),Y	POSITION				
92D9-	20	5B	5350	ACCEPT	JSR TRNSFR				
92DC-	20	33	5360	STEP24	JSR GDBUF1	PLACE EOL, MASK MSB'S			
92DF-	4C	44	5370	JMP PUTMEM	STORE INTO MEMORY				
			5380						
92E2-	A9	00	5390	RETAUT	LDA #00	PLACE "00" (EOL)			
92E4-	A8		5400	TAY	AT PRESENT CURSOR				
92E5-	91	1A	5410	STA (BUFPTR),Y	POSITION				
92E7-	20	5B	5420	AUTO	JSR TRNSFR				
92EA-	A5	36	5430	LDA CSMW	STORE CHARACTER				
92EC-	B5	06	5440	STA TEMP	OUTPUT SWITCH				
92EE-	A5	37	5450	LDA CSMH					
92F0-	B5	07	5460	STA TEMP1					
92F2-	A9	FC	5470	LDA #INTCEP	SET INTERCEPT				
92F4-	B5	36	5480	STA CSMW					
92F6-	A9	92	5490	LDA /INTCEP					
92F8-	B5	37	5500	STA CSMH					
92FA-	D0	E0	5510	BNE STEP24	->STORE INTO MEMORY				
			5520						
92FC-	A5	06	5530	INTCEP	LDA TEMP	INTERCEPTED!			
92FE-	B5	36	5540	STA CSMW	RESET CHARACTER				
9300-	A5	07	5550	LDA TEMP1	OUTPUT SWITCH				
9302-	B5	37	5560	STA CSMH					
9304-	A5	1F	5570	LDA INCR	INCREMENT SET?				
9306-	D0	21	5580	BNE INCNUM	YES!->BRANCH				
9308-	A0	00	5590	LDY #00	FIND NEXT LINE				
930A-	B1	9B	5600	LDA (LOWTR),Y	IN MEMORY				
930C-	18		5610	CLC					
930D-	69	02	5620	ADC #02					
930F-	B5	06	5630	STA TEMP	SET TEMP				
9311-	C8		5640	INY	TO ADDRESS				
9312-	B1	9B	5650	LDA (LOWTR),Y	OF NEXT				
9314-	69	00	5660	ADC #00	LINE NUMBER				
9316-	B5	07	5670	STA TEMP+1					
9318-	B1	06	5680	LDA (TEMP),Y					
931A-	B5	51	5690	STA LINNUM+1	STORE LINE				
931C-	8B		5700	DEY	NUMBER AT				
931D-	B1	06	5710	LDA (TEMP),Y	LINNUM				
931F-	B5	50	5720	STA LINNUM					
9321-	20	1A	5730	JSR FNDLIN	LAST LINE?				
9324-	B0	03	5740	BCS INCNUM	NO!->BRANCH				
9326-	4C	3C	5750	JMP BASIC	->YES!				
9329-	A5	50	5760	LDA LINNUM	INCREMENT				
932B-	18		5770	CLC	LINE NUMBER				
932C-	65	1F	5780	ADC INCR					
932E-	B5	9F	5790	STA DSCTMP+1					
9330-	A5	51	5800	LDA LINNUM+1					
9332-	69	00	5810	ADC #00					
9334-	B5	9E	5820	STA DSCTMP					
9336-	A2	90	5830	LDX #90					
9338-	38		5840	SEC					
9339-	20	A0	5850	JSR CHNG	CHANGE TWO-BYTE				
933C-	20	34	5860	JSR FOUT	HEXIDEcimal				
933F-	20	E7	5870	JSR STRLIT	LINE NUMBER				
9342-	20	00	5880	JSR PREFAC	TO STRING				
9345-	A0	00	5890	LDY #00					
9347-	B1	5E	5900	STEP25	LDA (INDEX),Y	MOVE THE			
9349-	99	01	5910	STA KEYBRD+1,Y	LINE NUMBER				
934C-	F0	03	5920	BEQ STEP26	TO KEYBOARD				
934E-	C8		5930	INY	BUFFER				
934F-	D0	F6	5940	BNE STEP25					
9351-	A0	01	5950	STEP26	LDY #01	SET TEXT POINTER			
9353-	B4	B8	5960	STY TXTPTR					
9355-	C8		5970	INY					
9356-	B4	B9	5980	STY TXTPTR+1					
9358-	4C	30	5990	JMP START	->NEXT LINE NUMBER				
			6000						
935B-	A2	00	6010	TRNSFR	LDX #00	TRANSFER BUFFER			
935D-	A0	00	6020	LDY #00	TO KEYBOARD				
935F-	B4	06	6030	STY TEMP	BUFFER				
9361-	B4	08	6040	STY RDLFG	CLEAR FLAGS				
9363-	B4	09	6050	STY QFLAG					
9365-	A5	25	6060	LDA VPOS	CURSOR				
9367-	38		6070	SEC	POSITIONING				
9368-	E5	19	6080	SBC VCOUNT					
936A-	B5	25	6090	STA VPOS					
936C-	A9	00	6100	LDA #BUFFER	SET BUFFER				
936E-	B5	1A	6110	STA BUFPTR	POINTER TO				
9370-	A9	94	6120	LDA /BUFFER	BUFFER START				
9372-	B5	1B	6130	STA BUFPTR+1	ADDRESS				
9374-	B1	1A	6140	STEP27	LDA (BUFPTR),Y				
9376-	F0	77	6150	BEQ STEP37	IF EOL THEN BRANCH				
9378-	C9	52	6160	CHP #52	"R"?				
937A-	D0	19	6170	BNE STEP29	NO!->BRANCH				
937C-	C8		6180	INY	YES				
937D-	B1	1A	6190	LDA (BUFPTR),Y					
937F-	C9	45	6200	CHP #45	"E"?				
9381-	D0	0E	6210	BNE STEP28	NO!->BRANCH				
9383-	C8		6220	INY	YES				
9384-	B1	1A	6230	LDA (BUFPTR),Y					
9386-	C9	4D	6240	CHP #4D	"M"?				
9388-	D0	07	6250	BNE STEP28	NO!->BRANCH				
938A-	C8		6260	INY	YES				
938B-	B1	1A	6270	LDA (BUFPTR),Y					
938D-	C9	20	6280	CHP #20	SPACE?				
938F-	F0	24	6290	BEQ STFLG	YES!->BRANCH				
9391-	A0	00	6300	STEP28	LDY #00	NO			
9393-	B1	1A	6310	LDA (BUFPTR),Y					
9395-	C9	44	6320	STEP29	CHP #44	"D"?			
9397-	D0	22	6330	BNE STEP31	NO!->BRANCH				
9399-	C8		6340	INY	YES				
939A-	B1	1A	6350	LDA (BUFPTR),Y					
939C-	C9	41	6360	CHP #41	"A"?				
939E-	D0	17	6370	BNE STEP30	NO!->BRANCH				
93A0-	C8		6380	INY	YES				
93A1-	B1	1A	6390	LDA (BUFPTR),Y					
93A3-	C9	54	6400	CHP #54	"T"?				
93A5-	D0	10	6410	BNE STEP30	NO!->BRANCH				</

OTHELLO

Othello

by Doug Hennig
#224-13512 Maham Rd.
Dallas, TX 75240

Tired of all those space games where all that is required of you are quick reflexes and a fast trigger finger? Yearning for some real mental stimulation? Well, your wait is over because OTHELLO is here.

Othello was invented in the nineteenth century in England where it was the sport of rich businessmen in their gentlemen's clubs. However, it is rumored that the game actually originated almost a thousand years ago in China (the source of many of today's intellectual games). Its recent popularity is due to a combination of marketing expertise on the part of a number of large game companies and the sheer fun of the game.

PLAYING OTHELLO

Othello is played on an eight by eight board, much the same as chess or checkers. The pieces used in the game are white on one side and black on the other (in this computer version, green and orange are used for better contrast). The game starts with two pieces of each player's color in the center of the board.

Each player takes turns putting pieces on the board, trying to capture the opponent's pieces. **You capture pieces by flanking runs of your opponent's pieces with pieces of your own. This causes all of his intervening pieces to be flipped; that is, all of his captured pieces become yours.**

RULES OF THE GAME

The rules of the game are simple. You may only place a piece on the board if you can capture at least one opponent piece. If you cannot, you forfeit your turn. The game ends when: A) The board is filled; B) Neither player can place a piece; C) One player has all of his pieces removed from the board.

Othello sounds like a simple game but it can be quite complex, and strategy plays an important role. The game becomes exciting by sudden reversals in the lead — you may have six pieces more than your opponent when suddenly with one move, you are seven pieces behind. Most of the strategy involves controlling the edges and especially the corners, since from an edge a player can control a large number of squares.

This version of Othello was written in Applesoft. You have the option of playing against the computer or a human opponent. The computer plays a rather good game considering how simple the program is. It often makes some very surprising moves (sometimes I'd swear it is a mind-reader). The game is displayed in low resolution color graphics — the board is blue, the player's pieces are green and orange, and the cursor is red.

POSITIONING THE PIECES

You tell the computer where you want to put your pieces by moving a red cursor around the game board using the game paddles. When the cursor is over the position where you want to put your piece, press either paddle button. If the move is illegal (such as a square already occupied or a square which does not allow you to capture pieces), the cursor will reappear and you will have to choose another square. If you cannot move, press the RETURN key to forfeit your turn. The program keeps track of the score at all times so you can tell at a glance how you are doing.

HOW IT WORKS

I will not describe the program's playing strategy because I feel that it would take something away from the game. I will, however, give a brief description of the program. First the computer sets up the screen and asks if you want instructions. If you do, it gives a very brief description of the game and the operation of the program. Then there is a short wait while variables are initialized. **The board is dimensioned as an array** and the player pieces (WH and BL, for white and black) are set to the colors of each opponent. A number of other arrays contain strategy and board look-up data.

The program then asks how many people are playing and requests their names. Entering information into Othello is done by a subroutine which GETs each character individually and concatenates it into a string. The reason for doing this is that I am allergic to INPUT — I always fear that non-computer oriented people are going to hit RETURN or type commas or other equally "destructive" characters and totally ruin my nice screen with ugly error messages. If you are immune to such fears, feel free to replace GOSUB 60000 with INPUT A\$ and delete the subroutine. If only one person is playing, the second name is set to Apple.

Next the board is set up. The middle four squares of the board are set to two pieces of each player's color. The board is then drawn on the low resolution screen, the names are printed at the bottom of the screen in the text window and the action begins. The cursor is flashed on the screen according to the settings of the game paddles (note that whatever was on the screen at the cursor position is not destroyed but alternately displayed with the cursor to produce the flashing effect). If you are wondering why (in line 10000) I divided the paddle readings by 230 rather than 255, it is because one of my paddles does not work very well (the hazards of "Star Cruiser") and only gives a maximum reading of 235 rather than 255. Set this number to 255 if you wish (it will not affect the game in any way, only give you more "play" in the paddles).

If a key is pressed and it is the RETURN key, the next player's turn begins (if it is not RETURN it is ignored). When either paddle button is depressed, the program plots the color of the player's piece at the cursor position. It then checks to see if the move is legal. If the square is already occupied or if no pieces can be captured, the player's piece is removed and the cursor again flashes, waiting for input.

SCORING

Once a valid move is made, the computer checks every possible means of flipping pieces from that position, across rows, up and down columns and along diagonals. It counts the number of pieces flipped, adds that to the player's score plus one (because he just placed a piece) and subtracts it from the opponent's score. It flips the opponent's pieces by plotting over the opponent's color with the current player's color. The program then checks for an end situation — one player's score equals zero or the board is full. If either occurs, the game ends, the scores are given, the winner's name is printed and the option to play again is given.

If the game is not over, the next player's turn commences. If there are two players, this turn proceeds in a similar fashion to the previous. If the Apple is the opponent, it examines its own strategy to find the best possible move (I was not going to tell you, remember?) and then proceeds with the flipping and score adjustment as described above.



```
2 REM *****
3 REM << OTHELLO >>
4 REM BY DOUG HENNIG
5 REM COPYRIGHT (C) 1982
6 REM BY MICRO-SPARC INC
7 REM LINCOLN, MA. 01773
8 REM ALL RIGHTS RESERVED
9 REM *****
40 TEXT : HOME : INVERSE : FOR I = 1 TO 40: FOR J = 1 TO
5 STEP 4: HTAB I: VTAB J: PRINT " ": NEXT : NEXT : NORMAL
: VTAB 3: HTAB 17: PRINT "OTHELLO"
50 POKE 34,6: HOME : PRINT "WANT INSTRUCTIONS (Y OR N)?
": GET A$: PRINT A$: IF A$ = "Y" THEN GOSUB 62000
90 REM VARIABLE INITIALIZATION
100 DIM BD(9,9):WH = 9:BL = 12:MT = 0
110 DIM RC(7),CC(7)
120 FOR I = 0 TO 7: READ RC(I),CC(I): NEXT
130 DATA -1,-1,-1,0,-1,1,0,1
132 DATA 1,1,1,0,1,-1,0,-1
140 DIM DC(9): FOR I = 1 TO 9: READ DC(I): NEXT
150 DATA 6,5,4,7,-1,3,0,1,2
240 CR$ = CHR$(13)
500 DIM VLX(7,7),V(4)
```

```
510 V(1) = 2:V(2) = 3:V(3) = 1:V(4) = 4
520 FOR I = 0 TO 7:V = V(ABS(3.5 - I) + 0.5) / 2
530 FOR J = 0 TO 7:VLX(I,J) = VLX(I,J) + V: NEXT
540 FOR J = 0 TO 7:VLX(J,I) = VLX(J,I) + V: NEXT
550 NEXT I
555 REM SEE WHO WILL PLAY
560 HOME : POKE 49168,0: PRINT "HOW MANY PEOPLE ARE PLAY
ING? ": GET A$: PRINT A$
580 NP = VAL(A$) : (A$ = "1" OR A$ = "2"): IF NP = 0 THEN
NP = 1
600 PRINT : IF NP = 1 THEN 650
620 FOR I = 1 TO 2: PRINT "NAME OF PLAYER "I" - ": GOSUB
60000
630 N$(I) = LEFT$(A$,10): NEXT
640 GOTO 700
650 PRINT "YOUR NAME - ": GOSUB 60000:N$(I) = LEFT$(A
$,10)
660 N$(2) = "APPLE"
670 REM SET UP INITIAL BOARD
700 FOR I = 1 TO 8: FOR J = 1 TO 8:BD(I,J) = 0: NEXT J,I
710 BD(4,4) = WH:BD(4,5) = BL
720 BD(5,4) = BL:BD(5,5) = WH
```



```

730 POKE 34,0
900 GOSUB 50000
910 FOR I = 4 TO 5: FOR J = 4 TO 5: COLOR= BD(I,J):R = I
  $ 4 + 1: C = J $ 4 + 1: PLOT C,R
920 NEXT J,I
950 LR = 3:HR = 6:LC = 3:HC = 6
1000 M = 5
1010 WC = 2:BC = 2
1020 REM MAIN GAME ROUTINE
1040 HOME : VTAB 24: PRINT N$(1) TAB( 11)WC TAB( 20)N$(2
  ) TAB( 31)BC
2000 CH = 0
2010 T = (M AND 1): IF T THEN 2100
2020 VTAB 21: HTAB 20: INVERSE : PRINT N$(2)"S MOVE" CHR$
  (7): NORMAL
2025 PL = BL
2030 ON NP GOSUB 3500,3000
2035 IF CH > 0 THEN BC = BC + 1
2040 GOTO 2200
2100 VTAB 21: INVERSE : PRINT N$(1)"S MOVE" CHR$ (7): NORMAL

2105 PL = WH
2110 GOSUB 3000
2120 IF CH > 0 THEN WC = WC + 1
2200 IF (M AND 1) = 0 THEN CH = - CH
2220 WC = WC + CH:BC = BC - CH
2230 HOME : VTAB 24: PRINT N$(1) TAB( 11)WC TAB( 20)N$(2
  ) TAB( 31)BC
2240 IF CH = 0 THEN SK = SK + 1
2250 IF CH < > 0 THEN SK = 0
2260 M = 5 - M
2300 IF WC + BC = 64 THEN 2500
2310 IF SK = 2 THEN 2500
2320 IF WC = 0 OR BC = 0 THEN 2500
2400 IF R1 < = LR AND LR > 1 THEN LR = R1 - 1: IF LR <
  0 THEN LR = 0
2410 IF R1 > = HR AND HR < 8 THEN HR = R1 + 1
2430 IF C1 < = LC AND LC > 1 THEN LC = C1 - 1: IF LC <
  0 THEN LC = 0
2440 IF C1 > = HC AND HC < 8 THEN HC = C1 + 1
2450 GOTO 2000
2490 REM GAME OVER
2500 HOME : VTAB 21: PRINT "THE GAME IS OVER. FINAL SCOR
  E:"
2505 INVERSE
2510 PRINT N$(1) TAB( 11)WC TAB( 20)N$(2) TAB( 31)BC
2512 NORMAL
2515 IF WC = BC THEN PRINT "IT WAS A TIE GAME.": GOTO 2
  600
2520 W = 1: IF BC > WC THEN W = 2
2530 PRINT N$(W)" WAS THE WINNER."
2600 PRINT "WANT TO PLAY AGAIN? "; GET A$: PRINT A$
2610 IF A$ < > "N" THEN 700
2620 TEXT : HOME : END
2990 REM FLASH CURSOR AND SEE IF BUTTON PRESSED
3000 GOSUB 10000
3010 OC = SCRN( C,R): COLOR= 1
3020 PLOT C,R
3030 FOR I = 1 TO 50: NEXT
3035 COLOR= OC: PLOT C,R: FOR I = 1 TO 50: NEXT
3040 X = PEEK (49249):Y = PEEK (49250)
3050 IF X > 127 OR Y > 127 THEN 3400
3060 A = PEEK (49152):K = PEEK (49168): IF A = 141 THEN
  CH = 0: RETURN
3070 GOTO 3000
3390 REM SEE IF HE CAN GO THERE
3400 R1 = (R - 1) / 4:C1 = (C - 1) / 4: IF BD(R1,C1) < >
  MT THEN 3035
3410 GOSUB 38200: IF CH = 0 THEN BD(R1,C1) = MT: GOTO 30
  35
3420 RETURN
3490 REM COUNT FLIPS
3500 CH = 0:RM = 0:CM = 0:MC = - 1:D = 0
3510 FOR R1 = LR TO HR
3520 FOR C1 = LC TO HC
3530 IF BD(R1,C1) < > MT THEN 3610
3550 GOSUB 38000
3560 IF CH > 0 THEN CH = CH + VLZ(R1 - 1,C1 - 1)
3570 IF CH = MC THEN D = D + 1: IF RND (1) < 1 / D THEN
  RM = R1:CM = C1
3590 IF CH > MC THEN RM = R1:CM = C1:MC = CH
3610 NEXT C1
3620 NEXT R1
3800 IF MC > 0 THEN R1 = RM:C1 = CM:R = R1 $ 4 + 1:C = C
  1 $ 4 + 1: GOSUB 38200
3900 RETURN
4000 END
9990 REM GET PADDLE INPUT
10000 R = INT ( PDL (0) / 230 $ 8) $ 4 + 1:C = INT ( PDL
  (1) / 230 $ 8) $ 4 + 1
10010 IF C = 1 THEN C = 5
10020 IF R = 1 THEN R = 5
10030 RETURN
37990 REM CHECK IF PIECES FLIPPED
38000 CH = 0: FOR D = 0 TO 7
38010 RC = RC(D):CC = CC(D)
38020 GOSUB 39000: IF N > 0 THEN CH = CH + N
38030 NEXT D
38040 RETURN
38190 REM PLOT PIECES FLIPPED
38200 COLOR= PL: PLOT C,R:BD(R1,C1) = PL
38210 CH = 0: FOR D = 0 TO 7
38220 RC = RC(D):CC = CC(D)
38230 GOSUB 39000: GOSUB 39100: IF N > 0 THEN CH = CH +
  N
38240 PLOT C,R: NEXT D
38250 RETURN
38990 REM CHECK PIECES ON BOARD
39000 RO = R1:CO = C1
39010 FOR I = 1 TO 8
39020 RO = RO + RC:CO = CO + CC:BD = BD(RO,CO)
39030 IF BD = PL THEN N = I - 1: RETURN
39040 IF BD = MT THEN 39070
39050 NEXT I

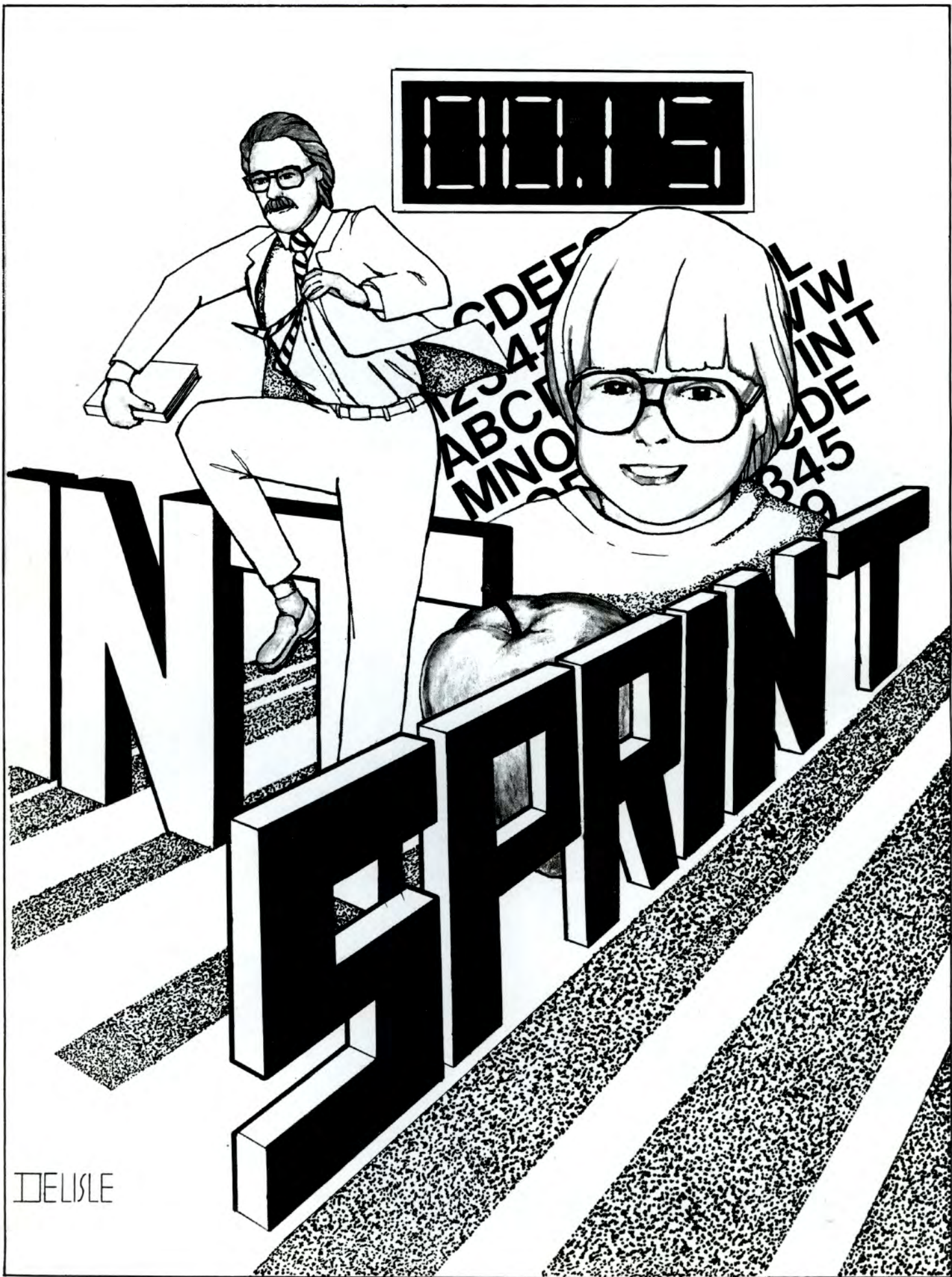
```

```

39070 N = - 1: RETURN
39100 RO = R1:CO = C1: IF N < 1 THEN RETURN
39110 FOR I = 1 TO N
39120 R = R + RC $ 4: C = C + CC $ 4
39130 R1 = (R - 1) / 4:C1 = (C - 1) / 4:BD(R1,C1) = 21 -
  BD(R1,C1)
39135 COLOR= PL: PLOT C,R
39140 NEXT I:R1 = RO:R = R1 $ 4 + 1:C1 = CO:C = C1 $ 4 +
  1: RETURN
49900 REM SET UP BOARD
50000 GR : COLOR= 2: FOR I = 3 TO 37 STEP 4
50010 HLIN 3,35 AT I: VLIN 3,35 AT I: NEXT I: RETURN
59900 REM INPUT SUBROUTINE
60000 A$ = ""
60010 GET B$
60020 IF B$ = CHR$ (13) THEN PRINT : RETURN
60030 IF B$ < > CHR$ (8) THEN A$ = A$ + B$: PRINT B$:
  GOTO 60010
60040 IF LEN (A$) = 0 THEN 60010
60050 PRINT B$ "B$; IF LEN (A$) = 1 THEN A$ = "": GOTO
  60010
60060 A$ = LEFT$ (A$, LEN (A$) - 1): GOTO 60010
62000 REM INSTRUCTIONS
62010 HOME : PRINT " THE OBJECT OF OTHELLO IS TO OWN T
  HE": PRINT "MOST PIECES AT THE END OF THE GAME.": PRINT
  "YOU GAIN PIECES BY FLANKING ROWS OF": PRINT "OPPONE
  NT'S PIECES WITH YOUR OWN AND": PRINT "'FLIPPING' TH
  EM TO BECOME YOUR OWN."
62020 PRINT " USE THE GAME PADDLES TO MOVE THE": PRINT
  "RED SQUARE (CURSOR) AROUND THE SCREEN."
62030 PRINT "WHEN THE CURSOR IS WHERE YOU WANT TO": PRINT
  "MOVE, PRESS EITHER GAME BUTTON. IF THE": PRINT "MOV
  E IS ILLEGAL, THE CURSOR WILL RE-"
62040 PRINT "APPEAR AND YOU WILL HAVE TO MOVE SOME-": PRINT
  "WHERE ELSE. IF YOU HAVE NO MOVE AT ALL,": PRINT "PR
  ESS "; INVERSE : PRINT "RETURN"; NORMAL : PRINT " .
  "
62045 PRINT " THE FIRST PLAYER'S PIECES ARE ORANGE,":
  PRINT "AND THE SECOND PLAYER'S ARE GREEN."
62050 VTAB 23: HTAB 9: INVERSE : PRINT "PRESS RETURN TO
  CONTINUE": NORMAL
62060 K = PEEK (49152): IF K < 128 THEN 62060
62070 K = PEEK (49168): RETURN

```





IELISLE

APPLE SPRINT

Apple SPRINT (Speed Reading Improvement Technique)

by Ken & Michelle McCandless
2747 Inspiration Drive
Colorado Springs, CO 80917

Have you ever wished you could cut the amount of time you spend wading, muddling and plodding through those mountains of paper? Use some of that lost time for other constructive purposes? Well, here is a solution for you: Apple SPRINT.

Have you ever desired something to help your preschool or early grammar school child with reading? Well, here is your wish: Apple SPRINT.

SPRING does as its name implies. **Both your reading speed and your reading comprehension can be beneficially increased.** Within a short period of time you could realize a **100% increase.** Your child can also be aided, even if your child can only read letters and/or numbers.

SPRINT flashes data (letters, numbers, phrases or sentences) on the display screen upon your cue. You read the flashed data and type in what you just read. Sound easy? It is and it's fun.

LOADING THE PROGRAM

Load the program from whatever media you have saved it on (disk or tape). With DOS and APPLESOFT in ROM, at least 32K of memory is required. Without DOS but with APPLESOFT in ROM, only 16K of memory is required.

HOW TO EFFECTIVELY USE SPRINT

Before you use the program find an article of moderate length (about 500 words should be sufficient) and time yourself while you read that article. **Determine your reading speed by dividing the number of words in the article by the elapsed time for reading it.** This provides you with your initial reading speed. Take note of this value. Now periodically repeat this procedure while using SPRINT. This way you can track your improvement.

ON RUNNING SPRINT

When you run SPRINT you are initially asked if you want instructions, which are basically what follow here. Then you are asked to respond to the start-up control parameters.

The first control parameter selects the **format of the data** to be flashed. There are **two formats** available. One is designed for the young child, the other for a participant with normal reading ability. A "Y" response to **PRESCHOOLER? (Y/N)** selects the child's format. Don't mistakenly think that the child's format is easy. It can and does provide a challenge. It will even challenge an adult who might venture into this realm.

The next control parameter selects the **type of data** to be flashed. The response to the **NUMBERS? (Y/N)** question selects the type, with a "Y" selecting numeric data. A negative response selects phrases or letters. The letters are for the preschool child's format. The phrases, which may be complete sentences, are for the normal format. The sentences

generated are, for the most part, nonsensical but grammatically correct.

The next two prompts are for the level of difficulty. The question **LEVEL? (0=EASIEST/4=HARDEST)** selects the amount of data to be flashed. The question **SPEED? (0=SLOWEST/9=FASTEST)** selects how long the data appears on the display screen before it is erased.

Hitting the **ESCAPE** key in response to any of these questions will return you to the previous question (you cannot return to the instructions).

The following **default values** will be used for non-valid input to the control parameters:

INSTRUCTIONS	= N
PRESCHOOLERS	= N
NUMBERS	= N
LEVEL	= 2
SPEED	= 4

SPRINT-ing

Following the control parameter questions you will see a display of the level and speed which you are attempting. Below this you will see the box (or window) in which the data will be flashed. Below this box you will see a **flashing cursor**. This is your ready cue. **When you are ready to read, hit any key and the data will be flashed. Now you will see a question mark; this is your prompt to type in what you just read.** If you get it correct you will proceed on to another. If you get it wrong you will be shown the same data again. You will have two chances to get it correct. If you don't, you will

be shown the data you missed along with your responses. (Note: Spaces are added in the preschool format data. When typing in this data ignore those spaces. Just type in the data.)

The program will automatically adjust the difficulty level and speed based upon your performance.

If you wish to make a change to any of the control parameters, hit the **ESCAPE** key when you see the flashing cursor. This returns you to the data type question. Returning to the format question, **PRESCHOOLER? (Y/N)**, resets everything and starts afresh.

You are scored as follows:

2 points - If you are correct on the first try.
1 point - If you are correct on the second try.
0 points - If you blow it completely.

To terminate the program at any time, hit **Control Q**.

HOW SPRINT WORKS

Figure 1 lists all the **major program sections** and what they do. Figure 2 lists all the **variables** and their usage.

The program begins by poking in the sound routine. This is the routine from the RED BOOK, page 43. This routine expects the value of the tone to be in **768** (decimal) and the duration of that note in **769** (decimal). These values may range up to 255.

The program then builds the arrays, which contain the words for sentence building, from the data statements. Next the control parameters are questioned and accepted.

MAJOR PROGRAM SECTIONS

LINE NUMBERS

100-200

300

350-400

450-500

550-950

1000-1050

1100-2200

1350

1400

2250-3600

2250-2400

2450-2650

2700-2950

3000-3250

3300-3550

3600

3700-4200

4250

4300

4350-4900

4600-4700

4800-4900

4950-8050

5050-5500

5550-8050

8100-8900

8950-9400

9450-9950

10000-10900

10950-11750

11800-12150

12200-13100

13150-13950

14950-17450

Pokes in the sound routine

Descending notes. Used when making decreasing difficulty.

Ascending notes. Used when making it more difficult.

Buzz for wrong answer and for poor overall performance.

Tune for right answer and good overall performance.

Sound for excellent performance at end.

Program set-up and array building routines.

Function used to center data in display window.

Function used to generate numbers.

Control parameter acceptance.

Instructions.

Format determination.

Words or numbers determination.

Level determination.

Speed determination.

Reset of response counters.

Screen display.

Clean-up of extra strings.

Scoring control set-up.

Preschool format data construction.

Preschool numbers build routine.

Preschool letters build routine.

Normal format data construction.

Number build routine.

Phrases or sentences build routine.

Flash the constructed data.

Check input for correctness.

Correct response routine.

Wrong response routine.

Adjust difficulty routine.

Error handling routine.

Ending routine.

Instruction output format routine.

Data statements.

FIGURE 1

continued on next page

Apple SPRINT (Speed Reading Improvement Technique) (Cont.)

Following this, the display screen is built and the program progresses onward to generate the display data. The display data is created using randomization and concatenation from one of four routines. The routine is selected based upon the format control parameter and the data type control parameter. The amount of data to be developed is determined by the **level parameter**. These routines perform as follows:

1. If the preschool format is selected and numbers are not selected then **lines 4800 to 4900** are used. Here random numbers are used to select characters from the verb array and intersperse these characters between spaces. Also, this data is saved in **KIS** without the spaces for use in comparing the input string for correctness.
2. If the preschool format and numbers are both selected, the **lines 4600 to 4700** are used. Here random numbers are generated, converted to integers, and then ASCII, and interspersed between spaces. This data is also saved without spaces in **KIS**.
3. If the normal format is selected and numbers are selected, then **lines 5050 to 5500** are used. Here random numbers are generated, converted to integers and then ASCII.
4. If the normal format is selected and numbers are not selected then **lines 5550 to 8050** are used. Here random numbers are used as indexes into the arrays containing the words for the phrases or sentences. Then, based upon the level parameter, those random indexes are used to build the display data. If this created data is greater than 38 characters long, then another attempt is made, this time building fewer words. This will continue until the data generated is not greater than 38 characters. (This usage of random numbers as indexes to the word arrays produces some very strange sentences. This is intentional and used to increase the comprehensive portion of your reading ability.)

Once this data is built, **the data is ready to be flashed**. The length of time that the data will remain displayed is determined by the **Speed** parameter. The speed parameter is subtracted from the maximum speed of 9 when the speed control is accepted. This result is used to set the number of iterations of the **FOR..NEXT** loop at **line 8850**.

When the data has been flashed, the response is compared with the data displayed. If the response is correct then the program indicates correctness by going to the Sound subroutine.

If this is the **first attempt** at the data, then the program goes to the display screen and data generation point, **line 3700**. If the response is correct and this is the second attempt the program will show the previous response on the line above the correct response, wait a couple of seconds and then go to **line 3700**.

If the response was incorrect and this is the first attempt, the response is saved in **II\$**. The program then goes to the data flashing point to redisplay the same data, using **line 8350**. If

the response was incorrect and this is the second attempt, then the data is displayed along with both responses. The program will wait for a keystroke and then proceed to line 3700 to generate new data.

When five correct responses in a row or three incorrect responses in a row are attained, then the **Auto Adjust subroutines** are used. These routines first attempt to adjust the speed parameter. If the speed cannot be further adjusted, then the level is adjusted and the speed is set about midpoint. These routines use the sound routines to notify you of a change. Ascending notes are sounded when the difficulty is increased and descending notes when difficulty decreased.

An **Error routine** is used to trap invalid inputs and **Control-C**. If you are typing in this program yourself, don't include the **ONERR**

GOTO (line 1100) until you have run the program a couple of times.

The **Ending routine** is branched to when a **Control-Q** (or **Control-C**) is encountered. This routine prints out the score and a rating then stops.

INSTRUCTIONS AND FORMATTING

The instruction output routine was developed out of the basic need to maintain my sanity. After many repeated attempts to format the instructions so that a word would not wrap around on the display, I decided there must be an easier way. So I wrote this routine. This routine is very slow, but since giving instructions is not time critical, it is acceptable for this application.

VARIABLES AND THEIR USAGE

AS	Data to be flashed.
A%()	Holding array for indexes to articles. Used in sentence generation.
AD	Holds the number of adjectives that are in the data statements.
AD\$()	Adjective array.
AD%()	Holding array for indexes to adjectives. Used in sentence generation.
AR	Holds the number of articles that are in the data statements.
AR\$()	Article array.
B	Work variable. Multiple uses.
BB	Work variable. Multiple uses.
CP	Character position offset within the data, Q\$. Used in instructions.
CT	Correct response counter.
ER	Error code.
G\$	Bell (Control-G).
H	Data type switch. H=1 means numeric data.
IS	Input response.
II\$	Hold area for an initially wrong response.
JP	Character position offset from CP, for end of word.
K	Format switch. K=1 means Preschool format.
KA\$	Preschool data for display.
KIS	Preschool data without imbedded spaces. Used in correctness check.
L	Level of difficulty.
LP\$	Line to be output. Used in instruction output.
N%()	Holding array for indexes to nouns. Used in sentence generation.
NO	Holds the number of nouns that are in the data statements.
NO\$()	Noun array.
P%	Holds the index to the preposition. Used in sentence generation.
PC	Counter to locate the bottom of the screen. Used in instructions.
PR	Holds the number of prepositions that are in the data statements.
PR\$()	Preposition array.
Q	Maximum number of data statements to be read in the instruction routine.
Q\$	Data to be formatted for output. Used in instructions.
S	Score.
S2	Scoring control.
SS	Space.
T	Number of data displays generated.
T1	Control for T.
V%	Holds the index to the verb. Used in sentence generation.
VB	Holds the number of verbs that are in the data statements.
VBS()	Verb array.
W	Counter used to give two tries per display.
WC	Wrong response counter.
WP\$	The word extracted from Q\$. Used in instruction output.
X	Speed. Used in the screen display.
XX	Speed. Used for flash duration. (XX=9-X)
Z\$	Work variable. Used for input.

FIGURE 2

This routine reads the data and builds a string for display that will be less than 40 characters in length. This string is displayed and it continues building another string for display. When 23 lines have been displayed the routine goes into a keystroke wait. Upon receiving a keystroke, it clears the screen and starts building the next display string. This continues until there is no more data to be read. The variable "Q" contains the number of data statements to be read for the entire output and is used in a FOR..NEXT loop to control the ending point.

When developing the data, remember that commas and colons are meaningful to APPLE-SOFT. In those sentences which require a comma, colon or the ignored leading spaces, just enclose that data statement in quotation marks.

Two special control indicators are used in this routine to force desired breaks. A single "@" is used to print the currently developed output and advance to the next line. As an

example, see line 16450; the first @ forces the printing of the data. The second @, since there is currently no data in the output line, causes a blank line to be output.

The second control indicator is "@@". This control forces a full display screen condition. It is highly probable that while developing the output a paragraph will be split, such that only part will be displayed at the bottom of the display screen. When the next display screen is being output, the remaining portion of the split paragraph will be displayed. To prevent this, if necessary, place "@@" at the end of the preceding paragraph (see line 16600).

This routine has one minor idiosyncrasy in that the very last character in the very last data statement must be a single @. If not, a portion of the data in that data statement may not be displayed. If "@@" is used there, an extra empty screen will be displayed.

HOW TO CUSTOMIZE/MODIFY SPRINT

The most obvious place to make a modification to the program is in the word lists at

lines 14150 to 16300. I wouldn't recommend any changes to the article list (line 14100) because the randomizing routine at lines 5550 to 5750 is size dependent and besides, the sentences would read poorly without the articles.

The other lists, however, can be changed quite easily. Word for word changes do not require any additional changes within the program. If you wish to delete some entries (or add some) you must modify the appropriate count field and array dimension. For example, suppose you wished to add five verbs to the verb list. The count field which must be modified is VB, on line 1700, which now would read VB=60. The dimension statement on line 1650 must also be changed to read VB\$(60).

Another area you might like to modify would be the sound generation. If you are musically creative, some very dramatic tunes could be included in this program.



```

50 REM *****
55 REM << APPLE SPRINT >>
60 REM BY
65 REM KEN & MICHELLE MCCANDLESS
70 REM COPYRIGHT (C) 1982
75 REM BY MICRO-SPARC INC
80 REM LINCOLN, MA. 01773
85 REM ALL RIGHTS RESERVED
87 REM *****
100 REM INITIALIZE SOUND ROUTINE
150 POKE 770,173: POKE 771,48: POKE 772,192: POKE 773,136
    : POKE 774,208: POKE 775,5: POKE 776,206: POKE 777,1:
    : POKE 778,3: POKE 779,240: POKE 780,9: POKE 781,202
200 POKE 782,208: POKE 783,245: POKE 784,174: POKE 785,0:
    : POKE 786,3: POKE 787,76: POKE 788,2: POKE 789,3: POKE
    : 790,96: POKE 791,0: POKE 792,0
250 GOTO 1100
300 FOR B = 1 TO 6: POKE 768,76 + 4 * B: POKE 769,50: CALL
    : 770: POKE 768,1: POKE 769,10: CALL 770: NEXT : POKE 7
    : 68,76 + 4 * B: POKE 769,50: CALL 770: RETURN
350 REM ASCENDING NOTES
400 FOR B = 6 TO 1 STEP -1: POKE 768,56 + 4 * B: POKE 7
    : 69,50: CALL 770: POKE 768,1: POKE 769,10: CALL 770: NEXT
    : POKE 768,56 + 4 * B: POKE 769,50: CALL 770: RETURN
450 REM SOUND FOR WRONG ANSWER
500 FOR B = 1 TO 50: POKE 768,200: POKE 769,5: CALL 770: NEXT
    : RETURN
550 REM SOUND FOR RIGHT ANSWER
600 POKE 768,102: POKE 769,90: CALL 770
650 POKE 768,1: POKE 769,10: CALL 770
700 POKE 768,102: POKE 769,50: CALL 770
750 POKE 768,1: POKE 769,20: CALL 770
800 POKE 768,102: POKE 769,10: CALL 770
850 POKE 768,1: POKE 769,10: CALL 770
900 POKE 768,76: POKE 769,230: CALL 770
950 RETURN
1000 REM SOUND FOR EXCELLENT PERFORMANCE
1050 FOR B = 1 TO 20: POKE 768,64: POKE 769,10: CALL 770:
    : POKE 768,72: POKE 769,10: CALL 770: NEXT : RETURN
1100 ONERR GOTO 11900
1150 TEXT : NORMAL : NOTRACE : SPEED= 255
1200 CALL - 936
1250 VTAB 8: HTAB 12: PRINT "APPLE SPRINT"
1300 VTAB 11: HTAB 5: PRINT "BY KEN AND MICHELLE MCCANDLE
    : SS"
1310 VTAB 13: HTAB 9: PRINT "COPYRIGHT (C) 1982": VTAB 14
    : HTAB 9: PRINT "BY MICRO-SPARC, INC."
1325 BB = INT ( RND (1) * 10): FOR B = 1 TO BB: BC = RND
    (1): NEXT : REM INDUCE MORE RANDOMNESS
1350 DEF FN A(Z) = 21 - LEN (A$) / 2
1400 DEF FN H(Z) = INT ( RND (1) * 10) + 48
1450 SPEED= 255
1500 DIM KA$(6)
1550 DIM AZ(3),NZ(3),AD$(3)
1600 DIM AR$(2),PR$(43),ND$(50)
1650 DIM AD$(50),VB$(35)
1700 AD = 50:VB = 55
1750 AR = 2:PR = 43:ND = 50
1800 S$ = " "
1850 G$ = CHR$ (7)
1900 Q = 22: REM # OF DATA STATEMENTS OF INSTRUCTIONS
1950 REM BUILD THE DATA ARRAYS
2000 FOR B = 1 TO AR: READ AR$(B): NEXT B
2050 FOR B = 1 TO PR: READ PR$(B): NEXT B
2100 FOR B = 1 TO ND: READ ND$(B): NEXT B
2150 FOR B = 1 TO AD: READ AD$(B): NEXT B
2200 FOR B = 1 TO VB: READ VB$(B): NEXT B
2250 REM GET CONTROL PARAMETERS
2300 VTAB 21: HTAB 10: PRINT "INSTRUCTIONS (Y/N)? ": GET
    : Z$
2350 IF Z$ = CHR$ (17) THEN END
2400 IF Z$ = "Y" THEN GOSUB 13200
2450 CALL - 936: VTAB 10: HTAB 11: PRINT "PRESCHOOLER? (
    : Y/N)": GET Z$

```

```

2500 K = 0
2550 IF Z$ = "Y" THEN K = 1
2600 IF Z$ = CHR$ (17) THEN 12250
2650 T = 0: S = 0
2700 CALL - 936: VTAB 10: HTAB 13
2750 H = 0
2800 PRINT "NUMBERS? (Y/N)": GET Z$
2850 IF Z$ = CHR$ (17) THEN 12250
2900 IF Z$ = CHR$ (27) THEN 2450
2950 IF Z$ = "Y" THEN H = 1
3000 CALL - 936
3050 VTAB 10: HTAB 5: PRINT "LEVEL? (0=EASIEST/4=HARDEST)
    ": GET Z$
3100 IF Z$ = CHR$ (17) THEN 12250
3150 IF Z$ = CHR$ (27) THEN 2700
3200 L = INT ( ASC (Z$) - 48)
3250 IF L < 0 OR L > 4 THEN L = 2
3300 CALL - 936
3350 VTAB 10: HTAB 5: PRINT "SPEED? (0=SLOWEST/9=FASTEST)
    ": GET Z$
3400 IF Z$ = CHR$ (17) THEN 12250
3450 IF Z$ = CHR$ (27) THEN 3000
3500 X = INT ( ASC (Z$) - 48): XX = 9 - X
3550 IF X < 0 OR X > 9 THEN X = 4: XX = 5
3600 CT = 0: WC = 0: W = 0
3650 REM BUILD THE DISPLAY
3700 CALL - 936
3750 VTAB 1: HTAB 1: INVERSE : PRINT "LEVEL=": L
3800 VTAB 1: HTAB 13: NORMAL : PRINT "READING IMPROVER"
3850 INVERSE : VTAB 1: HTAB 34: PRINT "SPEED=": X: NORMAL
    : HTAB 1
3900 IF K = 1 THEN VTAB 3: HTAB 16: INVERSE : PRINT "PRE
    : SCHOOL": NORMAL
3950 VTAB 24: HTAB 5: PRINT "TO QUIT-HIT <CTRL>-Q THEN RE
    : TURN": HTAB 1
4000 VTAB 8: PRINT "-----"
4050 VTAB 9: PRINT "I": SPC( 38): "I"
4100 VTAB 10: PRINT "I": SPC( 38): "I"
4150 VTAB 11: PRINT "I": SPC( 38): "I"
4200 VTAB 12: PRINT "-----"
4250 B = FRE (0)
4300 G2 = 2: T1 = 1
4350 REM PRESCHOOLER
4400 IF K < > 1 THEN 5000
4450 KI$ = "": A$ = ""
4500 IF H < 1 THEN 4800
4550 REM PRESCHOOLER NUMBERS
4600 FOR B = 1 TO L + 2: KA$(B) = CHR$ (( RND (1) * 10) +
    : 48): KI$ = KI$ + KA$(B): A$ = A$ + KA$(B) + " ": NEXT
4650 B = LEN (A$) - 2: A$ = MID$ (A$,1,B)
4700 GOTO 8350
4750 REM PRESCHOOLER WORDS
4800 FOR B = 1 TO L + 2: KA$(B) = MID$ (VB$( RND (1) * (V
    : B - 1) + 1), ( RND (1) * 2 + 1), 1): KI$ = KI$ + KA$(B):
    : A$ = A$ + KA$(B) + " ": NEXT
4850 B = LEN (A$): A$ = MID$ (A$,1,B)
4900 GOTO 8350
4950 REM WORDS OR NUMBERS?
5000 IF H < 1 THEN GOTO 5600
5050 REM DOING NUMBERS
5100 IF L = 4 THEN BB = 12
5150 IF L = 3 THEN BB = 10
5200 IF L = 2 THEN BB = 8
5250 IF L = 1 THEN BB = 6
5300 IF L = 0 THEN BB = 4
5350 IF RND (1) > .65 THEN BB = BB + 1
5400 A$ = ""
5450 FOR B = 1 TO BB: A$ = A$ + CHR$ ( FN H(Z)): NEXT B
5500 GOTO 8350
5550 REM RANDOMIZE 3 ARTICLES

```

continued on next page

Apple SPRINT (Speed Reading Improvement Technique) (Cont.)

```

5600 FOR B = 1 TO 3
5650 AZ(B) = RND (1) * 10
5700 IF AZ(B) < 5 THEN AZ(B) = 1: NEXT B: GOTO 5900
5750 AZ(B) = 2: NEXT B
5800 REM RANDOMIZE 3 NOUNS
5900 FOR B = 1 TO 3
5950 NZ(B) = RND (1) * NO + 1
6000 IF NZ(B) < 1 THEN NZ(B) = 1
6050 IF (NZ(B) > NO) THEN NZ(B) = NO
6100 NEXT B
6150 REM RANDOMIZE A PREPOSITION
6200 PZ = RND (1) * PR + 1
6250 IF PZ < 1 THEN PZ = 1
6300 IF PZ > PR THEN PZ = PR
6350 REM RANDOMIZE 3 ADJECTIVES
6400 FOR B = 1 TO 3
6450 ADZ(B) = RND (1) * AD + 1
6500 IF ADZ(B) < 1 THEN ADZ(B) = 1
6550 IF ADZ(B) > AD THEN ADZ(B) = AD
6600 NEXT B
6650 REM RANDOMIZE VERB
6700 VZ = RND (1) * VB + 1
6750 IF VZ < 1 THEN VZ = 1
6800 IF VZ > VB THEN VZ = VB
6850 IF L > 0 THEN VZ = 7000
6900 A$ = PR$(PZ) + S$ + AR$(AZ(1)) + S$ + NO$(NZ(1))
6950 GOTO 8350
7000 IF L > 1 THEN 7150
7050 A$ = PR$(PZ) + S$ + AR$(AZ(1)) + S$ + AD$(ADZ(1)) + S$ + NO$(NZ(1))
7100 GOTO 8350
7150 IF L > 2 THEN 7350
7200 A$ = AR$(AZ(1)) + S$ + NO$(NZ(1)) + S$ + VB$(VZ) + S$ + AR$(AZ(2)) + S$ + NO$(NZ(2))
7250 IF LEN (A$) < 38 THEN 8350
7300 GOTO 6900
7350 IF L > 3 THEN 7650
7400 A$ = AR$(AZ(1)) + S$ + AD$(ADZ(1)) + S$ + NO$(NZ(1)) + S$ + VB$(VZ) + S$ + AR$(AZ(2)) + S$ + AD$(ADZ(2)) + S$ + NO$(NZ(2))
7450 IF LEN (A$) < 38 THEN 8350
7500 A$ = AR$(AZ(1)) + S$ + AD$(ADZ(1)) + S$ + NO$(NZ(1)) + S$ + VB$(VZ) + S$ + AR$(AZ(2)) + S$ + NO$(NZ(2))
7550 IF LEN (A$) < 38 THEN 8350
7600 GOTO 7200
7650 A$ = AR$(AZ(1)) + S$ + AD$(ADZ(1)) + S$ + NO$(NZ(1)) + S$ + VB$(VZ) + S$ + AR$(AZ(2)) + S$ + AD$(ADZ(2)) + S$ + NO$(NZ(2)) + S$ + PR$(PZ) + S$ + AR$(AZ(3)) + S$ + AD$(ADZ(3)) + S$ + NO$(NZ(3))
7700 IF LEN (A$) < 38 THEN 8350
7750 A$ = AR$(AZ(1)) + S$ + AD$(ADZ(1)) + S$ + NO$(NZ(1)) + S$ + VB$(VZ) + S$ + AR$(AZ(2)) + S$ + AD$(ADZ(2)) + S$ + NO$(NZ(2)) + S$ + PR$(PZ) + S$ + AR$(AZ(3)) + S$ + NO$(NZ(3))
7800 IF LEN (A$) < 38 THEN 8350
7850 A$ = AR$(AZ(1)) + S$ + AD$(ADZ(1)) + S$ + NO$(NZ(1)) + S$ + VB$(VZ) + S$ + AR$(AZ(2)) + S$ + NO$(NZ(2)) + S$ + PR$(PZ) + S$ + AR$(AZ(3)) + S$ + NO$(NZ(3))
7900 IF LEN (A$) < 38 THEN 8350
7950 A$ = AR$(AZ(1)) + S$ + NO$(NZ(1)) + S$ + VB$(VZ) + S$ + AR$(AZ(2)) + S$ + NO$(NZ(2)) + S$ + PR$(PZ) + S$ + AR$(AZ(3)) + S$ + NO$(NZ(3))
8000 IF LEN (A$) < 38 THEN 8350
8050 GOTO 7400
8100 REM FLASH THE DATA ON ANY
8150 REM KEY INPUT EXCEPT...
8200 REM CTRL-Q (QUIT) OR
8250 REM ESC (GET NEW CONTROL
8300 REM PARAMETERS)
8350 VTAB 21: HTAB 1
8400 POKE - 16368,0
8450 GET Z$
8500 IF Z$ = CHR$(27) THEN 2700
8550 IF Z$ = CHR$(17) THEN 12250
8600 IF T1 = 1 THEN T1 = 0: T = T + 1
8650 VTAB 10
8700 HTAB FN A(2)
8750 REM HOLD FLASH FOR SPEED-XX
8800 PRINT A$
8850 FOR B = 1 TO XX * 180: NEXT B: VTAB 10: HTAB 2: PRINT SPC(38)
8950 REM GET AND CHECK THE INPUT
9000 REM CTRL-Q IS TO QUIT
9050 POKE - 16368,0
9100 VTAB 21: HTAB 2: INPUT "?"; I$
9125 BB = INT (RND (1) * 10): FOR B = 1 TO BB: BC = RND (1): NEXT B: REM INDUCE MORE RANDOMNESS
9150 IF I$ = CHR$(17) THEN 12250
9200 IF K < > 1 THEN 9400
9250 REM CHECK FOR PRESCHOOLER INPUT
9300 IF I$ < > K1$ THEN 10100
9350 GOTO 9475
9400 IF I$ < > A$ THEN 10100
9450 REM CORRECT ANSWER
9475 BB = 0
9500 IF S2 = 1 THEN VTAB 20: HTAB 1: PRINT "1"; I1$: VTAB 21: HTAB 1: PRINT "2"; BB = 1
9550 VTAB 23: HTAB 12: FLASH: PRINT "*** CORRECT ***"
9600 GOSUB 600
9650 FOR B = 1 TO 500 + (300 * (L + 1) * BB): NEXT B
9700 NORMAL
9750 W = 0
9800 S = S + S2
9850 WC = 0: CT = CT + 1
9900 IF CT = 5 THEN GOSUB 11150
9950 GOTO 3700
10000 REM WRONG ANSWER

```

```

10100 VTAB 23: HTAB 10: FLASH: PRINT "*** WRONG ANSWER ***";: GOSUB 500
10150 FOR B = 1 TO 500: NEXT B
10200 NORMAL
10250 W = W + 1
10300 S2 = S2 - 1
10350 HTAB 1
10400 IF W = 1 THEN VTAB 21: PRINT SPC(119): I1$ = LEFT$(I$,38): GOTO 8350
10450 VTAB 10: HTAB FN A(2): PRINT A$
10500 PRINT G$: REM <CTRL>-G (BELL)
10550 VTAB 20: HTAB 1: PRINT "1"; I1$: VTAB 21: HTAB 1: PRINT "2";:
10600 POKE - 16368,0
10650 VTAB 23: HTAB 8: INVERSE: PRINT "HIT ANY KEY TO CO
NTINUE";: NORMAL: PRINT " ";: GET Z$
10700 IF Z$ = CHR$(17) THEN 12250
10750 IF Z$ = CHR$(27) THEN 2700
10800 CT = 0: WC = WC + 1
10850 IF WC = 3 THEN GOSUB 11450
10900 W = 0: GOTO 3700
10950 REM AUTO ADJUST BASED
11000 REM UPON PERFORMANCE
11050 REM
11100 REM DOING WELL-MAKE HARDER
11150 WC = 0: CT = 0
11200 IF X = 9 THEN 11300
11250 X = X + 1: XX = 9 - X: GOTO 400
11300 IF L = 4 THEN RETURN
11350 L = L + 1: X = 4: XX = 5: GOTO 400
11400 REM NOT DOING SO WELL-MAKE EASIER
11450 CT = 0: WC = 0
11500 IF L + X = 0 THEN RETURN
11550 IF X > 0 THEN X = X - 1: XX = 9 - X: GOTO 300
11650 IF L > 0 THEN L = L - 1: X = 4: XX = 5: GOTO 300
11750 L = 2: X = 4: XX = 5: GOSUB 300: GOTO 400: REM IMPOSSIBLE-BUT
11800 REM ERROR HANDLING
11850 REM ROUTINE
11900 ER = PEEK (222)
11950 IF ER = 53 THEN RESUME
12000 IF ER = 163 THEN RESUME
12050 IF ER = 254 THEN RESUME
12100 IF ER = 255 THEN 12250
12150 STOP
12200 REM ENDING ROUTINE
12250 CALL - 936: VTAB 1: HTAB 12: PRINT "READING IMPROV
ER": VTAB 10: HTAB 1
12300 PRINT "YOUR SCORE WAS "; S;
12350 PRINT "OUT OF A POSSIBLE "; T * 2;
12400 IF S = 0 OR T = 0 THEN 12900
12450 S = INT ((S / (T * 2)) * 100)
12500 PRINT "POINTS FOR A RANKING OF "; S; "%."
12550 PRINT
12600 IF S = 100 THEN PRINT "EXCELLENT! KEEP UP THE GOOD
WORK.": GOSUB 1050: GOTO 12900
12700 IF S > 74 THEN PRINT "VERY GOOD!": GOSUB 600: GOTO
12900
12750 IF S > 49 THEN PRINT "GOOD. TRY A LITTLE HARDER N
EXT TIME.": GOSUB 600: GOTO 12900
12800 PRINT "POOR. TRY HARDER NEXT TIME.": PRINT "YOU CA
N DO IT. I KNOW YOU CAN."
12850 GOSUB 500
12900 FOR B = 1 TO 750: NEXT B: VTAB 18
12950 PRINT: PRINT "WHEN YOU QUIT, YOUR ";
13000 INVERSE: PRINT "SPEED";: NORMAL: PRINT " WAS ";
13010 INVERSE: PRINT X
13050 NORMAL: PRINT "AND YOUR ";: INVERSE: PRINT "LEVEL
";: NORMAL: PRINT " WAS ";: INVERSE: PRINT L: NORMAL
13075 PRINT "USING THE ";: INVERSE: IF K = 1 THEN PRINT
"PRESCHOOL";
13080 IF K = 0 THEN PRINT "NORMAL";
13090 NORMAL: PRINT " FORMAT."
13100 END
13150 REM INSTRUCTION OUTPUT
13200 CALL - 936
13250 FOR B = 1 TO Q
13300 READ Q$: GOSUB 13400: NEXT B
13350 PC = 23: GOSUB 13850: RETURN
13400 REM FORMAT OUTPUT OF DATA IN Q$
13450 FOR CP = 1 TO LEN (Q$)
13500 FOR JP = CP TO LEN (Q$): IF MID$(Q$,JP,1) < > "
" THEN NEXT JP
13550 WP$ = MID$(Q$,CP,JP - CP): CP = JP: WP$ = WP$ + " "
13575 IF LEFT$(WP$,2) = "00" THEN GOSUB 13850: WP$ = ""
:PC = 23: GOSUB 13850
13600 IF LEFT$(WP$,1) = "0" THEN GOSUB 13850: WP$ = ""
13650 IF (LEN (LP$) + LEN (WP$)) > 39 THEN GOSUB 13850
13700 LP$ = LP$ + WP$
13750 NEXT CP
13800 RETURN
13850 POKE - 16368,0
13900 IF PC = 23 THEN VTAB 24: HTAB 8: PRINT "HIT ANY KE
Y TO CONTINUE";: PC = 0: GET Z$: HTAB 1: CALL - 936:
IF Z$ = CHR$(17) THEN END
13950 PRINT LP$: LP$ = "": PC = PC + 1: RETURN
14000 REM DATA FOR THE ARRAYS
14050 REM ARTICLES
14100 DATA A,THE
14150 REM PREPOSITIONS
14200 DATA AT,BY,IN,ON,NEAR,TO,FROM
14250 DATA DOWN,OFF,THROUGH,OUT,PAST,UP
14300 DATA OF,FOR,WITH,LIKE,ABOUT,ABOVE
14350 DATA ACROSS,AFTER,AGAINST,ALONG
14400 DATA AMONG,AROUND,BEFORE,BEHIND
14450 DATA BELOW,BENEATH,BESIDE,BETWEEN
14500 DATA BEYOND,DURING,EXCEPT,INSIDE
14550 DATA OUTSIDE,OVER,UNDER,INTO,ONTO
14600 DATA UPON,WITHIN,WITHOUT

```


14650 REM NOUNS
 14700 DATA HOUSE, TREE, BOAT, CAR, BICYCLE
 14750 DATA RAFT, LAKE, STREAM, RIVER, BUSH
 14800 DATA FLOWER, PLANT, SHACK, TENT, DESK
 14850 DATA PENCIL, PEN, DOOR, WALL, WINDOW
 14900 DATA CLOCK, BOOK, BIRD, DOG, CAT
 14950 DATA STREET, POLE, HILL, ROCK, PUDDLE
 15000 DATA KEY, VASE, PILLOW, MONEY, CAVE
 15050 DATA FORK, SPOON, DISH, CUP, GLASS
 15100 DATA SCHOOL, CLASS, SHIRT, PANTS, TIE
 15150 DATA CHAIR, SOFA, STEP, LADDER, STOVE
 15200 REM ADJECTIVES
 15250 DATA WET, RED, GREEN, BLUE, DRY
 15300 DATA SLIPPERY, LONG, FAT, SHORT, TALL
 15350 DATA SKINNY, BIG, SMALL, HUGE, LITTLE
 15400 DATA BLACK, BURNED, FAST, SLOW, ODD
 15450 DATA GROWING, SWEET, DULL, HARD, FRAMED
 15500 DATA SOFT, FAR, LOUD, QUIET, NEW
 15550 DATA OLD, ANCIENT, FRAGILE, BROKEN, WOODEN
 15600 DATA COLORFUL, WHITE, ROUGH, SMOOTH, STRIPED
 15650 DATA GLOWING, DEADLY, FUNNY, MESSY, SHARP
 15700 DATA DUMPY, HONEST, CROOKED, HOT, COLD
 15750 REM VERBS
 15800 DATA RAN, JUMPED, WALKED, SWAM, COVERED
 15850 DATA BROKE, FIXED, GOT, PUT, TOOK
 15900 DATA DROVE, PILOTED, PUSHED, SHOOK, STOPPED
 15950 DATA DRANK, ATE, CLIMBED, THREW, CAUGHT
 16000 DATA MADE, READ, CARRIED, DROPPED, REPLACED
 16050 DATA TICKLED, PULLED, LIFTED, TOSSED, FLEW
 16100 DATA EXPLODED, STACKED, PLACED, RODE, PLAYED
 16150 DATA WROTE, HUNG, PLANTED, BURIED, REMOVED
 16200 DATA WATCHED, TOLD, SWUNG, ATTACKED, KILLED
 16250 DATA KISSED, HELD, KICKED, HIT, TRIPPED
 16300 DATA BURNED, FIXED, REPAIRED, DESTROYED, ENTERED
 16350 REM DATA FOR INSTRUCTIONS
 16400 DATA " THIS PROGRAM WILL AID IN INCREASING YOUR READING SPEED AND COMPREHENSION. @ @"
 16450 DATA " DATA WILL BE FLASHED IN A BOX ON THE SCREEN AND YOU MUST TYPE IN THIS DATA TO GET IT CORRECT. @ @"
 16500 DATA " THE DATA CONSISTS OF EITHER NUMBERS OR SENTENCES (IN THE CASE FOR A PRESCHOOL CHILD, JUST LETTERS). THE SENTENCES ARE, FOR THE MOST PART, NONSENSICAL, BUT GRAMMATICALLY CORRECT. @ @"
 16550 DATA " THERE ARE TWO FORMATS AVAILABLE, A NORMAL FORMAT AND A FORMAT DESIGNED FOR THE PRESCHOOL CHILD"
 16600 DATA "WHO CAN READ THE ALPHABET AND/OR NUMBERS. THE RESPONSE TO THE QUESTION 'PRESCHOOLER' SELECTS THE FORMAT. @ @"
 16650 DATA " FOLLOWING THE FORMAT QUESTION YOU ARE ASKED FOR THE TYPE OF DATA (NUMBERS? (Y/N)), DIFFICULTY LEVEL"
 16700 DATA AND SPEED AT WHICH YOU WILL ATTEMPT READING @ @
 16750 DATA " (NOTE: HITTING <ESC> AT ANY POINT DURING THIS QUESTIONING WILL SEND YOU TO THE PREVIOUS QUESTION.) @ @"
 16800 DATA " WHEN ANSWERING THESE QUESTIONS, HITTING ANY CHARACTER OTHER THAN THOSE SPECIFIED WILL CAUSE THAT QUESTION TO"
 16850 DATA " ACCEPT THE FOLLOWING DEFAULTS: @ @ INSTR UCTIONS = N @ PRESCHOOLER = N @ NUMBERS = N @ L EVEL = 2 @ SPEED = 4 @ @"
 16900 DATA " FOLLOWING THE SET-UP, YOU WILL SEE THE LEVEL AND SPEED, AT WHICH YOU ARE ATTEMPTING, DISPLAYED. BELOW THIS YOU WILL SEE THE BOX IN WHICH THE DATA WILL BE FLASHED. @ @"
 16950 DATA " BELOW THE BOX DRAWN ON THE SCREEN YOU WILL SEE A FLASHING CURSOR. THIS IS YOUR READY CUE."
 17000 DATA "WHEN YOU ARE READY TO READ, HIT ANY KEY AND THE DATA WILL BE FLASHED. @ @"
 17050 DATA " AFTER THE DATA HAS BEEN FLASHED AND YOU HAVE READ IT, A QUESTION MARK (?) WILL APPEAR."
 17100 DATA THIS IS YOUR CUE TO TYPE IN WHAT YOU READ. @
 17150 DATA " (NOTE: IN THE PRESCHOOLER FORMAT, SPACING IS IGNORED. JUST TYPE IN THE DATA.) @ @"
 17200 DATA " YOU WILL HAVE TWO CHANCES AT GETTING IT CORRECT. IF YOU"
 17250 DATA "DON'T YOU WILL BE SHOWN THE DATA YOU MISSED AND YOUR RESPONSES. @ @"
 17300 DATA " THE PROGRAM WILL AUTOMATICALLY ADJUST THE DIFFICULTY LEVEL AND SPEED BASED UPON YOUR PERFORMANCE "
 17350 DATA " IF YOU WISH TO MAKE AN ADJUSTMENT TO THE DATA TYPE, LEVEL AND SPEED; JUST HIT THE <ESC> KEY. @ @"
 17400 DATA " YOU ARE SCORED AS FOLLOWS: @ 2 POINTS IF CORRECT ON THE FIRST TRY @ 1 POINT IF CORRECT ON THE SECOND TRY @ 0 POINTS IF WRONG ON BOTH TRIES @ @
 17450 DATA " TO QUIT, JUST HIT <CTRL>-Q. @ @



TIPS 'N TECHNIQUES

Disk Dump

by David Krathwohl
19 Tri St.
Ashland, MA 01721

During a couple of recent projects, I found myself with a disk full of relatively short programs for which I needed printed listings. Rather than face an afternoon of loading and listing each one, and in anticipation of similar projects, I developed a program to do the work for me. The result is Disk Dump, the lazy person's way of obtaining a printed listing of any or all of the programs on a disk by running just a single program.

Essentially, Disk Dump performs four operations:

1. It reads the disk directory into memory.
2. It checks the directory entries for Apple-soft programs.
3. It creates a text file containing alternating load and list commands.
4. It EXEC's the text file, causing the computer to load and list each program.

To accomplish the first task, Disk Dump relies on the RWTS subroutine. This is the "Read-Write-Track-Sector" subroutine which DOS uses to access the disk. For a complete explanation of RWTS, there are two excellent sources. One is the DOS manual on pages 94-98, and the other is the Disk Snooping series in Nibble (see the Nibble Express, Volume 2).

THE RWTS

The RWTS subroutine consists of three parts, all written in machine language:

1. The Controlling Routine which controls the jump to:
2. The IOB (Input/Output Block) which holds information such as the track and sector

of the disk being accessed, the operation (reading or writing) to be performed, and the address of the data buffer which is to be either read from or written to.

3. The Device Characteristics Table which holds technical information about the timing of the disk drive.

Disk Dump uses decimal data and a loop (at line 460) to poke these three machine language blocks into memory starting at 768 (decimal). Disk Dump is written for Disk Drive 1 in slot 6 (DOS 3.3). To change this would require changing only two of the data entries, the second "96" and the "1" immediately following it. See the DOS manual (p. 95) for details.

READING THE DISK DIRECTORY

After setting up the RWTS subroutine, the program uses a loop to step through the sectors of the disk directory on track 17 (\$11) (at line 60). It first reads a sector into a data buffer starting at 8192 (\$2000) by executing a CALL 768 (at line 80). It then increments a counter for the starting address of the data buffer (SB) and pokes the new address into the IOB (at line 100). Next it loops back and reads the next sector from the disk.

Having transferred the contents of track 17 to memory starting at location 8192, the next task is to sort through the data and find the names of all Applesoft programs. These will then be stored in an array (CTS(n)).

Each sector from the disk (256 bytes) and therefore each data segment (256 bytes) starting at 8192 contains space enough for 7 file names. Preceding each file name (maximum of 30 bytes) are the codes which indicate the file type (binary, text, or Integer) and whether a particular file has been deleted. Since the total space allotted for each file is 35 bytes, the search loop is set with an increment of 35 (at line 130). The index value (I) is set at the byte immediately preceding the

start of the first file name (13 bytes into the sector).

FILE CHECKING

Disk Dump first checks to see if a file has been deleted (at line 150). This is done by PEEK-ing the second byte preceding the index value (I-2). If it contains a value of 255 (\$FF), the file has been deleted and Disk Dump increments to the next file space.

The program next checks the byte at the index value (I) for a value of 132 (\$82) or 2 (\$2) (at line 160). A value of 132 is used to indicate a locked Applesoft program, while a value of 2 is used to indicate an unlocked Applesoft file. If either value is found, the program loops through the entire space allocated to the file name (regardless of the actual length of the name) and uses PEEKs and concatenation to build that name into the CTS(n) array. The variable EN counts the number of Applesoft entries.

When an entire sector's worth of information has been examined, the program increments to the next data buffer segment and repeats the process.

After building the CTS(n) array with program names, Disk Dump offers the user the option of dumping the whole disk or of selecting particular files to be dumped. It then opens a text file called "Dumper" and writes alternating LOAD and LIST commands to it which include the names of the programs. Notice that in line 370, CHR\$(34) has to be used to write quotation marks to the text file.

Line 400 writes a form feed command to the text file suitable for an Epson MX-80. This line may be modified or eliminated according to your printer's needs. Line 420 writes a command to the text file which will cause it to delete itself when its work is done.

Finally, the text file is EXEC'ed, causing the programs selected to be loaded and listed automatically.

LIST

```
1 REM *****
2 REM * DISK DUMP *
3 REM * BY DAVID KRATHWOHL *
4 REM * COPYRIGHT (C) 1982 *
5 REM * BY MICRO-SPARC INC *
6 REM * LINCOLN, MA. 01723 *
7 REM * ALL RIGHTS RESERVED *
8 REM *****
40 HOME : VTAB 10: HTAB 10: INVERSE : PRINT " READING
    DIRECTORY ": NORMAL
50 GOSUB 460: REM POKE IOB INTO MEMORY
60 FOR J = 15 TO 0 STEP - 1
70 POKE 783,J: REM STEPS THROUGH SECTORS
80 CALL 768
90 SB = SB + 1
100 POKE 787,32 + SB
110 NEXT J
120 DIM CT$(150)
130 DB = 8192:CO = 0:EN = 0:TE = 0
140 FOR I = DB + 13 + CO * 256 TO DB + (CO + 1) * 256
    STEP 35
150 DE = PEEK (I - 2): IF DE = 255 THEN 170: REM DEL
    ETED FILE
160 TE = PEEK (I): IF TE = 130 OR TE = 2 THEN 200: REM
    APPLESOFT FILE
170 TE = 0: NEXT I
180 CO = CO + 1: IF CO = 15 THEN 250
190 GOTO 140
200 FOR J = 1 TO 30:CH = PEEK (I + J):CT$(EN) = CT$(
    EN) + CHR$(CH): NEXT J
210 EN = EN + 1
```

```
220 TE = 0: NEXT I
230 CO = CO + 1: IF CO = 15 THEN 250
240 GOTO 140
250 HOME : VTAB 10: INPUT "DO YOU WANT TO DUMP THE WH
    OLE DISK?(Y/N)";AN$
260 IF LEFT$(AN$,1) = "Y" THEN PD = EN: GOTO 310
270 FOR K = 0 TO EN - 1
280 HOME : VTAB 10: PRINT "DO YOU WANT TO DUMP:": PRINT
    CT$(K): INPUT "(Y/N)?" ; AN$
290 IF LEFT$(AN$,1) = "Y" THEN CT$(PD) = CT$(K): PD =
    PD + 1
300 NEXT K
310 D$ = CHR$(4)
320 PRINT D$"OPEN DUMPER"
330 PRINT D$"DELETE DUMPER"
340 PRINT D$"OPEN DUMPER"
350 PRINT D$"WRITE DUMPER"
360 FOR K = 0 TO PD - 1
370 PRINT "PRINT "; CHR$(34);CT$(K); CHR$(34)
380 PRINT "LOAD ";CT$(K)
390 PRINT "LIST"
400 PRINT "PRINT CHR$(140):REM FORM FEED ON MX-80
410 NEXT K
420 PRINT "DELETE DUMPER"
430 PRINT D$"CLOSE DUMPER"
440 PRINT D$"EXEC DUMPER"
450 END
460 FOR I = 0 TO 30: READ ML: POKE 768 + I,ML: NEXT I
470 DATA 169,3,160,10,32,217,3,96,0,0,1,96,1,0,17
    ,0,27,3,0,32,0,0,1,0,0,96,1,0,1,239,216
480 RETURN
```


Hi-Res Pseudo-Colors

by Rob Smythe
1522 Rusholme Cres
Burlington, Ontario
Canada L7M 1M6

Did you know that you can get more high resolution colors than those mentioned in the Applesoft manual? While it is true that the command **HCOLOR = #** will only accept integers from 0 to 8, it turns out that you can get **256 different "colors"** by **Poking** different numbers into **location 228**. When you set **HCOLOR**, the following values get placed into that location.

HCOLOR	228 CONTAINS
0	0
1	42
2	85
3	127
4	128
5	170
6	213
7	255

So instead of using **HCOLOR = 5** to get **orange**, for example, you could **POKE 228,170** instead. The interesting thing is that although errors result trying to make **HCOLOR** greater than 7, **you can put any value from 0 to 255 into 228 and get some pretty weird colors (or color patterns)**.

Here is a little program to illustrate the effects you can produce. The display is a chessboard. Between the red squares you will see the "pseudo-colors" that can be obtained. Some of the plaid effects are quite beautiful.

One of the possible uses of these "colors" could be for displaying game board and markers so that the viewer can distinguish between them on a black and white console.

Type the program in and try it. Change the value in **line 2** if you wish to start with one of the other standard colors in the comparison squares.

LIST

```

2 REM *****
3 REM # PSEUDO COLORS
4 REM # BY ROB SMYTHE
5 REM # COPYRIGHT (C) 1982
6 REM # BY MICRO-SPARC INC
7 REM # LINCOLN, MA. 01773
8 REM *****
10 REM PSEUDO COLORS

20 REM R. M. SMYTHE

30 COL = 170
40 A = 154: HOME
50 HGR
60 POKE 228,COL
70 FOR I = 1 TO 20
80 FOR J = 1 TO 3
90 HPLLOT 40 # J,20 + I TO 40 # J
  + 20,20 + I
100 NEXT J,I
110 POKE 228,A
120 FOR I = 1 TO 20
130 FOR J = 1 TO 3
140 HPLLOT 40 # J + 20,20 + I TO
  40 # J + 40,20 + I
150 NEXT J,I
160 REM ENTER NUMBER BETWEEN 0
  AND 127 TO ACCOMPANY GREEN O
  R BLUE SQUARES AND A NUMBER
  BETWEEN 128 AND 255 TO USE W
  ITH ORANGE OR VIOLET."
170 VTAB 22: IF COL < 127 THEN INPUT
  "NEW CHOICE (0 ... 127) ? ";
  A: GOTO 50
180 INPUT "NEW CHOICE (128 TO 25
  5) ? ";A
190 GOTO 50

```

OILER

By Glen E. Bredon
521 State Rd.
Princeton, N.J. 08540

It may come as a bit of a surprise, but the Apple DOS binary file load routines are somewhat inefficient. This is not ordinarily much of a problem, but if you spend a great deal of time using a text processor or assembler which uses binary files, you begin to long for more efficient BLOADs. This is the purpose of the OILER program.

OILER sets up a DOS patch to the BLOAD routine which will cause all BLOADs to be processed over **3 times faster** than the standard routines. It is totally transparent to the user and will dramatically speed up file loading for word processors such as the Apple Writer or assemblers such as Merlin. It does not use any memory locations aside from those ordinarily used by DOS.

OILER is designed for use with standard DOS 3.3 in a 48K system. Other configurations would require some changes to the program.

DESCRIPTION OF PROGRAM ACTION

To initialize OILER, type **BRUN OILER**. This will establish the BLOAD patch in an area of DOS that is not used after a boot, and it places a jump at an appropriate place in the BLOAD routine. Because of the change in DOS, an INIT would not be possible, so the **INIT command is disabled**. At this point control is returned to the user. Note that since the Apple copy program COPYA does an INIT, you will have to reboot before using that program. Some other copy programs would not require this, and thus may be used.

When a BLOAD occurs, the patch checks to see if the length of the file is less than one page, or more than 120 pages (more than **\$78FF** in length—approximately **122 sectors**). If so, then the old BLOAD routine is used. The reason that files over 120 pages long cannot be handled is that such files need more than one "track/sector list" and there is not enough room in the patch to be able to handle this complication. Since binary files of this length are rare, this limitation is not very serious.

For files of the appropriate length, the patch proceeds to read the sectors of the file directly into memory using the **RWTS (Read/Write Track and Sector)** routine in DOS. The last sector of the file is processed differently in order to avoid overwrite of memory beyond the specified end of the file.

The patch does need some zero page locations for its own use, but the contents of these locations are saved on the stack upon entry and replaced upon exit, so that no zero page conflicts can arise.

COMMENTS ON THE ASSEMBLER LISTING

The assembler used for this program is Merlin, which is a new macro assembler with some advanced features not found in other assemblers. The following remarks will aid users of other assemblers in transcribing the program. The **ERR** opcode at the end is to ensure that the program is not too long for the space allotted to it. On another assembler this will have to be checked by hand. The **CHK** opcode near the end is a checksum byte which enables you to check your coding against the original. This will have to be replaced with **END EQU *** on other assemblers

and you will have to exercise greater care in writing the source. The code is designed to be assembled in one block to load to address **\$2DE through \$3B3 (or \$3B2 without the checksum byte)**. To do this on many assemblers you will have to replace the line **NOWADRS ORG \$B700** with:

```

NOWADRS EQU *
NEWBL EQU $B700

```

and remove the label **NEWBL** from the next line. In the present case this will not affect the code since it is entirely in relocatable form. The opcode **>>>** is a macro call. On Merlin you need only type the **>>>** line and then skip to the line following the **<<<**. On other assemblers, ignore the **>>>** and **<<<** lines and type the lines between (and omit the macro definitions at the start).

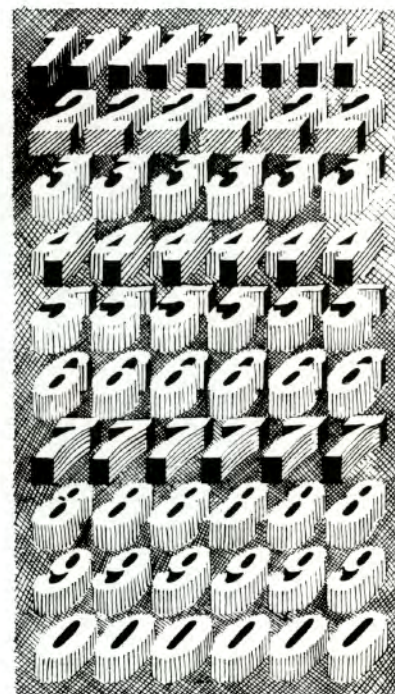
If your assembler cannot save the object code with the correct loading address, you should save the object code, then **BLOAD OILER,\$2DE,<RTN>** and then **BSAVE OILER,\$2DE,L\$D5,<RTN>**. (This ignores the checksum byte, which is not used by the program itself.)

DIRECT MEMORY ENTRY

Without an assembler you can enter the program into memory using the monitor's memory change by typing the part of the listing to the left of the line numbers with a RETURN at the end of each line. However, in the lines starting with "B7", the "B7" should be replaced with "3". When done, type **BSAVE OILER,\$2DE,L\$D5**. Then BLOAD OILER and compare the program with the listing very carefully.

It is wise to test the program out first using a scratch diskette. Any error in a program of this nature could have serious consequences.

continued on next page



Oiler (Cont.)

```

1 *****
2 *
3 *           OILER           *   Copyright (c) 1982
4 *           *               *   by Micro-SPARC Inc.
5 *   Fast Binary File Loader *   Lincoln, MA, 01773
6 *           *               *   All Rights Reserved
7 *   Glen E. Bredon 3/22/82 *
8 *
9 *****
10 *
11 * When this is BRUN it sets up *
12 * a DOS patch. Thereafter, *
13 * any BLOAD or BRUN of a *
14 * binary file will load about *
15 * 3 or more times faster (with *
16 * the exception of those over *
17 * $78FF length - about 123 *
18 * sectors - which are loaded *
19 * the old way because of space *
20 * constraints in this patch). *
21 *
22 * This patch is transparent to *
23 * the user. No memory is used *
24 * beyond normal DOS usage. *
25 *
26 *****
27
28
29
30 TSBUFFPTR = 0           ;T/S list pointer
31 DATAPTR = 2            ;Data buffer pointer
32 DESTPTR = 4            ;Temporarily holds BADR
33 NUMPAGES = 4           ;Countdown # of pages
34 TSPTR = 5             ;Offset to next T/S pair
35 RWTS = $B7B5
36 IOB = $B7E8
37 BLEN = $AA60
38 BADR = $AA72
39 DOSERR = $A6D2
40 PNTRS = $B5C9          ;Current DOS buffer addresses
41 INITCHD = $A884        ;"INIT" command word
42 BLJ = $A38C            ;Where patch is made
43 RR = $A471             ;Where it used to go
44 CLOSE = $A2EA
45
46 * Data:
47
48 VOLOFF = 3             ;Offsets from IOB start
49 TRACKOFF = 4           ;"
50 BOFF = 8               ;"
51 FIRSTS = $C            ;Offset to first T/S pair
52
53 *****
54 * Macro definitions:
55
56 DO 0
57
58
59 TRANS MAC
60 LDA J1
61 STA J2
62 LDA J1+1
63 STA J2+1
64 <<<
65
66 GORWTS MAC
67 LDY $IOB
68 LDA $IOB
69 JSR RWTS
70 <<<
71
72 STADR MAC
73 LDA $J1
74 STA J2
75 LDA $J1
76 STA J2+1
77 <<<
78
79 LDIND MAC
80 LDA (J1),Y
81 STA J2
82 INY
83 LDA (J1),Y
84 STA J2+1
85 <<<
86
87 FIN
88
89 *****
90
91 ORG $2DE
92
93 LDA INITCHD ;If INIT command is there
94 CMP #'I' ;then defeat it.
95 BNE SETADRS
96 LDA #' '
97 STA INITCHD
98 >>> STADR,NEWBL:BLJ
99 LDA $NEWBL
100 STA BLJ
101 LDA $NEWBL
102 STA BLJ+1
103 <<<
104 LDY $END-NEWBL ;Move this program to $B700
105 MOVE LDA NOWADRS-1,Y
106 STA NEWBL-1,Y
107 DEY
108 BNE MOVE
109 RTS ;Back to caller
110
111 NOWADRS ORG $B700

```

```

B700: AD 61 AA 107 NEWBL LDA BLEN+1 ;For files of length < $100
B703: F0 04 109 BEQ OLDWAY ;use former BLOAD routine
B705: C9 79 110 CMP #121 ;Same for length > $78FF
B707: 90 03 111 BLT NEWAY ; (approx 122 sectors)
B709: 4C 71 A4 112 JMP RR ;No room to handle these
113
B70C: A2 05 114 NEWAY LDX #5 ;Save some zero page
B70E: B5 00 115 SVZ LDA TSBUFFPTR,X
B710: 48 116 PHA
B711: BD C9 B5 117 LDA PNTRS,X ; and set up buffer pointers
B714: 95 00 118 STA TSBUFFPTR,X
B716: CA 119 DEX
B717: 10 F5 120 BPL SVZ
B719: AD 72 AA 121 LDA BADR ;Get destination address
B71C: 18 122 CLC
B71D: 69 FC 123 ADC #~4 ; add $100-4
B71F: 85 04 124 STA DESTPTR
B721: 8D F0 B7 125 STA IOB+BOFF
B724: A9 00 126 LDA #0
B726: 8D EB B7 127 STA IOB+VOLOFF ;Init "VOL 0"
B729: 6D 73 AA 128 ADC BADR+1
B72C: B5 05 129 STA DESTPTR+1
B72E: C6 05 130 DEC DESTPTR+1 ;Points to BADR-4
B730: 8D F1 B7 131 STA IOB+BOFF+1 ;Points to BADR+$100-4
B733: A0 FF 132 LDY #~1 ;Move the $100-4 bytes
B735: B1 02 133 FRST LDA (DATAPTR),Y ;now in data buffer
B737: 91 04 134 STA (DESTPTR),Y
B739: 88 135 DEY
B73A: C0 03 136 CPY #3
B73C: D0 F7 137 BNE FRST
B73E: AD 60 AA 138 LDA BLEN ;Calculate # of pages
B741: C9 FD 139 CMP #100-3 ;to be read directly
B743: AD 61 AA 140 LDA BLEN+1
B746: 69 00 141 ADC #0
B748: B5 04 142 STA NUMPAGES ;Put it in counter
B74A: A9 05 143 LDA $FIRSTS+2 ;Point to next T/S
B74C: B5 05 144 STA TSPTR
B74E: A4 05 145 MAINLOOP LDY TSPTR ;Get next T/S pair
146 >>> LDIND,TSBUFFPTR:IOB+TRACKOFF
B750: B1 00 146 LDA (TSBUFFPTR),Y
B752: BD EC B7 146 STA IOB+TRACKOFF
B755: C8 146 INY
B756: B1 00 146 LDA (TSBUFFPTR),Y
B758: BD ED B7 146 STA IOB+TRACKOFF+1
146 <<<
B75B: C8 147 INY
B75C: B4 05 148 STY TSPTR ; +2
B75E: C6 04 149 DEC NUMPAGES ;Are we at last sector?
B760: F0 0E 150 BEQ LASTSEC ;Branch if so.
151 >>> GORWTS ;Read sector directly
B762: A0 EB 151 LDY #IOB
B764: A9 B7 151 LDA $IOB
B766: 20 B5 B7 151 JSR RWTS
151 <<<
B769: EE F1 B7 152 INC IOB+BOFF+1 ;Next memory page
B76C: 90 E0 153 BCC MAINLOOP ;Loop if no RWTS error
B76E: B0 2C 154 BCS FINISH ;Abort and get error
155
156
157 * Special processing for last sector
158 * to avoid overwrite of memory:
159
160 LASTSEC >>> TRANS,IOB+BOFF:DESTPTR
B770: AD F0 B7 160 LDA IOB+BOFF
B773: B5 04 160 STA DESTPTR
B775: AD F1 B7 160 LDA IOB+BOFF+1
B778: B5 05 160 STA DESTPTR+1
160 <<<
161 >>> TRANS,DATAPTR:IOB+BOFF
B77A: A5 02 161 LDA DATAPTR
B77C: BD F0 B7 161 STA IOB+BOFF
B77F: A5 03 161 LDA DATAPTR+1
B781: BD F1 B7 161 STA IOB+BOFF+1
161 <<<
162 >>> GORWTS ;Read last sector to buffer
B784: A0 EB 162 LDY #IOB
B786: A9 B7 162 LDA $IOB
B788: 20 B5 B7 162 JSR RWTS
162 <<<
B78B: B0 0F 163 BCS FINISH ;Exit if RWTS error
B78D: AD 60 AA 164 LDA BLEN ;Calculate # of bytes to move
B790: 69 04 165 ADC #4
B792: A8 166 TAY
B793: 18 167 CLC ;Clear RWTS error flag
B794: 88 168 DEY ;then move required
B795: B1 02 169 LDA (DATAPTR),Y ;amount to memory.
B797: 91 04 170 STA (DESTPTR),Y
B799: 98 171 TYA
B79A: D0 F8 172 BNE TSLOOP ;Loop till all moved
173
B79C: A0 05 174 FINISH LDY #5 ;Replace zero page
B79E: A2 00 175 LDX #0
B7A0: B6 48 176 STX $48 ;Zero monitor's status save
B7A2: 68 177 ZBACK PLA
B7A3: 95 00 178 STA TSBUFFPTR,X
B7A5: E8 179 INX
B7A6: B8 180 DEY
B7A7: 10 F9 181 BPL ZBACK
B7A9: B0 03 182 BCS ERROR ;Branch if there was RWTS err
B7AB: 4C EA A2 183 JMP CLOSE ;otherwise close up & exit
184
B7AE: A9 08 185 ERROR LDA #8 ;I/O error
B7B0: 4C D2 A6 186 JMP DOSERR
187
B7B3: C2 188 END CHK ;This byte to compare codings
189 ; it is not moved
190 ERR END-1/RWTS

```

--End assembly--

214 bytes

APPLE A.R.C. SYSTEM

Applesoft Record Command System (ARC)

by Chuck Kiss
6504 Creighton Ave.
Huntsville, AL 35810

INTRODUCTION

The Nibble A.R.C. System (Applesoft Record Command System) is a major enhancement to and extension of Nibble's A.I.M. (Automated Intelligent Mailing System — Nibble Vol. 1 #7). It requires a 48K Apple II+ (with Applesoft) and one disk drive. It offers the following features:

- * Fast, Easy Screen Formatting of Records.
- * Simple Record Editing.
- * Fast Machine Language Record Search.
- * In-memory system for 50-125 record files.
- * Merging of two compatible files for sorting and hard-copy.
- * Support for the D.C. Hayes Micro-modem II.
- * Support for the Epson MX-70/80/100 printers with or without Grafrax-80, and other Centronics-compatible printers.

The A.R.C. System made up of the following four programs: INITIALIZE, SEARCH, A.R.C., and MERGE FILES. Their functions will become clear as you read further.

USING A.R.C.

Run A.R.C. (merely turn on the Apple with the A.R.C. disk in drive 1) and select option (1) in order to initialize a testfile. Select up to 12 fields and browse to your heart's content. After having initialized a file turn off the Apple and turn it back on.

At this time selection of options (2) thru (6) from the command menu will result in nothing happening because there currently is no file in memory. In order to specify an initial file (or any new file) you must hit ESC. The words FETCH FILE will appear in the lower left of the menu. Specifying option (2) now will cause the disk to be catalogued and you will be queried for a file name. Enter your testfile's name. You are now ready to enter data. Enter about 10 records worth of data so that you can exercise the various modules and get better acquainted with the program. Note again that options (1) and (7) are not restricted by what is or isn't in memory since these two options run separate programs.

Usage of the A.R.C. System is predominantly straight-forward and self-explanatory. As codes, the Printer should be in Slot #1 and the D.C. Hayes modem in Slot #2. The disk drive is assumed to be in Slot #6/Drive #1. Also, Maxfiles is set to one in order to de-allocate valuable buffer space. Keep this in mind if you plan to renumber the program!

Figure 2 contains a list of A.R.C. parameters and their locations. In the event that you want to check out MERGE FILES at this time, delete line 540 so that you can merge your testfile with itself. If you do not want to use paddles, delete those statements that say SPEED=PDL(1).

PERSPECTIVE

Nibble's A.I.M. is good — very good! But I simply could not shake the idea of having all of the records of a file in memory (which A.I.M. did not do). Visions of immediate record access, faster review, searches, and sorts kept haunting me. But would an attempt at modification turn out to be an exercise in futility? After all, 48K Bytes is not exactly big system storage, and when one includes DOS' overhead as well as program size, you're not left with a heck of a lot of free memory! This did pose a problem.

In the context of A.R.C. the problem manifests itself as spontaneous and time-con-

suming FRE(0) housecleaning which is a direct consequence of Apple's limited memory. FRE(0) is the bugaboo that must be minimized. It can strike swiftly and silently at the most inopportune times making an otherwise efficient program appear lethargic.

It probably comes as no surprise to learn that an Apple in-memory system is incapable of handling files having on the order of 500+ records. Ah ha, but would it work for files having on the order of 50+ records? Some fast arithmetic convinced me that it would, and I was off and running. Now at this point you're probably muttering something to the effect — 50 records, Big Deal! Hang in there, the plot thickens.

ESTIMATING RECORD CAPACITY

Let's try to quantify the problem so we can better understand what we're coping with. It stands to reason that if a record contains, let's say, 50 characters as opposed to 240, we can

continued on next page

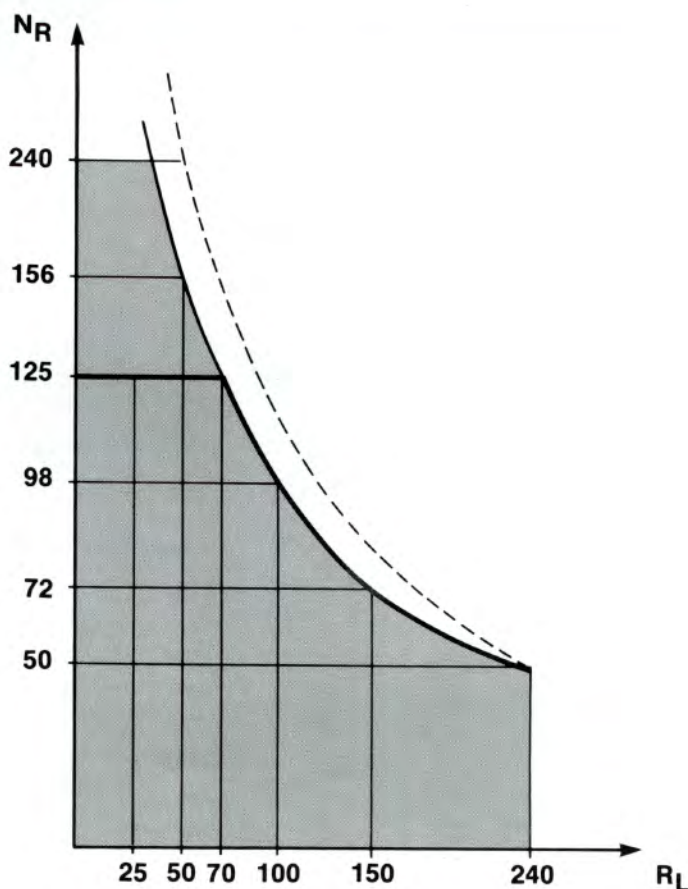


Figure 1.

Applesoft Record Command System (ARC) (Cont.)

shoehorn more of them into memory. In a system such as A.R.C. we would like to specify distinct fields. In order to keep things manageable and fit all fieldnames, data, and instructions on the text screen, we will limit ourselves to **12 fields**. If we allow ourselves **20 characters/field (c/f)** we can then have **up to 240 characters/record (c/r)**. Now in order to minimize FRE(0)'s duration and frequency it is crucial to have on the order of 8K Bytes of free memory available after loading the program and file data. Remember, the tradeoff is between record length and number of records in memory for a given FRE(0) level of performance. The selection of an 8K Byte buffer results in a memory balance sheet that looks something like this:

Program	16 K
DOS	10 K
Lomem	2 K
Memory Buffer	8 K
→File Data	12 K
	48 K

The resulting 12K Bytes of memory we have at our disposal for file data can be allocated in any way we desire. We can trade off **Record Length (RL)** and **Number of Records (NR)** as long as we don't exceed the 12K Byte constraint:

$$RL * NR = 12 \text{ K Bytes}$$

or

$$NR = \frac{12,000}{RL}$$

This mathematical representation is the equation of a hyperbola (don't worry about it), and is depicted in **Figure 1** as a dotted line. This graph can be interpreted as follows:

If we desire 12 fields of 20 characters each (RL=240), NR is limited to 50 records.

Alternatively, if we only desire 5 fields of 10 characters each (RL=50), NR expands upwards to 240 records.

Well, this is all well and good except for the fact that there is a certain amount of overhead in scrolling records and answering questions in the normal course of the program, all of which produce, you guessed it, garbage. So what we must include is a "safety factor" that is inversely proportional to NR. This means that we had better scale down our expectation of NR. In effect we do this by specifying:

$$NR = \frac{12,000}{RL + 30} + 6$$

In this way, we can maintain our original requirement of **50 records at RL=240** yet still attain **156 records at RL=50**. This equation is depicted as the **solid curve** in **Figure 1**. We're almost there but not quite.

As it turns out **A.R.C.** can call another program called **MERGE FILES**. **MERGE FILES**, as the name implies, can merge two compatible files for purposes of sorting. What this also means is that **MERGE FILES** must cope with up to **2 * 156** or **312** records without significant FRE(0) interruptions. I tried it. No Way! Although **A.R.C.** has no trouble with 156

records at 50 c/r, **MERGE FILES** cannot carry the 312 record load and gets bogged down repeatedly.

What we need is to **place an absolute limit on NR based on the requirements of MERGE FILES**. A bit of experimentation yields **NR = 125 for RL < 70** as a good compromise.

This then is the set of "rules" that must be obeyed when initializing files in order to guarantee that FRE(0) gremlins do not excessively interrupt program execution. Note that there WILL be internal FRE(0) housecleaning, but for the most part its occurrence is unobtrusive. Incidentally, **INITIALIZE** and **MERGE FILES** were originally part of **A.R.C.** In the interest of memory conservation they were made separate programs and are called from the **command menu (Figure 2)** of **A.R.C.** This is of no great concern since they are typically not required to load and run as often as **A.R.C.**, and when called their usage is transparent.

MERGE FILES

The reason for the existence of **MERGE FILES** should perhaps be explained. It was anticipated that there would inevitably arise a need to merge two files for purposes of sorting. Note that all other file manipulation is via **A.R.C.** on the individual files prior to merging.

The merged file is non-resident within the data base and is not maintained per se. For example, the following files with **RL = 100** are initialized: **NAMES(A-G)**, **NAMES(H-N)**, and **NAMES(O-Z)** where the first letter of a person's last name determines which file it will go into. This data base would allow up to a **3 * 98 = 294** record repertoire.

Given a full, 98 record, **NAMES(H-N)** file, the arrival of the 99-th name destined for that file would force the user to open a new file, presumably called **NAMES2(H-N)**. This precludes any sorting capability if **NAMES(H-N)** and **NAMES2(H-N)** cannot somehow be merged and the sorting performed on the merged file. In this manner, each of the original three files can be expanded resulting in a 588 record data base.

PLAN AHEAD

Even this hindrance can be avoided by merely **anticipating your needs in advance**. For instance, in operating within the **A.R.C.** limits the user can establish 26 files, one for each letter of the alphabet. This extreme example was used to drive home a practical restriction on data base size. While it is true that more records can be accommodated by a data base via **MERGE FILES** or by brute force constructs, such techniques have their limitations.

A large data base becomes impractical due to the low speed of currently available printers for the average home installation. The hard copy bottleneck is real and exists! While an Apple computer interfaced with an Epson MX-80 is a hard combination to beat, 80 cps hard copy on a 1000+ record **A.R.C.** data base is definitely not a ten minute endeavor.

As a practical necessity, it is recommended that a data base be limited to a maximum of 4 files (200-500 records depending on RL). Of course **any number of data bases** can be configured and maintained by **A.R.C.**, each on its own disk; for example, Boy Scout files on one disk, Girl Scout files on another, etc.

THE PROGRAMS

SEARCH

This is the **A.I.M.S.** Find Contest winning binary substring search (Nibble Vol. 2 #3). It is called from **A.R.C.** via the Ampersand operator.

INITIALIZE

This program is called to define and structure new files. Upon entry, a file name is requested. It cannot begin with an illegal DOS character. **Sixteen fields** are then presented for selection. The last six are options whose names are user-defined. Each field is used at most once, and **up to twelve fields** can be specified. Specifying a field highlights it on the screen. **If the optional fields are selected, a field name is requested.**



After specifying fields the user is queried for **field lengths**. This is a crucial step! Remember, **you're trading off RL for NR**, so it pays to consider this carefully. On completion, your record structure is displayed along with RL and the computed NR. If satisfactory, the file is initialized.



A header file is generated with a **CTRL-B** attached to the file name. This file is inviolate since it contains the file parameters **N**, **RL**, **NR**, and the field names.

All operations will be on the **second** initialized file that at this time merely contains **REC=0** and no data. After initialization, **A.R.C.** is automatically rerun with the Command Menu presented on the screen. If on the other hand, the file structure was unsatisfactory, program control will return you to a re-definition of field lengths, field selection, or exit to **A.R.C.**

A.R.C.

A.R.C. is the workhorse of the bunch and houses **options (2) - (6)** of the **command menu**. It has comprehensive error trapping.

A.R.C. is interesting in that it appears to re-dimension the arrays **RD\$, SRT\$, SO\$,**

and **C%**. It accomplishes this by merely rerunning A.R.C. each time a new file is fetched and **dynamically dimensions the arrays based on the file's NR**. The current file in memory is always displayed in inverse in the lower left of the menu. A.R.C. consists of the following five modules:

```

*****
| H E A R C S Y S T E M |
*****
[1] INITIALIZE NEW FILE
[2] ENTER FILE DATA
[3] REVIEW/EDIT DATA
[4] FIND/DIAL DATA
[5] SORT/PRINT DATA
[6] PURGE/PACK DATA
[7] MERGE/SORT FILES

<ESC> TO LOAD    CHOICE : [ ]
*****

```

Enter File Data — New data is entered here. Note that the cursor is maintained on the record field at all times. To input a null string (blank), hit RTN. After all entries to a record are made, corrections can be made or the record aborted. Note also, that the copy key (->) is disabled on the inverse screen. If **NR** records have been entered the user is notified. **RTN** transfers control back to the command menu.

```

*****
| R E V I E W |
*****
RECORD # [ ]
FULL NAME  SAR J SMITH
LAST NAME  SMITH
ADDRESS    111 NORTH AVENUE
CITY       CHICAGO
STATE      IL
ZIP CODE   60606
BIRTHDAY   1/24/55
Hobby      APPLE

ENTER NEW DATA

```

Review/Edit Data — All modification to existing records are made here. Printed copy of individual records via an Epson MX-series printer or other parallel printer, is also supported here. The -> key scrolls records forward while the <- key scrolls them backward. Use these keys with the REPT key and watch the data fly!

ESC requests a record to be fetched. If the record # specified is greater than the highest record # in memory, the last record is displayed and the user is notified.

"P" prints the record being displayed to the printer. **NOTE** — The user monitors his own Top-Of-Forms (this is handled automatically in the Sort/Print mode).

"F" searches for the first record containing the specified keyword from the current record being displayed (plus one) to the end of file. The user is queried for the search field and keyword. The search field is highlighted. See Find/Dial for the specific keyword syntax.

"E" edits records, in either a Modify or Erase mode. The cursor is maintained on

the 'record'. It allows erasure of a record but carries the record and shows it as erased by placing an asterisk (*) in Field #1. All other fields are blanked. Confirmation is made via a Yes/No response.

Find/Dial Data — The minimum capability any search subroutine should possess is to locate and retrieve information that is not uniquely addressable. This can be a single specific record, or many records that share a common attribute.

Find/Dial allows the user to search the entire file for the occurrence of his keyword, [K], in a specified field.

A keyword can be any combination of characters of a length less than or equal to the field length. A keyword preceded by an asterisk (*) indicates that the field searched must begin with the keyword.

A keyword not preceded by an asterisk (*) indicates that the field must merely contain the keyword. Note that a double asterisk (**) keyword for Field #1 will cause a search for erased records.

All record numbers containing the keyword are displayed, as well as the actual field data. If more than 12 records contain the keyword, the additional number of records correlated is shown.

If and only if one unambiguous record is returned, the option of dialing that record's phone field is made available. If so instructed, a check is then made to ascertain whether records for this file contain a phone number field and that a D.C. Hayes Micromodem II is in the system. The number is then dialed and a message flashed on the screen. If difficulty is encountered the Micromodem II will hang up automatically.

Sort/Print Data — When specifying fields for the label, inputting @ as the first entry automatically specifies all fields of the record. Sorting and line-merging options and conjunctives (joining consecutive fields together for printing) are user-selectable. A sort "window" has been incorporated enabling you to see the actual sort being performed. You have

continued on next page

Line #	Name/Value	Meaning
90,190,220,640	POKE 8,0-2	System Status: Initial Boot(0) ESC Pressed(1) File in Mem(2) & Jump Location for SEARCH
120	POKE 1013,76 POKE 1014,2 POKE 1015,3	
130	TS() TV%() PS() TX\$() TW%()	Field Names Field Lengths Field Data Conjunctives Field Labels
140	SX=6 DX=1 SP=25	Disk Slot # Disk Drive # Printer Tab (Review/Edit)
210	REC	# Records in File
400	NF RL NR	# Fields/Record Record Length File Capacity
430	RD\$() SRT\$() SO%() C%()	File Records Sorted Fields Pointer to Sorted Records Pointer to Erased Records
1370,3830,3880	PR#1 CHR\$(9);"80N"	Epson Enable 80 Col Printing
1950	MODEM=1	Modem in System (Otherwise 0)
2280	KEY\$	Search Keyword
2490	DUM\$	Field Data to be Searched
2500,2540	& DUM\$,KEY\$	Calls to SEARCH If Peek(26)=0 then No Match
2800,2820	NUMBERS	Phone Number (+ CTRL-J)
2810	PR#2	Modem Enable
2830	DIAL\$;NUMBERS	Modem Dial C'md DIAL\$=CHR\$(17)
2980	IDENT=25	Printer Tab (Sort/Print)
3120	TV\$="@	Specifies all Fields for Label
3880,3960	CHR\$(12)	Top-of-Form

Figure 2. A.R.C. Parameters

Applesoft Record Command System (ARC) (Cont.)

the choice of outputting the sorted, formatted data to the screen or to the printer. If the printer is chosen, the Top-Of-Forms is enabled. Note — Paddle 0 controls the data rate to both the screen and to the printer.

Purge/Pack Data — All erased records are carried within the file and can be purged at once. Hit **P** and the program does the rest. Housecleaning is performed and the amount of remaining free memory is displayed. If any records were purged from memory, the new file configuration will be stored to disk upon command.

MERGE FILES

This program merges and sorts two compatible files within the data base. To be compatible, the record length and the number of fields for each file **MUST** be identical. MERGE FILES, although a separate program, is substantially the same as option (5) of A.R.C. MERGE FILES will enable a data base to remain viable when one of its files has used up its NR record allocation. With proper file planning the use of MERGE FILES will be minimized. Upon completion, MERGE FILES will rerun A.R.C. and present you with the command menu.

LOADING A.R.C.

1. Clear the Apple's memory (**FP**). Enter the

A.R.C. program and initialize a new disk with **INIT A.R.C.**

2. Enter the monitor (**CALL -151**) and key in **SEARCH** beginning at **\$302**. Save to the disk with **BSAVE SEARCH, A\$302, L\$70**. For additional information on entering machine language code into memory, see the instructions in the Letters section of this issue.
3. Return to Applesoft by typing **CTRL C**. Roll up your sleeves and begin entering A.R.C. using the listing. You probably won't complete this in one sitting so be sure to back it up as you go along! When done **SAVE A.R.C.**
4. Enter **MERGE FILES** from the listing and save it on your disk.
5. Enter and save **INITIALIZE**.
6. Lock all files on the disk. Thereafter, you need only boot the disk to use the A.R.C. system.

CUSTOMIZATION

The sky is the limit. There is always room for improvement to suit each individual's needs. Listed below are a few fertile areas for customization:

- * Option for emphasized/expanded printing and variable margin justification (Italics also if you have Grafrax-80).
- * Compiled/Machine Language sort routine.

- * Faster file transfer using a machine language routine like Amper-Reader. (Nibble Express Vol. 2)
- * Search the entire record rather than only a specified field.
- * Automata: load a file, sort it, print it; load next file, sort it, print it; etc.
- * Modifications to maintain the merged file.
- * You can create your own field NAMES for the first 10 Fields in the **INITIALIZE** program by changing the DATA statement in **line 65** of **INITIALIZE**. You should make sure to include **at least 16** total items in the DATA statement.
- * Last but not least is the very exciting possibility of adapting A.R.C. to 16K RAM boards. This should not be too difficult and would significantly extend file capacity. (By replacing the 12K in the numerator of our equation with 28K, the NR-RL plot would be shifted upwards.)

The A.R.C. System has served to illustrate one technique of maintaining an "in-memory" DBMS. Its usefulness is made possible by its file merging capability and its ability to sequentially "block" a set of files into memory.

REFERENCES

1. A.I.M.S./Nibble Vol. 1 #7/1980, Nibble Express Vol. 1.
2. A.I.M.S./Contest Winner/Nibble #3/Vol. 2.



```

10 REM *****
20 REM * THE A.R.C. SYSTEM *
30 REM * BY CHUCK KISS *
40 REM * COPYRIGHT (C) 1982 *
50 REM * BY MICRO-SPARC INC *
60 REM * LINCOLN, MA. 01773 *
70 REM *****
80 PRINT CHR$ (4) "BLOAD SEARCH"
90 POKE 0,0
100 TEXT : HOME
110 PRINT CHR$ (4) "MAXFILES 1":D$ = CHR$ (4)
120 POKE 1013,76: POKE 1014,2: POKE 1015,3
130 DIM T$(13),TV$(12),P$(12),TX$(12),TW$(12)
140 SPEED= 255: SX = 6: DX = 1: FIN = 0: SP = 25: GOTO 490
150 PRINT D$ "OPEN "; FILE$, "L": RL
160 PRINT D$ "WRITE "; FILE$, "R": RX: RETURN
170 HTAB 5: PRINT "PRESS "; POKE 50,63: PRINT "RETU
RN": NORMAL : PRINT " TO QUIT ": POKE 35,22: RETURN
: REM POKE 50,63 IS INVERSE
180 VTAB 3: HTAB (20 - LEN (AA$) / 2): POKE 50,63: PRINT
AA$: NORMAL : RETURN
190 POKE 0,2
200 PRINT D$ "OPEN "; FILE$, "L": RL
210 PRINT D$ "READ "; FILE$, "R": RX: INPUT REC: IF REC = 0 THEN
220 240
220 IF REC = NR AND Y = 2 THEN POKE 0,0: FILE$ = "": GOTO
240
230 FOR R = 1 TO REC: PRINT D$ "READ "; FILE$, "R": R: INPUT
RD$(R): NEXT R
240 RETURN
250 VTAB 7: CALL - 958: PRINT
260 VTAB 7: FOR D = 1 TO NF
270 HTAB (1 + (D < 10)): PRINT D; ". "; T$(D); " ";
280 HTAB 16: POKE 50,63: PRINT " ": NORMAL
: PRINT " ":
290 NEXT D: CALL - 868: VTAB 7
300 IF BN = 1 THEN RETURN
310 FOR D = 1 TO NF
320 HTAB (1 + (D < 10)): PRINT D; ". "; T$(D); " ": HTAB 16
: POKE 50,63: PRINT P$(D): NORMAL
330 NEXT D: RETURN
340 VTAB 23: FOR K = 1 TO 38: PRINT "#": NEXT K: PRINT :
VTAB 1: FOR K = 1 TO 38: PRINT "#": NEXT K: PRINT
350 FOR K = 1 TO 22: PRINT "#": HTAB 38: PRINT "#": NEXT
K: RETURN
360 HOME : VTAB 3: PRINT D$ "CATALOG": PRINT : POKE - 163
68,0: POKE 50,63: PRINT "ENTER FILE NAME": G$: NORMAL
: INPUT " ": FILE$: IF LEN (FILE$) = 0 THEN GOTO 48
0
370 HOME : VTAB 12: HTAB (13 - LEN (FILE$) / 2): PRINT "
LOADING ": POKE 50,63: PRINT FILE$: NORMAL : PRINT
" FILES"
380 PRINT D$ "OPEN "; FILE$ + BC$
390 PRINT D$ "READ "; FILE$ + BC$
400 INPUT NF: INPUT RL: INPUT NR
410 FOR K = 1 TO NF: INPUT T$(K): INPUT TV$(K): NEXT K
420 PRINT D$ "CLOSE "; FILE$ + BC$
430 DIM RD$(NR),SRT$(NR),SO$(NR),C$(INT (NR / 4))
440 RETURN
450 PRINT D$ "OPEN "; FILE$, "L": RL
460 PRINT D$ "READ "; FILE$, "R"

```

```

470 RETURN
480 POKE 34,0: POKE 35,24: HOME
490 REM *** MENU ***
500 V1 = 7: B$ = " "
510 BC$ = CHR$ (2): D$ = CHR$ (4): G$ = CHR$ (7): H$ = CHR$
(B): V1$ = CHR$ (9): V2$ = CHR$ (93)
520 GOSUB 340
530 AA$ = "THE A. R. C. SYSTEM": GOSUB 180: RESTORE
540 FOR K = 1 TO V1: READ M$(K): NEXT K
550 DATA "INITIALIZE NEW FILE", "ENTER FILE DATA", "REVIEW/
EDIT DATA", "FIND/DIAL DATA"
560 DATA "SORT/PRINT DATA", "PURGE/PACK DATA", "MERGE/SORT
FILES"
570 FOR K = 1 TO V1: VTAB (2 * K + 4): HTAB 8: PRINT V1$;
K; V2$: " "; M$(K): NEXT K
580 VTAB 21: HTAB 5: POKE 50,63: PRINT FILE$: NORMAL : POKE
- 16368,0
590 ON PEEK (8) GOTO 600: VTAB 21: HTAB 5: PRINT "<ESC>
TO LOAD"
600 VTAB 21: HTAB 23: PRINT "CHOICE : "; G$: V1$: " "; V2$: H$
: H$: NORMAL : GET Y$: PRINT Y$: Y = VAL (Y$)
610 IF Y = 1 OR Y = 7 THEN 660
620 IF PEEK (8) = 0 AND Y < > 1 AND Y < > 7 AND Y < >
CHR$ (27) THEN 600
630 IF PEEK (8) < > 0 AND Y > 1 AND Y < 7 THEN 660
640 IF Y$ = CHR$ (27) THEN POKE 0,1: VTAB 21: HTAB 5: PRINT
"FETCH FILE ": RUN 110
650 GOTO 600
660 VTAB 2 * Y + 4: HTAB 9: PRINT Y; V2$: " ": POKE 50,63
: PRINT M$(Y): NORMAL
670 IF PEEK (8) = 1 AND Y > 1 AND Y < 7 THEN FOR D = 1 TO
300: NEXT D: GOSUB 360
680 FOR D = 1 TO 300: NEXT D
690 NB = Y
700 ON Y GOTO 720,740,1100,1950,2930,4130,4720
710 POKE 34,0: HOME : VTAB 12: HTAB 3: PRINT "THIS FILE H
AS ONLY BEEN INITIALIZED": PRINT : HTAB 9: PRINT "AND
CONTAINS NO RECORDS": G$: HTAB 22: PRINT "—": FOR D =
1 TO 2000: NEXT D: GOTO 480
720 PRINT D$ "RUN INITIALIZE"
730 REM ** DATA ENTRY **
740 RX = 0: IF REC (8) = 1 THEN GOSUB 190
750 PRINT D$ "CLOSE "; FILE$
760 REC = REC + 1: IF REC > NR THEN HOME : VTAB 12: PRINT
"THIS FILE ALREADY CONTAINS ": NR: " RECORDS": G$: G$: VTAB
14: HTAB 9: PRINT "PLEASE OPEN A NEW FILE ": FOR D =
1 TO 2500: NEXT D
770 IF REC > NR THEN REC = REC - 1: GOTO 480
780 POKE 34,0: HOME : VTAB 3: HTAB (21 - LEN (FILE$) / 2
): POKE 50,63: PRINT FILE$: NORMAL
790 IF REC = NR + 1 THEN HOME : VTAB 12: HTAB 5: PRINT N
R: " RECORDS MAXIMUM FOR THIS FILE": G$: G$: VTAB 22: HTAB
10: PRINT "HIT ANY KEY FOR MENU ": GET A$: REC = REC -
1: GOTO 480
800 POKE 34,4: HOME : HTAB (13 + (REC < 10)): PRINT "RECO
RD # "; V1$: " ": POKE 50,63: PRINT REC: NORMAL : PRINT
" "; V2$: POKE 34,6: MOD = 0
810 VTAB 24: HTAB 9: PRINT "PRESS ": POKE 50,63: PRINT "
RETURN": NORMAL : PRINT " NOW TO QUIT": VTAB 8: HTAB
1

```


Applesoft Record Command System (ARC) (Cont.)

```

820 BN = 1: GOSUB 260: FOR I = 1 TO NF:P*(I) = "": NEXT
830 FOR IN = 1 TO NF
840 BN = 0: Y = IN
850 GOTO 1720
860 IF IN = 1 AND PY% = CHR$(13) THEN 1000
870 VTAB 23: HTAB 1: CALL - 950: NEXT IN
880 VTAB 22: CALL - 950: VTAB 22: HTAB 14: PRINT "CORREC
T (Y/N) "; G$: GET Y$: PRINT Y$
890 VTAB 21: CALL - 950
900 IF Y% = "N" THEN 1020
910 IF Y% < > "Y" THEN 880
920 VTAB 22: HTAB 1: CALL - 950: HTAB 13: POKE 50,63: PRINT
" FILING RECORD ": NORMAL: RX = REC
930 RD$(RX) = "": FOR KK = 1 TO NF:RD$(KK) = RD$(RX) + P*(
KK): NEXT KK: GOSUB 150: PRINT RD$(RX)
940 RX = 0: GOSUB 150
950 PRINT REC
960 PRINT D$ "CLOSE"
970 FOR I = 1 TO NF:P*(I) = "": NEXT
980 REC = REC + 1: GOTO 790
990 PRINT D$ "CLOSE"
1000 REC = REC - 1: GOTO 480
1010 HTAB 1: GOSUB 260
1020 VTAB (7 + NF): HTAB (1 + (NF < 10)): PRINT NF + 1: ".
": POKE 50,63: PRINT "A-B-O-R-T": NORMAL
1030 MOD = 1: GOTO 1640
1040 REM *** PRINT ***
1050 FOR K = 1 TO NF
1060 HTAB SP
1070 PRINT T$(K): HTAB (SP + 14): PRINT P*(K)
1080 NEXT K: PRINT: RETURN
1090 REM *** REVIEW/EDIT ***
1100 MOD = 0: RX = 0: IF PEEK(8) = 1 THEN GOSUB 190
1110 PRINT D$ "CLOSE": FILE$
1120 IF REC = 0 THEN 710
1130 RD$ = "1": POKE 34,0: HOME: VTAB 3: HTAB (21 - LEN
(FILE$) / 2): POKE 50,63: PRINT FILE$: POKE 34,4: NORMAL
1140 IF VAL(RD%) < 1 THEN 1280
1150 VTAB 5: HTAB 1: CALL - 868
1160 POKE 37,4: POKE 36,16: POKE 50,63: PRINT "RECORD": NORMAL
1170 IF VAL(RD%) > REC THEN RD$ = STR$(REC): FR = 1
1180 PRINT " ": POKE 50,63: PRINT RD$: NORMAL
1190 POKE 35,24: VTAB 23: CALL - 950: VTAB 9
1200 RD = INT(VAL(RD%)): IF RD < 0 THEN 1280
1210 RX = RD: IF RX < 0 THEN RX = 0
1220 FL = 0
1230 FOR K = 1 TO NF:P*(K) = MID$(RD$(RX), FL + 1, TVZ(K)
): FL = FL + TVZ(K): NEXT K
1240 GOSUB 260
1250 IF FR = 0 THEN 1270
1260 VTAB 23: HTAB 5: FLASH: PRINT " ONLY "; REC: " RECORD
S ARE ON FILE ": FOR D = 1 TO 2000: NEXT D: FR = 0: NORMAL
1270 VTAB 5: HTAB 24: POKE 50,63: PRINT RD: NORMAL: POKE
- 16368,0
1280 VTAB 21: HTAB 2: POKE 50,63: PRINT "ESC": NORMAL: PRINT
" FETCH ": POKE 50,63: PRINT "<--": NORMAL: PRINT
" LAST ": POKE 50,63: PRINT "-->": NORMAL: PRINT
" NEXT"
1290 VTAB 23: HTAB 2: POKE 50,63: PRINT "P ": NORMAL: PRINT
" PRINT ": POKE 50,63: PRINT "E ": NORMAL: PRINT
" EDIT ": POKE 50,63: PRINT "F ": NORMAL: PRINT
" FIND "; G$: " ": GET Y$: PRINT
1300 IF Y% = "F" THEN VTAB 21: HTAB 2: CALL - 950: POKE
50,63: PRINT "FIELD #": NORMAL: PRINT " TO SEARCH:
"; G$: INPUT " "; TV$: TV = VAL(TV%): IF TV < 1 OR TV
> NF THEN 1300
1310 IF Y% = "F" THEN VTAB (6 + TV): HTAB 5: FLASH: PRINT
T$(TV): NORMAL
1320 IF Y% = "F" THEN VTAB 20: CALL - 950: GOTO 2280
1330 IF Y% = CHR$(27) THEN VTAB 21: HTAB 1: CALL - 95
8: VTAB 22: HTAB 11: PRINT "RECORD TO FETCH ": G$: INPUT
" "; D$: VTAB 21: CALL - 950
1340 IF Y% = CHR$(27) AND LEN(D%) = 0 THEN 1280
1350 IF Y% = CHR$(27) THEN RD$ = D$: GOTO 1140
1360 IF Y% = CHR$(13) OR LEN(RD%) = 0 THEN 480
1370 IF Y% = CHR$(80) THEN PRINT D$ "PR#1": PRINT CHR$(
9): "B0N": GOSUB 1050: PRINT: PRINT D$ "PR#0": GOTO
1280
1380 IF Y% = CHR$(21) THEN RD = RD + 1: IF RD > REC THEN
RD = 1
1390 IF RD = 1 AND Y% < > "E" THEN VTAB 5: HTAB 24: PRINT
"
1400 IF Y% = CHR$(21) THEN VTAB 5: HTAB 24: POKE 50,63
: PRINT " ": IF RD - 1 > 9 THEN PRINT " ": IF RD -
1 > 99 THEN PRINT "
1410 IF Y% = CHR$(21) THEN NORMAL: PRINT " ": GOTO 12
10
1420 IF Y% = CHR$(8) THEN RD = RD - 1: IF RD < 1 THEN R
D = REC
1430 IF Y% = CHR$(8) THEN VTAB 5: HTAB 24: POKE 50,63:
PRINT " ": IF RD > 9 THEN PRINT " ": IF RD > 99 THEN
PRINT "
1440 IF Y% = CHR$(8) THEN NORMAL: PRINT " ": GOTO 121
0
1450 IF Y% < > "E" THEN 1280
1460 VTAB 21: HTAB 1: CALL - 950: VTAB 22: POKE 50,63: PRINT
" M ": NORMAL: PRINT " MODIFY ": POKE 50,63: PRINT
" E ": NORMAL: PRINT " ERASE ": POKE 50,63: PRINT
" RTN ": NORMAL: PRINT " ABORT "; G$: " ": GET Y$
1470 IF Y% = CHR$(13) THEN HTAB 1: VTAB 21: CALL - 95
8: GOTO 1270
1480 IF Y% = "M" THEN 1640
1490 IF Y% < > "E" THEN 1460
1500 VTAB 21: HTAB 1: CALL - 950: VTAB 22: HTAB 8: PRINT
"VERIFY ERASURE (YES/NO) "; G$: INPUT " "; Y$: IF Y% =
"YES" THEN 1530
1510 IF Y% = "NO" THEN VTAB 21: HTAB 1: CALL - 950: GOTO
1270
1520 PRINT G$: GOTO 1500
1530 FOR D = 1 TO 250: NEXT D

```

```

1540 VTAB 22: HTAB 1: CALL - 868: HTAB 11: FLASH: PRINT
" ERASING RECORD "; RX: " ": NORMAL
1550 IF RD < > REC THEN 1590
1560 RX = 0: REC = REC - 1: GOSUB 160: PRINT REC
1570 PRINT D$ "CLOSE": FILE$
1580 FOR D = 1 TO 500: VTAB 22: HTAB 1: CALL - 950: RD$ =
STR$(REC): GOTO 1140
1590 FOR K = 2 TO NF:P*(K) = "": FOR J = 1 TO TVZ(K): P*(K
) = P*(K) + " ": NEXT J: K: P*(1) = " ": FOR J = 2 TO T
VZ(1): P*(1) = P*(1) + " ": NEXT J
1600 RD$(RD) = "": FOR K = 1 TO NF:RD$(RD) = RD$(RD) + P*(
K): NEXT K
1610 RX = RD: GOSUB 150: PRINT RD$(RX)
1620 PRINT D$ "CLOSE": FILE$
1630 FOR D = 1 TO 500: VTAB 22: HTAB 1: CALL - 950: RD$ =
STR$(RD): GOTO 1140
1640 REM ** MODIFY/ENTER **
1650 LC = 0: IN = 0
1660 VTAB 22: HTAB 1: CALL - 868: PRINT "ENTER ": POKE
50,63: PRINT "FIELD #": NORMAL: PRINT " TO CHANGE:
": G$
1670 INPUT " "; Y$: Y = VAL(Y%): IF LEN(Y%) = 0 AND MOD
= 0 THEN 1890
1680 IF LEN(Y%) = 0 AND MOD = 1 THEN 920
1690 IF INT(Y) < 1 OR INT(Y) > NF + 1 THEN 1660
1700 IF MOD = 0 AND INT(Y) > NF THEN 1660
1710 IF INT(Y) = NF + 1 THEN FOR I = 1 TO NF:P*(I) = "
": NEXT: GOTO 800
1720 HT = 17
1730 NORMAL: P% = "": PY% = "": HT = HT - 1: VTAB 22: CALL
- 868: HTAB 14: PRINT "ENTER NEW DATA": G$
1740 VTAB (6 + Y): HTAB (HT): POKE 50,63: GET P$: IF P% =
CHR$(8) AND PEEK(36) = 15 THEN POKE 36,15: GOTO
1720
1750 IF P% = CHR$(8) AND LEN(PY%) = 1 THEN 1730
1760 IF P% = CHR$(8) THEN HT = HT - 1: PY% = LEFT$(PY%
, LEN(PY%) - 1): GOTO 1740
1770 IF LEN(PY%) = TVZ(Y) AND P% < > CHR$(13) THEN PRINT
G$: GOTO 1740
1780 PRINT P$: NORMAL: PY% = PY% + P%
1790 IF IN = 1 AND PY% = CHR$(13) THEN REC = REC - 1: GOTO
480
1800 IF PY% = CHR$(13) AND LEN(PY%) < 2 THEN P*(Y) =
" ": GOTO 1830
1810 IF P% = CHR$(13) THEN P*(Y) = LEFT$(PY%, LEN(PY
%) - 1): GOTO 1830
1820 HT = HT + 1: GOTO 1740
1830 IF LEN(P*(Y)) < TVZ(Y) THEN FOR J = 1 TO TVZ(Y) -
LEN(P*(Y)): P*(Y) = P*(Y) + " ": NEXT J: GOTO 1850
1840 IF LEN(P*(Y)) = TVZ(Y) THEN 1850
1850 GOSUB 260: RD$(RX) = "": FOR K = 1 TO NF:RD$(RX) = RD
$(RX) + P*(K): NEXT K: LC = 1
1860 IF MOD = 1 THEN 1020
1870 IF NB = 2 THEN 860
1880 GOTO 1660
1890 IF LC = 0 AND Y% = " " THEN 1930
1900 VTAB 22: HTAB 1: CALL - 950: HTAB 9: POKE 50,63: PRINT
"SAVING MODIFIED RECORD": NORMAL
1910 RX = RD: GOSUB 150: PRINT RD$(RX)
1920 PRINT D$ "CLOSE": FILE$
1930 HTAB 1: VTAB 20: CALL - 950: HTAB 5: HTAB 17: POKE
50,63: PRINT "RECORD": NORMAL: GOTO 1270
1940 REM ** KISS' FIND **
1950 A1$ = CHR$(91): A2$ = CHR$(93): Y% = "NF": MODEN = 1
1960 RX = 0: IF PEEK(8) = 1 THEN GOSUB 190
1970 PRINT D$ "CLOSE": FILE$
1980 IF REC = 0 THEN 710
1990 POKE 34,0: HOME: VTAB 3: HTAB (21 - LEN(FILE%) /
2): POKE 50,63: PRINT FILE$: POKE 34,4: NORMAL
2000 HOME: PRINT: PRINT " THE FOLLOWING FIELDS ARE AVAI
LABLE ": HTAB 9: PRINT "FOR THE FIND FUNCTION ": PRINT
2010 T$(NF + 1) = H% + H% + H% + B%
2020 FOR K = 1 TO NF STEP 2
2030 IF K > 9 THEN 2070
2040 IF K = 9 THEN PRINT " "; K: ". "; T$(K): TAB(21): K +
1: ". "; T$(K + 1): GOTO 2080
2050 PRINT " "; K: ". "; T$(K): TAB(22): K + 1: ". "; T$(K +
1)
2060 GOTO 2080
2070 PRINT " "; K: ". "; T$(K): TAB(21): K + 1: ". "; T$(K +
1)
2080 NEXT K: K = NF
2090 IF K / 2 < > INT(K / 2) THEN VTAB PEEK(37): HTAB
21: CALL - 950
2100 POKE - 16368,0
2110 VTAB 21: HTAB 10: GOSUB 170: PRINT: VTAB 17: CALL -
868: HTAB 2: PRINT "ENTER ": POKE 50,63: PRINT " FIE
LD # ": NORMAL: PRINT " TO BE SEARCHED ": G$: INPUT
" "; TV$
2120 IF LEN(TV%) = 0 THEN 480
2130 TV = INT(VAL(TV%)): IF TV < 1 OR TV > NF THEN 211
0
2140 IF LEN(TV%) > 1 AND TV < 10 ^ (LEN(TV%) - 1) THEN
2110
2150 POKE 34,0: HOME: VTAB 1: HTAB 11: POKE 50,63: PRINT
" F I N D R U L E S ": NORMAL
2160 PRINT
2170 HTAB 4: PRINT "THE FIND FUNCTION WILL SEARCH FOR"
2180 HTAB 3: PRINT "AND DISPLAY ALL RECORD NUMBERS WHOSE"
: PRINT
2190 HTAB (21 - LEN(T$(TV)) / 2): POKE 50,63: PRINT T$(
TV): NORMAL: PRINT
2200 HTAB 3: PRINT "FIELD 'MATCHES' YOUR KEYWORD...": A1$:
" K "; A2$
2210 PRINT: HTAB 5: POKE 50,63: PRINT "KEYWORD": HTAB 2
4: PRINT "FIELD MATCH": NORMAL
2220 HTAB 3: PRINT "-----": HTAB 20: PRINT "-----"
2230 HTAB 5: PRINT "A1$: " K "; A2$: HTAB 21: PRINT "BE
GINS WITH "; A1$: " K "; A2$
2240 HTAB 6: PRINT A1$: " K "; A2$: HTAB 21: PRINT "CONTAI
NS "; A1$: " K "; A2$: PRINT: PRINT
2250 PRINT: POKE 50,63: PRINT "NOTE": NORMAL: PRINT " ":
A1$: " K "; A2$: " CAN BE ANY COMBINATION OF"
2260 HTAB 7: PRINT "LETTERS, NUMBERS, AND SYMBOLS.": PRINT
2270 POKE 35,24

```

continued on next page

Applesoft Record Command System (ARC) (Cont.)

```

2280 L = 0: PRINT : HTAB 2: POKE 50,63: PRINT "ENTER KEYWORD"; NORMAL: PRINT " "; G%: INPUT " "; KEY%
2290 IF KEY% = "" THEN 1950
2300 IF LEFT$(KEY%,1) = "*" THEN L = 1
2310 IF LEN(KEY%) > TV%(TV) + L THEN VTAB 21: HTAB 16: CALL - 950: HTAB 18: POKE 50,63: PRINT " KEYWORD > FIELD "; G%,G%: FOR D = 1 TO 1000: NEXT D: HTAB 16: CALL - 860: VTAB 20: NORMAL: GOTO 2280
2320 IF KEY% = CHR$(42) THEN VTAB 20: HTAB 18: CALL - 950: GOTO 2280
2330 A% = KEY%
2340 FOR D = 1 TO 100: NEXT D
2350 IF Y% = "F" THEN 2390
2360 HOME: VTAB 3: HTAB (21 - LEN(FILE%)/2): POKE 50,63: PRINT FILE%: NORMAL: VTAB 6: POKE 34,4: HOME
2370 HTAB (17 - LEN(A%)/2): PRINT A%: " K "; A2%: " = "
2380 VTAB 22: HTAB (21 - LEN(T%(TV))/2): POKE 50,63: PRINT T%(TV): NORMAL: VTAB 7: HTAB 1
2390 IF LEFT$(KEY%,1) < > CHR$(42) THEN DUM = 2: GOTO 2410
2400 KEY% = RIGHT$(KEY%, LEN(KEY%) - 1): DUM = 1
2410 CC = 0
2420 IF TV = 1 THEN FL = 1: GOTO 2440
2430 FL = 0: FOR JK = 1 TO TV - 1: FL = FL + TV%(JK): NEXT JK: FL = FL + 1
2440 LD = 1: IF Y% = "F" THEN LD = RD + 1: IF RD + 1 > REC THEN LD = 1
2450 FOR K = LD TO REC
2460 IF Y% = "F" THEN 2480
2470 VTAB 5: HTAB 7: POKE 50,63: PRINT K: NORMAL
2480 IF DUM = 2 THEN 2530
2490 DUM% = LEFT$(MID$(RD%(K),FL,TV%(TV)), LEN(KEY%))

2500 & DUM%,KEY%
2510 IF PEEK(26) = 0 THEN 2610
2520 GOTO 2570
2530 DUM% = MID$(RD%(K),FL,TV%(TV))
2540 & DUM%,KEY%
2550 CK = 0
2560 IF PEEK(26) = 0 THEN 2610
2570 IF CC > 11 THEN CC = CC + 1: CK = 1: GOTO 2610
2580 IF Y% = "F" THEN Y% = CHR$(21): RD = K - 1: GOTO 1380
2590 CC = CC + 1: CK = 1: ALPHA = K
2600 VTAB 6 + CC: HTAB 3: POKE 50,63: PRINT "RECORD "; HTAB (10 + (K < 100) + (K < 10)): PRINT K: NORMAL: PRINT " ..... "; MID$(RD%(K),FL,TV%(TV))
2610 NEXT K
2620 IF Y% = "F" THEN VTAB (6 + TV): HTAB 5: PRINT T%(TV): GOTO 1280
2630 IF CC = 0 THEN VTAB 13: HTAB 7: PRINT "NO SUCCESSFUL CORRELATION"
2640 IF CC > 13 THEN CC = CC - 12: VTAB 20: HTAB 3: PRINT "+ "CC;" ADDITIONAL RECORDS"
2650 VTAB 24: HTAB 7: PRINT "HIT "; POKE 50,63: PRINT " F "; NORMAL: PRINT " FOR ANOTHER SEARCH "; G%
2660 IF CC = 1 THEN VTAB 22: HTAB 9: PRINT "HIT "; FLASH: PRINT " D "; NORMAL: PRINT " TO DIAL NUMBER": VTAB 22: HTAB 14
2670 GET Z%: IF Z% = CHR$(70) THEN GOTO 1990
2680 IF Z% = CHR$(68) AND CC = 1 THEN 2720
2690 IF Z% = CHR$(13) THEN 480
2700 GOTO 2650
2710 REM -> D.C. HAYES MODEM <-
2720 HOME: VTAB 10: DIAL% = CHR$(17)
2730 FOR I = 1 TO 12: IF LEFT$(T%(I),5) = "PHONE" OR LEFT$(T%(I),4) = "TELE" THEN 2750
2740 NEXT I: GOTO 2760
2750 IF MODEM = 1 THEN 2780
2760 VTAB 14: HTAB 6: POKE 50,63: PRINT " NO PHONE NUMBER S IN RECORDS ": NORMAL
2770 HTAB 6: POKE 50,63: PRINT " OR NO MODEM CONNECTIO N "; G%,G%: NORMAL: FOR D = 1 TO 2500: NEXT D: GOTO 1980
2780 PRINT
2790 FL = 0: FOR IK = 1 TO I - 1: FL = FL + TV%(IK): NEXT I: K: FL = FL + 1
2800 NUMBER% = MID$(RD%(ALPHA),FL,TV%(I))
2810 PRINT D%PR#2"
2820 NUMBER% = NUMBER% + CHR$(10)
2830 VTAB 11: PRINT DIAL%: NUMBER%
2840 FOR D = 1 TO 2000: NEXT D
2850 VTAB 17: HTAB 8: FLASH: PRINT "

2860 HTAB 8: FLASH: PRINT " PLEASE PICK UP THE "
2870 HTAB 8: FLASH: PRINT "
2880 HTAB 8: FLASH: PRINT " THE TELEPHONE-RECEIVER "
2890 HTAB 8: FLASH: PRINT " "; NORMAL
: FOR D = 1 TO 5000: NEXT D
2900 ALPHA = 0: VTAB 13: HTAB 1: CALL - 950: VTAB 13: HTAB 10: G% = CHR$(26): PRINT G%
2910 PRINT D%PR#0": FOR D = 1 TO 1500: NEXT D: POKE 34,0: HOME: FOR D = 1 TO 500: NEXT D: GOTO 1990
2920 REM ** SORT/PRINT **
2930 LABEL = 0
2940 RX = 0: IF PEEK(8) = 1 THEN GOSUB 190
2950 PRINT D%CLOSE "; FILE%
2960 IF REC = 0 THEN 710
2970 REM USER SETS TOP OF FORM
2980 B1% = "LABEL": INDENT = 25
2990 POKE 34,0: HOME: VTAB 3: HTAB (21 - LEN(FILE%)/2): POKE 50,63: PRINT FILE%: POKE 34,4: NORMAL
3000 T%(NF + 1) = H% + H% + H% + B%: FOR K = 1 TO NF: TX%(K) = CHR$(13): NEXT K
3010 HOME: L = 0: PRINT: PRINT " THE FOLLOWING FIELDS ARE AVAILABLE ": HTAB 12: PRINT "FOR THE "; B1%: ";: PRINT

```

```

3020 FOR K = 1 TO NF STEP 2
3030 PRINT " "; TAB(4 - (K > 9)): K1%. " "; T%(K): TAB(22 - ((K + 1) > 9)): K + 1%. " "; T%(K + 1)
3040 NEXT K: NF
3050 IF K / 2 < > INT(K / 2) THEN VTAB PEEK(37): HTAB 21: CALL - 950
3060 L = L + 1
3070 IF L > NF THEN 3150
3080 VTAB 21: HTAB 10: GOSUB 170: PRINT: VTAB 17: CALL - 860: PRINT "ENTER FIELD # FOR "; POKE 50,63: PRINT " LINE "; L1: G%: NORMAL: INPUT " "; T%
3090 IF L = 1 AND LEN(T%) = 0 THEN 480
3100 IF LEN(T%) = 0 THEN 3150
3110 IF TV% = CHR$(21) OR TV% = CHR$(32) THEN 480
3120 IF TV% = "Q" THEN L = NF + 1: TV = NF: FOR I = 1 TO NF: TW%(I) = I: NEXT I: GOTO 3150
3130 TV = INT(VAL(T%)): IF TV < 1 OR TV > NF THEN 3080
3140 TW%(L) = TV: GOTO 3060
3150 HOME: LINES = L - 1: PRINT: HTAB 8: PRINT "YOUR "; B1%: " WILL CONSIST OF ": HTAB 7: PRINT "-----"
3160 FOR K = 1 TO LINES: HTAB (14 - (K > 9)): PRINT K1%. " "; T%(TW%(K)): NEXT K
3170 POKE 34,20: VTAB 22: HTAB 10: PRINT "IS THIS CORRECT (Y/N) "; G%: GET Y%: IF Y% = "N" THEN POKE 34,4: GOTO 3010
3180 IF Y% = CHR$(21) OR Y% = CHR$(32) THEN 480
3190 IF Y% < > "Y" THEN 3170
3200 LC = 0
3210 VTAB 21: CALL - 860: VTAB 22: HTAB 6: PRINT "MERGE TWO OR MORE LINES (Y/N) "; G%: GET Y%: IF Y% = "N" THEN 3420
3220 IF Y% < > "Y" THEN 3210
3230 FOR K = 1 TO NF: TX%(K) = CHR$(13): NEXT K
3240 GOTO 3260
3250 VTAB 21: HTAB 16: CALL - 950: HTAB 16: PRINT "TO FIELD # "; VTAB 23: HTAB 10: GOSUB 170: GOTO 3300
3260 HTAB 1: VTAB 21: CALL - 950: POKE 50,63: PRINT "JOIN "; NORMAL: PRINT " FIELD #"; G%: INPUT " "; LX%: IF LEN(LX%) = 0 THEN 3420
3270 VTAB 21: HTAB 16: CALL - 950: LX = VAL(LX%)
3280 IF LX < 1 OR LX > LINES - 1 THEN POKE 50,63: HTAB 14: PRINT "INVALID": NORMAL: PRINT G%,G%: FOR D = 1 TO 500: NEXT D: VTAB 21: HTAB 1: GOTO 3260
3290 LC = LC - 1: GOTO 3250
3300 VTAB 21: HTAB 27: PRINT LX + 1: "
3310 VTAB 21: HTAB 32: PRINT "ENTER "; FLASH: PRINT " ? "; G%: VTAB 21: HTAB 39: NORMAL: TX%(LX) = "
3320 GET T1%: IF T1% = CHR$(13) THEN TX%(LX) = CHR$(13): GOTO 3420
3330 IF T1% < > " " THEN 3350
3340 TX%(LX) = TX%(LX) + T1%: GOTO 3360
3350 TX%(LX) = TX%(LX) + " " + T1% + " "
3360 Z% = T%(TW%(LX)) + T1% + T%(TW%(LX + 1))
3370 ZZ = LEN(Z%) + 2: ZZ = (40 - ZZ) / 2 - 1
3380 VTAB 23: HTAB 1: CALL - 950
3390 HTAB 22: POKE 50,63: PRINT T%(TW%(LX)): NORMAL: PRINT " "; POKE 50,63: PRINT TX%(LX): NORMAL: PRINT " "; FOR D = 1 TO 500: NEXT D
3400 GOTO 3260
3410
3420 POKE 34,4: POKE 35,23: HOME: PRINT: HTAB 6: PRINT "THE "; B1%: " WILL LOOK LIKE THIS: "
3430 HTAB 6: PRINT "-----"
3440 FOR K = 1 TO LINES: PRINT T%(TW%(K)): TX%(K): NEXT K
3450 VTAB 22: HTAB 14: PRINT "CORRECT (Y/N) "; G%: GET Y%: IF Y% = "N" THEN 3010
3460 IF Y% < > "Y" THEN 3450
3470 HOME: VTAB 12: HTAB 8: PRINT "WANT SORTED "; B1%: " (Y/N) ? "; G%: GET Y1%: PRINT Y1%: IF Y1% < > "Y" THEN 3740
3480 HOME: PRINT: PRINT "THE FOLLOWING LINES ARE AVAILABLE FOR"
3490 PRINT "THE SORT FIELD: "; PRINT
3500 FOR K = 1 TO NF STEP 2: HTAB (4 - (K > 9)): PRINT K1%. " "; T%(K): TAB(22 - (K > 8)): K + 1%. " "; T%(K + 1): NEXT K: IF NF / 2 < > INT(NF / 2) THEN VTAB PEEK(37): HTAB 21: CALL - 950: PRINT
3510 PRINT: PRINT "SORT ON WHICH FIELD (BY NUMBER) "; G%: INPUT " "; SF%: SF = VAL(SF%): IF SF < 1 OR SF > NF THEN 3480
3520 POKE 34,0: POKE 35,24: HOME
3530 VTAB 10: HTAB (13 - LEN(T%(SF))/2): PRINT "###" : POKE 50,63: PRINT "ON "; T%(SF): NORMAL: PRINT "###"
3540 PRINT: PRINT: POKE 50,63: PRINT "ASCENDING": NORMAL: PRINT " OR "; POKE 50,63: PRINT "DESCENDING": NORMAL: PRINT " SORT (A/D) "; G%: GET S%: PRINT S%: S1 = (S% = "A")
3550 IF S% < > "A" AND S% < > "D" THEN VTAB (PEEK(37) - 1): GOTO 3540
3560 S% = "ASCENDING": IF S% = "D" THEN S% = "DESCENDING"
3570 VTAB 12: CALL - 950: VTAB 17: HTAB 14: PRINT "S O R T I N G"
3580 VTAB 7: HTAB 16: POKE 50,63: PRINT S%: NORMAL
3590 FOR K = 1 TO REC
3600 SOX(K) = K
3610 IF SF = 1 THEN FL = 1: GOTO 3630
3620 FL = 0: FOR I = 1 TO SF - 1: FL = FL + TV%(I): NEXT I: FL = FL + 1
3630 SRT%(K) = MID$(RD%(K),FL,TV%(SF))
3640 NEXT K
3650 M = 1
3660 M = 3 + M + 1: IF M < REC THEN 3660
3670 M = (M - 1) / 3: IF M < 1 THEN 3720
3680 FOR J = M + 1 TO REC: LL = J - M: S% = SRT%(J): S = 90: Z(J)
3690 VTAB 20: HTAB 19: POKE 50,63: PRINT " "; HTAB (19 + (J < 100) + (J < 10)): PRINT J: NORMAL
3700 IF S1 = (SRT%(LL) > S%) THEN SRT%(LL + M) = SRT%(LL): SOX(LL + M) = SOX(LL): LL = LL - M: IF LL > 0 THEN 3700
3710 SRT%(LL + M) = S%: SOX(LL + M) = S: NEXT J: GOTO 3670
3720 REM SHELL SORT LIST DONE
3730 FOR D = 1 TO 1000: NEXT D

```



```

3740 IF LABEL > 0 THEN 3700
3750 HOME : VTAB 9: PRINT "HOW MANY "; POKE 50,63: PRINT
"CARTRIDGE RETURNS"; NORMAL : PRINT " FROM END OF: PRINT
: HTAB 11: PRINT "ONE ";B1%; TO NEXT ? ";G%; INPUT
";CR
3760 VTAB 15: HTAB 13: POKE 50,63: PRINT "V"; NORMAL : PRINT
"IEW OR "; POKE 50,63: PRINT "P"; NORMAL : PRINT "R
INT ? ";G%; GET VP%; IF VP% < > "V" AND VP% < > "P
" THEN 3760
3770 PRINT VP%; FOR D = 1 TO 500: NEXT D:XX = 6
3780 GOTO 3810
3790 HOME
3800 FOR K1 = 1 TO REC:RX = 90%(K1): GOTO 3960
3810 POKE 35,24: HOME
3820 VTAB 23: POKE 50,63: PRINT "PADDLE #1"; NORMAL : PRINT
" CONTROLS SCROLL/PRINT SPEED !"; POKE 35,21: VTAB 6:
FOR D = 1 TO 750: NEXT D
3830 IF VP% = "P" THEN PRINT D*"PR#1": PRINT CHR% (9); "
00N";
3840 IF Y1% = "Y" THEN 3790
3850 RX = RX + 1: IF RX < 1 THEN RX = 1
3860 IF RX < = REC THEN 3960
3870 PRINT D*"PR#0": PRINT : SPEED= 255: HTAB 15: POKE 50
,63: PRINT " END OF FILE ": NORMAL : FOR D = 1 TO 100
0: NEXT D
3880 IF VP% = "P" THEN PRINT D*"PR#1": PRINT CHR% (12):
PRINT D*"PR#0"
3890 POKE 35,24: HOME : VTAB 9
3900 PRINT : PRINT : PRINT : POKE - 16368,0
3910 HTAB 5: PRINT "PRESS "; POKE 50,63: PRINT "RETURN";
: NORMAL : PRINT " TO RE-FORMAT COPY": PRINT
3920 HTAB 7: PRINT "PRESS "; POKE 50,63: PRINT " C "; NORMAL
: PRINT " FOR ANOTHER COPY ";G%; GET Y%; PRINT
3930 IF Y% = CHR% (67) THEN HOME : LABEL = 1:RX = XR: GOTO
3750
3940 IF Y% = CHR% (13) THEN LABEL = 0:RX = XR: GOTO 2990
3950 VTAB 14: GOTO 3920
3960 IF VP% = "P" THEN HTAB (INDENT): IF (XX + LINES + L
C) > 60 THEN PRINT CHR% (12):XX = 6: HTAB (INDENT)
3970 FL = 0: FOR K = 1 TO NF:P%(K) = MID% (RD%(RX),FL + 1
,TVZ(K)):FL = FL + TVZ(K): NEXT
3980 IF VP% = "P" THEN HTAB (INDENT)
3990 FOR K = 1 TO LINES
4000 L = LEN (P%(TVZ(K))): IF L < 2 THEN L = 1: GOTO 4040
4010 FOR J = 1 TO L - 1
4020 IF MID% (P%(TVZ(K)),L,1) < > " " THEN 4040
4030 L = L - 1: NEXT
4040 PRINT LEFT% (P%(TVZ(K)),L):TX%(K): IF TX%(K) = CHR%
(13) THEN XX = XX + 1: IF VP% = "P" THEN HTAB (INDEN
T)
4050 SPEED= PDL (1): NEXT K
4060 ON CR + 1 GOTO 4100,4090
4070 FOR D = 1 TO CR: PRINT :XX = XX + 1: NEXT D
4080 GOTO 4100
4090 PRINT :XX = XX + 1
4100 IF Y1% < > "Y" THEN 3850
4110 NEXT K1: PRINT D*"PR#0": GOTO 3870
4120 REM *** PURGE/PACK ***
4130 B1% = CHR% (91):B2% = CHR% (93)
4140 RX = 0: IF PEEK (8) = 1 THEN GOSUB 190
4150 PRINT D*"CLOSE ";FILE%
4160 IF REC = 0 THEN 710
4170 POKE 34,0: HOME : VTAB 2: HTAB (21 - LEN (FILE%)) /
2: POKE 50,63: PRINT FILE%: POKE 34,4: NORMAL
4180 NP = 0:L = 1
4190 VTAB 5: HTAB 1: POKE 50,63: PRINT "ALL"; NORMAL : PRINT
" ERASED/BLANK RECORDS WILL BE PURGED": PRINT
4200 PRINT "FROM YOUR FILE AND THE REMAINING RECORDS"
4210 PRINT "WILL BE REPACKED. FOR EXAMPLE "; PRINT : PRINT
4220 HTAB 4: PRINT "1 2 3 4 5 6 7 8 9"
4230 HTAB 3: PRINT "-----"
4240 PRINT : HTAB 3: PRINT B1%; "A";B2%; " ";B1%; "B";B2%;
";B1%; " ";B2%; " ";B1%; "D";B2%; " ";B1%; "E";B2%; " ";B1%
";B2%; " ";B1%; "G";B2%; " ";B1%; "H";B2%; " ";B1%; "I";
B2%
4250 PRINT : HTAB 3: PRINT B1%; "A";B2%; " ";B1%; "B";B2%;
";B1%; "I";B2%; " ";B1%; "D";B2%; " ";B1%; "E";B2%; " ";B1%
";H";B2%; " ";B1%; "G";B2%
4260 PRINT : HTAB 3: PRINT "-----"
4270 VTAB 21: HTAB 10: PRINT "HIT "; POKE 50,63: PRINT "
P "; NORMAL : PRINT " TO PURGE FILES"
4280 VTAB 23: HTAB 13: PRINT "HIT "; POKE 50,63: PRINT "
RTN"; NORMAL : PRINT " FOR MENU ";G%; GET A%
4290 IF A% = "P" THEN 4320
4300 IF A% = CHR% (13) THEN 480
4310 GOTO 4280
4320 FOR D = 1 TO 100: NEXT D
4330 VTAB 12: HTAB 1: CALL - 868
4340 FOR D = 15 TO 17: VTAB (D): HTAB 1: CALL - 868: NEXT
D
4350 VTAB 16: HTAB 11: PRINT "## "; POKE 50,63: PRINT "P
URGING FILES"; NORMAL : PRINT " ##"
4360 VTAB 21: HTAB 1: CALL - 958
4370 FOR D = 1 TO 1500: NEXT D
4380 CC = 0
4390 FOR K = 1 TO REC
4400 IF LEFT% (RD%(K),1) = "*" THEN CC = CC + 1:C%(CC) =
K
4410 NEXT K
4420 IF L > 1 THEN 4440
4430 NP = CC
4440 IF CC = 0 THEN 4500
4450 FOR K = 1 TO CC
4460 RD%(C%(K)) = RD%(REC)
4470 REC = REC - 1
4480 NEXT K
4490 L = L + 1: GOTO 4380
4500 HOME : VTAB 10: HTAB 14: PRINT "PURGE COMPLETED"
4510 VTAB 12: HTAB 13: POKE 50,63: PRINT NP; NORMAL : PRINT
" RECORDS DELETED"
4520 GOTO 4670

```

```

4530 VTAB 14: HTAB 11: PRINT "HIT "; POKE 50,63: PRINT "
RTN"; NORMAL : PRINT " TO SAVE FILES ";G%; GET A%:
PRINT " "
4540 IF A% < > CHR% (13) THEN 4530
4550 PRINT D*"OPEN ";FILE%,L";RL
4560 PRINT D*"DELETE ";FILE%
4570 PRINT D*"OPEN ";FILE%,L";RL
4580 FOR R = 1 TO REC
4590 PRINT D*"WRITE ";FILE%,R";R
4600 PRINT RD%(R)
4610 NEXT
4620 PRINT D*"WRITE ";FILE%,R";0
4630 PRINT REC
4640 PRINT D*"CLOSE"
4650 FOR D = 1 TO 1000: NEXT D
4660 GOTO 480
4670 VTAB 18: HTAB 9: PRINT "'HOUSECLEANING' - STANDBY ";
G%;F = FRE (0): FOR D = 1 TO 500: NEXT D: VTAB 18: HTAB
1: CALL - 868: HTAB 12: PRINT "FREE MEMORY = ";F
4680 IF NP < > 0 THEN 4530
4690 VTAB 22: HTAB 13: PRINT "HIT "; POKE 50,63: PRINT "
RTN"; NORMAL : PRINT " FOR MENU ";G%; GET A%
4700 IF A% < > CHR% (13) THEN 4690
4710 GOTO 480
4720 PRINT D*"RUN MERGE FILES"
4730 END

```

```

10 REM *****
20 REM # MERGE FILES #
30 REM # BY CHUCK KISS #
40 REM # COPYRIGHT (C) 1982 #
50 REM # BY MICRO-SPARC INC #
60 REM # LINCOLN, MA. 01773 #
70 REM *****
80 PRINT CHR% (4); "NOMON C,I,0"
90 PRINT CHR% (4); "MAXFILES 1"
100 D% = CHR% (4)
110 DIM T%(13),TVZ(32),P%(12),TX%(24),TW%(24)
120 POKE 768,6: POKE 769,1: SPEED= 255: TEXT : HOME :FIN =
0: GOTO 250
130 HTAB 5: PRINT " PRESS "; INVERSE : PRINT "RETURN
"; NORMAL : PRINT " TO QUIT "; POKE 35,22: RETURN
140 VTAB 3: HTAB (20 - LEN (AA%)) / 2: INVERSE : PRINT A
A%: NORMAL : POKE 34,4: RETURN
150 PRINT D*"OPEN ";FILE%,L";RL
160 PRINT D*"READ ";FILE%,R";RX: INPUT REC: IF REC = 0 THEN
PRINT D*"CLOSE": GOTO 460
170 FOR R = 1 TO REC: PRINT D*"READ ";FILE%,R";R: INPUT
RD%(1J + R): NEXT R: PRINT D*"CLOSE": RETURN
180 VTAB 15: HTAB 2: FOR K = 1 TO 36: PRINT "#"; NEXT K:
PRINT : VTAB 1: HTAB 2: FOR K = 1 TO 36: PRINT "#";
NEXT K: PRINT
190 FOR K = 2 TO 14: PRINT "#"; HTAB 38: PRINT "#": NEXT
K: RETURN
200 PRINT D*"OPEN ";FILE% + BC%
210 PRINT D*"READ ";FILE% + BC%
220 INPUT NF: INPUT RL: INPUT NR
230 FOR K = 1 TO NF: INPUT T%(K): INPUT TVZ(K): NEXT K
240 PRINT D*"CLOSE": RETURN
250 REM *** MENU ***
260 REM
270 V1 = 7:B% = " "
280 BC% = CHR% (2):G% = CHR% (7):H% = CHR% (8):V1% = CHR%
(91):V2% = CHR% (93)
290 TEXT : HOME : GOSUB 180
300 AA% = " M E R G E F I L E S ": GOSUB 140: RESTORE
310 VTAB 5: HTAB 3: PRINT "THIS SUBROUTINE WILL MERGE TWO
(2)"
320 HTAB 4: PRINT "FILES AND ALLOW THE USER TO SORT"
330 HTAB 3: PRINT "& PRINT DATA FROM THE NEWLY MERGED"
340 HTAB 4: PRINT "FILE. NOTE THAT THIS MERGED FILE"
350 HTAB 3: PRINT "IS NOT RESIDENT WITHIN THE SYSTEM."
360 HTAB 5: PRINT "ALSO NOTE THAT ONLY COMPATIBLE"
370 HTAB 5: PRINT "FILES CAN BE MERGED: THEY "; INVERSE
: PRINT "MUST": NORMAL
380 HTAB 3: PRINT "HAVE THE SAME # OF FIELDS AND ALSO"
390 HTAB 4: PRINT "THE SAME OVERALL RECORD LENGTH."
400 VTAB 20: HTAB 10: PRINT "HIT "; INVERSE : PRINT " C
"; NORMAL : PRINT " FOR CATALOG ";G%
410 PRINT : HTAB 11: PRINT "OR "; INVERSE : PRINT "RTN";
: NORMAL : PRINT " FOR EXIT ";
420 POKE - 16368,0
430 GET Y%; PRINT "": IF Y% < > CHR% (13) AND Y% < > "
C" THEN 400
440 IF Y% = CHR% (13) THEN PRINT D*"RUN A.R.C."
450 PRINT "": GOTO 490
460 POKE 34,0: HOME : VTAB 5: HTAB 15: INVERSE : PRINT "
N O T I C E "; NORMAL
470 VTAB 10: HTAB (21 - LEN (FILE%)) / 2: PRINT FILE%: VTAB
12: HTAB 8: PRINT "HAS ONLY BEEN INITIALIZED": PRINT
: HTAB 9: PRINT "AND CONTAINS NO RECORDS";G%; HTAB 22
: PRINT "---": FOR D = 1 TO 2500: NEXT D
480 PRINT D*"CLOSE": RUN
490 POKE 34,0: HOME
500 PRINT D*"CATALOG": PRINT
510 PRINT "ENTER "; INVERSE : PRINT "1-ST"; NORMAL : PRINT
" FILE TO MERGE ";G%; INPUT " ";F1%; PRINT
520 FOR D = 1 TO 250: NEXT D
530 HTAB 7: INVERSE : PRINT "2-ND"; NORMAL : PRINT " FIL
E TO MERGE ";G%; INPUT " ";F2%; PRINT
540 IF F1% = F2% THEN VTAB (PEEK (37) - 1): GOTO 530
550 VTAB 23: HTAB 14: INVERSE : PRINT "S T A N D B Y": NORMAL
560 PRINT D*"OPEN ";F1% + BC%
570 PRINT D*"READ ";F1% + BC%
580 INPUT N1: INPUT R1: INPUT X1
590 PRINT D*"CLOSE"
600 PRINT D*"OPEN ";F2% + BC%
610 PRINT D*"READ ";F2% + BC%
620 INPUT N2: INPUT R2: INPUT X2
630 PRINT D*"CLOSE"

```

continued on next page

Applesoft Record Command System (ARC) (Cont.)

```

640 IF N1 < > N2 OR R1 < > R2 THEN HOME : VTAB 11: HTAB
5: INVERSE : PRINT "ERROR": NORMAL : PRINT " - FILES
ARE INCOMPATIBLE ";G%;G%: HTAB 8: PRINT "HIT ANY KEY
TO CONTINUE ": GET A%: PRINT "": GOTO 490
650 PRINT D% "OPEN ";F1%
660 PRINT D% "READ ";F1%,R0%: INPUT R1
670 PRINT D% "CLOSE"
680 PRINT D% "OPEN ";F2%
690 PRINT D% "READ ";F2%,R0%: INPUT R2
700 PRINT D% "CLOSE"
710 DIM RD%(R1 + R2),SRT%(R1 + R2),SO%(R1 + R2)
720 FILE% = F1%: GOSUB 200:RX = 0:IJ = 0: GOSUB 150:RT = R
EC
730 PRINT D% "CLOSE"
740 FILE% = F2%:RX = 0:IJ = REC: GOSUB 150:RT = RT + REC
750 PRINT D% "CLOSE"
760 REM ** SORT/PRINT **
770 LABEL = 0:REC = RT
780 VTAB 23: HTAB 12: PRINT "HIT ": INVERSE : PRINT "RTN
": NORMAL : PRINT " FOR MENU"
790 B1% = "LABEL":INDENT = 25
800 TEXT : HOME :RX = 0
810 POKE 34,0: VTAB 3: CALL - 868: FOR D = 1 TO 500: NEXT
D: VTAB 3: HTAB 15: INVERSE : PRINT "MERGE FILES": POKE
34,4: NORMAL
820 VTAB 3: INVERSE : PRINT F1%: HTAB (40 - LEN (F2%)):
PRINT F2%: NORMAL
830 T%(NF + 1) = H% + H% + H% + B%: FOR K = 1 TO NF:TX%(K)
= CHR$(13): NEXT K
840 HOME :L = 0: PRINT : PRINT " THE FOLLOWING FIELDS AR
E AVAILABLE ": HTAB 6: PRINT "FOR THE MERGED FILES ":
B1%: ": PRINT
850 FOR K = 1 TO NF STEP 2
860 PRINT " ": TAB(4 - (K > 9)):K1% = "T%(K): TAB(22 -
((K + 1) > 9)):K + 1%". "T%(K + 1)
870 NEXT K
880 L = L + 1
890 IF L > NF THEN 970
900 VTAB 21: HTAB 10: GOSUB 130: PRINT : VTAB 17: CALL -
868: PRINT "ENTER FIELD # FOR ": INVERSE : PRINT "L
INE ";L;G%: NORMAL : INPUT " ": TV%
910 IF L = 1 AND LEN (TV%) = 0 THEN 250
920 IF LEN (TV%) = 0 THEN 970
930 IF TV% = CHR$(21) OR TV% = CHR$(32) THEN 250
940 IF TV% = "2" THEN L = NF + 1:TV = NF: FOR I = 1 TO NF
:TV%(I) = I: NEXT : GOTO 970
950 TV = INT (VAL (TV%)): IF TV < 1 OR TV > NF THEN 900
960 TW%(L) = TV: GOTO 880
970 HOME :LINES = L - 1: PRINT : HTAB 8: PRINT "YOUR "JB
1%": WILL CONSIST OF ": HTAB 7: PRINT "-----"
980 FOR K = 1 TO LINES: HTAB (14 - (K > 9)): PRINT K1% ". "
:POKE(TW%(K)): NEXT K
990 POKE 34,20: VTAB 22: HTAB 10: PRINT "IS THIS CORRECT
(Y/N) ":G%: GET Y%: IF Y% = "N" THEN POKE 34,4: GOTO
840
1000 IF Y% = CHR$(21) OR Y% = CHR$(32) THEN 470
1010 IF Y% < > "Y" THEN 990
1020 LC = 0
1030 VTAB 21: CALL - 868: VTAB 22: HTAB 6: PRINT "MERGE
TWO OR MORE LINES (Y/N) ":G%: GET Y%: IF Y% = "N" THEN
1250
1040 IF Y% < > "Y" THEN 1030
1050 FOR K = 1 TO NF:TX%(K) = CHR$(13): NEXT K
1060 GOTO 1080
1070 VTAB 21: HTAB 16: CALL - 958: HTAB 16: PRINT "TO FI
ELD # ": VTAB 23: HTAB 10: GOSUB 130: GOTO 1120
1080 HTAB 1: VTAB 21: CALL - 958: INVERSE : PRINT "JOIN"
: NORMAL : PRINT " FIELD #":G%: INPUT " ":LX%: IF LEN
(LX%) = 0 THEN 1250
1090 VTAB 21: HTAB 16: CALL - 958:LX = VAL (LX%)
1100 IF LX < 1 OR LX > LINES - 1 THEN INVERSE : HTAB 14:
PRINT "INVALID": NORMAL : PRINT G%;G%: FOR D = 1 TO
500: NEXT D: VTAB 21: HTAB 1: GOTO 1080
1110 LC = LC - 1: GOTO 1070
1120 VTAB 21: HTAB 27: PRINT LX + 1%: "
1130 VTAB 21: HTAB 32: PRINT "ENTER ": FLASH : PRINT " ?
":G%: VTAB 21: HTAB 39: NORMAL :TX%(LX) = ""
1140 GET T1%: IF T1% = CHR$(13) THEN TX%(LX) = CHR$(1
3): GOTO 1250
1150 IF T1% < > " " THEN 1170
1160 TX%(LX) = TX%(LX) + T1%: GOTO 1180
1170 TX%(LX) = TX%(LX) + " " + T1% + " "
1180 ZZ% = T%(TW%(LX)) + T1% + T%(TW%(LX + 1))
1190 ZZ = LEN (ZZ%) + 2:ZZ = (40 - ZZ) / 2 - 1
1200 VTAB 23: HTAB 1: CALL - 958
1210 HTAB 22: INVERSE : PRINT T%(TW%(LX)): NORMAL : PRINT
" ": INVERSE : PRINT TX%(LX): NORMAL
1220 PRINT " ": INVERSE : PRINT T%(TW%(LX + 1)): NORMAL
1230 FOR D = 1 TO 500: NEXT D
1240 GOTO 1080
1250 POKE 34,4: POKE 35,23: HOME : PRINT : HTAB 6: PRINT
"THE "B1%:" WILL LOOK LIKE THIS : "
1260 HTAB 6: PRINT "-----"
1270 FOR K = 1 TO LINES: PRINT T%(TW%(K)):TX%(K): NEXT K
1280 VTAB 22: HTAB 14: PRINT "CORRECT (Y/N) ":G%: GET Y%
: IF Y% = "N" THEN 840
1290 IF Y% < > "Y" THEN 1280
1300 HOME : VTAB 12: HTAB 8: PRINT "WANT SORTED "B1%:" (
Y/N) ? ":G%: GET Y1%: PRINT Y1%: IF Y1% < > "Y" THEN
1570
1310 HOME : PRINT : PRINT "THE FOLLOWING LINES ARE AVAILA
BLE FOR"
1320 PRINT "THE SORT FIELD ": PRINT
1330 FOR K = 1 TO NF STEP 2: HTAB (4 - (K > 9)): PRINT K1
% ". "T%(K): TAB(22 - (K > 9)):K + 1%". "T%(K + 1): NEXT
K: PRINT
1340 PRINT : PRINT "SORT ON WHICH FIELD (BY NUMBER): ":G%
: INPUT " ":SF%:SF = VAL (SF%): IF SF < 1 OR SF > N
F THEN 1310
1350 POKE 34,0: POKE 35,24: HOME

```

```

1360 VTAB 10: HTAB (13 - LEN (T%(SF)) / 2): PRINT "### "
: INVERSE : PRINT "SORT ON "T%(SF): NORMAL : PRINT
" ###"
1370 PRINT : PRINT : INVERSE : PRINT "ASCENDING": NORMAL
: PRINT " OR ": INVERSE : PRINT "DESCENDING": NORMAL
: PRINT " SORT (A/D) ":G%: GET S%: PRINT S%: S1 =
(S% = "A")
1380 IF S% < > "A" AND S% < > "D" THEN VTAB ( PEEK (37
) - 1): GOTO 1370
1390 SS% = "ASCENDING": IF S% = "D" THEN SS% = "DESCENDING
"
1400 VTAB 12: CALL - 958: VTAB 17: HTAB 14: PRINT "S O R
T I N G"
1410 VTAB 7: HTAB 16: INVERSE : PRINT SS%: NORMAL
1420 FOR K = 1 TO REC
1430 SO%(K) = K
1440 IF SF = 1 THEN FL = 1: GOTO 1460
1450 FL = 0: FOR I = 1 TO SF - 1:FL = FL + TV%(I): NEXT I:
FL = FL + 1
1460 SRT%(K) = MID$(RD%(K),FL,TV%(SF))
1470 NEXT K
1480 M = 1
1490 M = 3 * M + 1: IF M < REC THEN 1490
1500 M = (M - 1) / 3: IF M < 1 THEN 1550
1510 FOR J = M + 1 TO REC:LL = J - M:SS% = SRT%(J):S = SO
%(J)
1520 VTAB 20: HTAB 19: INVERSE : PRINT " ": HTAB (19 +
(J < 100) + (J < 10)): PRINT J: NORMAL
1530 IF S1 = (SRT%(LL) > SS%) THEN SRT%(LL + M) = SRT%(LL
):SO%(LL + M) = SO%(LL):LL = LL - M: IF LL > 0 THEN 1
530
1540 SRT%(LL + M) = SS%:SO%(LL + M) = S: NEXT J: GOTO 1500
1550 REM SHELL SORT LIST DONE
1560 FOR D = 1 TO 1000: NEXT D
1570 IF LABEL > 0 THEN 1610
1580 HOME : VTAB 9: PRINT "HOW MANY ": INVERSE : PRINT "
CARRIAGE RETURNS": NORMAL : PRINT " FROM END OF": PRINT
: HTAB 11: PRINT "ONE "B1%:" TO NEXT ? ":G%: INPUT
" ":CR
1590 VTAB 15: HTAB 13: INVERSE : PRINT "V": NORMAL : PRINT
"IEW OR ": INVERSE : PRINT "P": NORMAL : PRINT "RIN
T ? ":G%: GET VP%: IF VP% < > "V" AND VP% < > "P" THEN
1590
1600 PRINT VP%: FOR D = 1 TO 500: NEXT D:XX = 6
1610 GOTO 1640
1620 HOME
1630 FOR K1 = 1 TO REC:RX = SO%(K1): GOTO 1790
1640 POKE 35,24: HOME
1650 VTAB 23: INVERSE : PRINT "PADDLE #1": NORMAL : PRINT
" CONTROLS SCROLL/PRINT SPEED !": POKE 35,21: VTAB 6:
FOR D = 1 TO 750: NEXT D
1660 IF VP% = "P" THEN PRINT D% "PR#1": PRINT CHR$(9): "
88N":
1670 IF Y1% = "Y" THEN 1620
1680 RX = RX + 1: IF RX < 1 THEN RX = 1
1690 IF RX < = REC THEN 1790
1700 PRINT D% "PR#0": PRINT : SPEED= 255: HTAB 15: INVERSE
: PRINT " END OF FILE ": NORMAL : FOR D = 1 TO 1000: NEXT
D
1710 IF VP% = "P" THEN PRINT D% "PR#1": PRINT CHR$(12):
PRINT D% "PR#0"
1720 POKE 35,24: HOME : VTAB 9
1730 PRINT : PRINT : POKE - 16368,0
1740 HTAB 5: PRINT "PRESS ": INVERSE
1745 PRINT "RETURN": NORMAL : PRINT " TO RE-FORMAT COPY"
: PRINT
1750 HTAB 7: PRINT "PRESS ": INVERSE : PRINT " C ": NORMAL
: PRINT " FOR ANOTHER COPY ":G%: GET Y%: PRINT
: IF Y% = CHR$(67) THEN HOME :LABEL = 1:RX = XR: GOTO
1580
1770 IF Y% = CHR$(13) THEN LABEL = 0:RX = XR: GOTO 880
1780 VTAB 14: GOTO 1750
1790 IF VP% = "P" THEN HTAB (INDENT): IF (XX + LINES + L
C) > 60 THEN PRINT CHR$(12):XX = 6: HTAB (INDENT)
1800 FL = 0: FOR K = 1 TO NF:P%(K) = MID$(RD%(RX),FL + 1
,TV%(K)):FL = FL + TV%(K): NEXT
1810 IF VP% = "P" THEN HTAB (INDENT)
1820 FOR K = 1 TO LINES
1830 L = LEN (P%(TW%(K))): IF L < 2 THEN L = 1: GOTO 1870
1840 FOR J = 1 TO L - 1
1850 IF MID$(P%(TW%(K)),L,1) < > " " THEN 1870
1860 L = L - 1: NEXT
1870 PRINT LEFT$(P%(TW%(K)),L):TX%(K): IF TX%(K) = CHR$(
13) THEN XX = XX + 1: IF VP% = "P" THEN HTAB (INDEN
T)
1880 SPEED= PDL (1): NEXT K
1890 ON CR + 1 GOTO 1930,1920
1900 FOR D = 1 TO CR: PRINT :XX = XX + 1: NEXT D
1910 GOTO 1930
1920 PRINT :XX = XX + 1
1930 IF Y1% < > "Y" THEN 1680
1940 NEXT K1: PRINT D% "PR#0": GOTO 1780
1950 END

```

continued on page 190

APPLE ART GALLERY

The Apple Art Gallery

by Edgar J. Young
64 Hillside Drive
Beacon Falls, CT 06403

This delightful program is (quite literally) a work of art. It creates a fascinating show by building geometric shapes, occasionally accompanied by sound. It is semi-passive in nature. You mostly sit back and enjoy it. The instructions are simple, and are an integral part of the program. This program indeed demonstrates that computers have many uses.

THE ORIGIN

I was intrigued by the computer generated geometric shapes that appear in magazines and on TV. So I decided to recreate some of these shapes using the high resolution graphics feature of the Apple II Plus computer. Soon I had an assortment of shape-generating programs. These were fun to watch, but viewers got bored waiting for me to load and run the programs. Hence, the idea for this program was born.

This program had to be fast so that people wouldn't lose interest. For the same reason it had to be rich in variety. Because art appeals to people of all ages, and because a lot of people are not computer or mathematically oriented, the instructions had to be simple and goof-proof. I believe the Art Gallery has succeeded in reaching these aims.

The people who have viewed this show are usually delighted. It seems to hold the fascinating lure of a camp fire. It's almost as much fun to watch the people watching it as it is to watch the screen. The viewers invariably tilt their heads to get different perspectives.

HOW IT WORKS

The program has two major parts. The first part is the set-up and control function. It is the framework in which the pictures are shown. The second part is a group of subroutines which are the mathematical shape generators.

SET-UP AND CONTROL

The set-up and control section is structured as follows:

* Display the Copyright (lines 40-60)

This is a reminder to keep honest people honest.

* The Instructions (lines 100-310)

The instructions are single keystroke entries that can be made at any time when the program is running. The response time for the instructions is minimized to maintain viewer interest.

The instructions are:

- I — will cause the program to stop and display the instructions.
- H — will cause the program to pause with whatever it is doing until 'H' is pressed again. This feature allows you to halt a shape generation in progress so you can see how it is being done. It also allows a longer appreciation time if you want to view a finished shape longer than the normally allowed time.

S — will cause the program to stop and ask for an art form number. The program will then show **sequentially numbered art forms** beginning at the selected number.

R — will cause the program to stop and ask for an art form number. The program will then show that art form until it receives a different command. Each of the forms is generated using the RND(X) function. So **there are variations within a selected art form**. The 'R' command allows you to explore these variations. Some of the art forms vary more than others.

N — will change the status of the numbering feature, turning it off if it was on, and on if it was off. This control performs the status change immediately, but the change is not apparent until the next 'slide' is viewed.

SPACE — pressing the space bar causes the program to stop what it is doing and start generating 'slides' in a random order. This is a good command for parties because you can just leave the program running and not show the same thing twice.

ESC — will immediately clear the screen and put your Apple back into Applesoft BASIC.

The array **P(4,150,2)** is one of the secrets to the speed of this program. It is the point location array for the shapes. The appreciation time at the end of each geometric construction is being used by your Apple to generate the next shape and load it into this array. **It is far faster to HPLOT to elements in an array than to mathematically generate and plot each point at the time of viewing.**

* GOSUB 500 — The Control Center (lines 490-660)

This interesting subroutine is what makes the control so fast. The shape generating routines are laced with **GOSUB 500**'s. Each time this routine is called, it checks to see if a key has been pressed. If not, control is given back to the shape generator. If a key has been pressed, the validity of the key is checked. If a valid command is issued then control is transferred. If the command demands a totally new program direction, then a **CALL 54915** is executed. This handy **CALL zeroes the GOSUB stack without affecting anything else**. So you can break in the middle of nested subroutines without having to POP off the GOSUB stack. This is vital because **GOSUB 500** is being called from a large number of places within the program. And many of the places are at different levels of GOSUB nesting.

* Art Form Selection (lines 670-790)

This is where the art form is selected, and where execution moves out to the shape generating routines.

* Art Form Number Input (lines 800-940)

This subroutine accepts answers to the 'which art form' questions asked by the 'S' and 'R' commands. It refuses to accept improper or impossible inputs.

* Art Form Numbering (lines 950-990)

This subroutine displays the art form number on the screen when the numbering feature is activated.

* Choose Screen Color (lines 1000-1020)

This is one of the added variety features of the program. The graphing screen is randomly black with white lines or white with black lines.

THE ART FORM GENERATORS

The art forms are mathematically defined shapes. The **RND(X)** function is used for defining parameter values for the shape generating equations. Therefore, **each of the 33 different art forms is actually a family of art forms** in itself. There is so much variation within some of the art forms that no two of them are alike.

It is difficult to describe most of the shapes, so I won't. I'll let you be surprised. When debugging this program, you will probably only be able to eliminate syntax (typing) errors. Programming errors will be difficult because you won't know exactly what you are aiming for. But don't worry, your mistakes may lead to new art forms that are more beautiful than the original. If your results are pleasing to the eye then you aren't too far off.

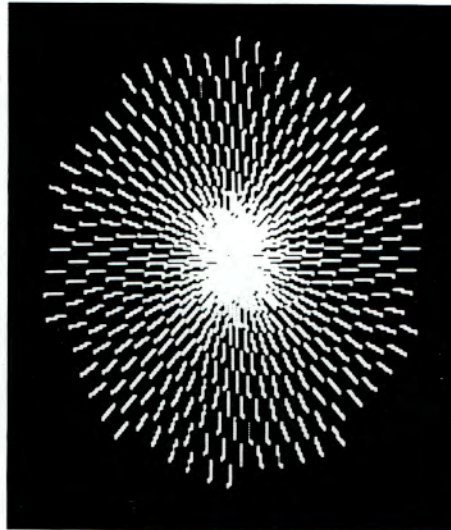
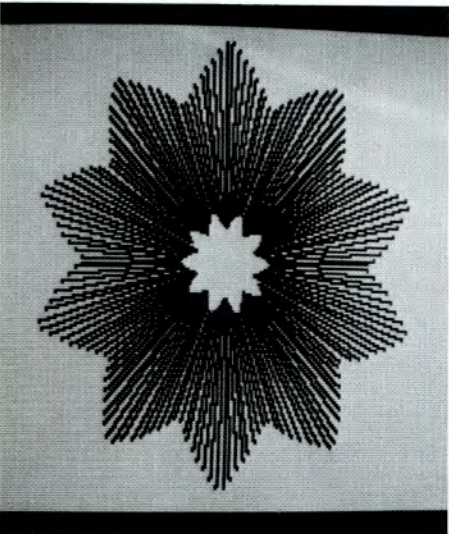
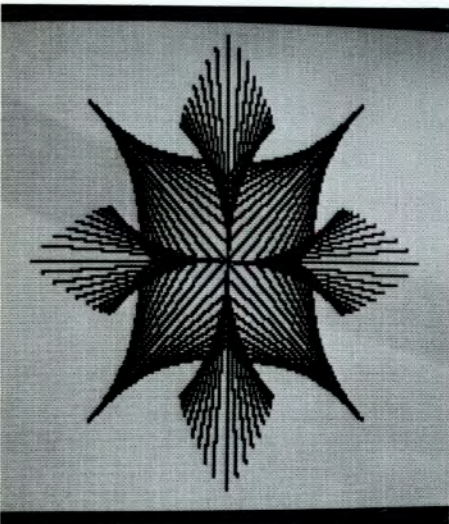
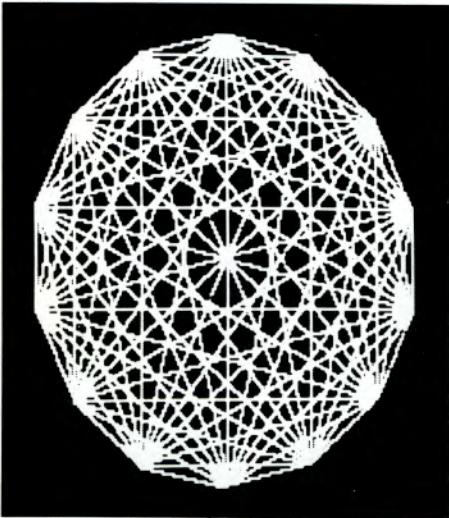
CONTROLLING YOUR OWN ART

To use the control part of this program as a vehicle for displaying your own art, arrange your shape generating subroutines in the following order:

1. Use the **RND(X)** function to generate the parameters that define the shape of your art. This will add variety to the viewing.
2. Generate the points of the art form and load them into **P(4,150,2)**. Be sure to **GOSUB 500** in the innermost loop to maintain rapid program control.
3. **GOSUB 950** to number the art form and/or clear and select the background screen for the next slide.
4. **HPLOT** the array **P(4,150,2)**. Remember to include **GOSUB 500** in the innermost loop or where appropriate to maintain program control during plotting.
5. **RETURN**

continued on next page

The Apple Art Gallery (Cont.)



EQUIPMENT NEEDED

Apple II Plus with 48K RAM

Applesoft in ROM

1 Disk Drive

Monitor (preferably a color monitor or color TV)

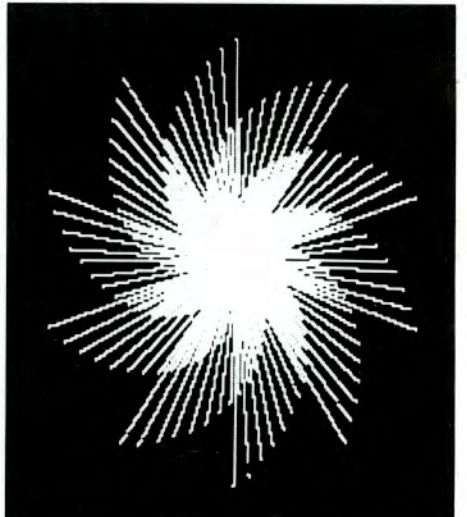
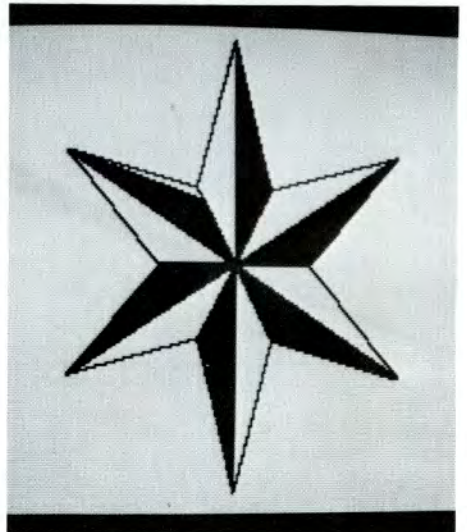
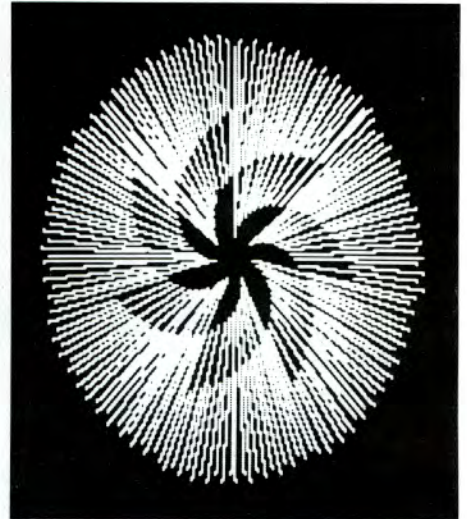
SUMMARY

This program is great to show to non-computer users. It gets them asking questions about the computer. It is also a great conversation piece at a party. It can be used for attracting attention at a display booth, or for some other form of advertising. A friend used it to help get ideas for designing a new company logo. ENJOY!!

VARIABLE LIST FOR APPLE ART GALLERY

A	Answer to questions
AS	Answer to questions
A1	ASC(I)
A2	ASC(H)
A3	ASC(S)
A4	ASC(R)
A5	ASC(N)
A6	ASC(SPACE)
A7	ASC(ESC)
C	Input to # Art Form question
C\$	Input to # Art Form question
C1\$	Input to # Art Form question
CG	Color of graph lines
CS	Color of screen
CX	Center of screen in X direction
CY	Center of screen in Y direction
DE	Delay (FOR DE = 1 TO 100: NEXT)
FL	Flag to denote art form numbering
FR	Flag to denote running an art form continuously
FS	Flag to denote sequencing art forms
IX	Starting cursor position for screen answer
NF	Total number of art forms
NN	Number of art form currently showing
PI	3.14159...
PK	-16384: Peek to check for keystroke
RM	Maximum permissible radius
RP	1/2 maximum radius
X	Position of cursor on screen

The remainder of the variables are used in the form definition and plotting routines. Remember: if you use other variable names, they have to be initialized before graphing to avoid shifting the HGR2 screen.



The Apple Art Gallery (Cont.)

LIST

```

2 REM *****
3 REM * APPLE ART GALLERY *
4 REM * BY EDGAR J. YOUNG *
5 REM * COPYRIGHT (C) 1982 *
6 REM * BY MICRO-SPARC INC *
7 REM * LINCOLN, MA. 01773 *
8 REM * ALL RIGHTS RESERVED *
9 REM *****
35 LOMEM: 24576
40 HGR2 : TEXT : HOME : VTAB 7: HTAB 9: PRINT "APPLE
  ART GALLERY": PRINT : PRINT
50 HTAB 9: PRINT "COPYRIGHT (C) 1982": PRINT : PRINT
60 HTAB 9: PRINT "BY MICRO-SPARC, INC."
80 GOSUB 320: REM VARIABLE INITIALIZATION
90 VTAB 23: HTAB 6: PRINT "PRESS SPACE BAR TO CONTINU
  E..."
92 POKE - 16368,0
94 A = RND (1): IF PEEK ( - 16384) < 128 GOTO 94: REM
  RANDOMIZES RND(1)
100 REM INSTRUCTIONS
110 TEXT : HOME : PRINT
120 PRINT " THIS PROGRAM RANDOMLY DISPLAYS A"
130 PRINT "VARIETY OF GRAPHIC ART FORMS. TO"
140 PRINT "CONTROL IT, USE THE FOLLOWING SINGLE"
150 PRINT "KEY INSTRUCTIONS AT ANY TIME:"
160 PRINT : PRINT " I - DISPLAY THESE 'INSTRUCTIONS
  '"
170 PRINT : PRINT " H - 'HOLD' THIS PICTURE UNTIL <
  H>"
180 HTAB 8: PRINT "IS PRESSED AGAIN"
190 PRINT : PRINT " S - 'SEQUENTIALLY' RUN ALL ART
  FORMS"
200 PRINT : PRINT " R - 'RUN' ONE ART FORM CONTINUO
  USLY"
210 HTAB 8: PRINT "UNTIL THE SPACE BAR IS PRESSED"
220 PRINT : PRINT " N - START OR STOP 'NUMBERING'"
230 HTAB 8: PRINT "THE ART FORMS"
240 PRINT : PRINT " SPACE BAR - CONTINUE RANDOM DIS
  PLAYS"
250 PRINT : PRINT " ESC - ESCAPE THIS PROGRAM"
260 CALL 62450: REM CLEAR HGR2 SCREEN
265 POKE - 16368,0
270 PRINT : HTAB 20: PRINT "YOUR COMMAND? "; GET A$:
  A = ASC (A$)
280 IF A = A1 OR A = A2 GOTO 100
290 IF A = A5 THEN FL = 1: GOTO 670
300 GOSUB 520
310 GOTO 100
320 REM CONSTANTS AND DIMENSIONS
400 PI = 3.141592654: CX = 140: CY = 95: RM = 90: RP = 45
410 A1 = 73: A2 = 72: A3 = 83: A4 = 82: A5 = 78: A6 = 32: A7
  = 27
420 FL = 0: FS = 0: FR = 0: P = 0
430 CS = 0: CG = 0
440 PK = - 16384
450 NF = 33
460 C$ = "": C1$ = ""
470 DIM P(4,150,2): REM
480 RETURN
490 REM PROGRAM CONTROL
500 IF PEEK (PK) < 128 THEN RETURN
510 GET A$: A = ASC (A$)
520 IF A < > A1 AND A < > A2 AND A < > A3 AND A < > A4
  AND A < > A5 AND A < > A6 AND A < > A7 THEN RETURN
530 IF A = A1 THEN CALL 54915: FS = 0: FR = 0: GOTO 100
540 REM CALL 54915 RESETS THE GOSUB STACK TO ZERO WITHOU
  T AFFECTING ANYTHING ELSE
550 IF A = A2 THEN GET A$: RETURN
560 IF A = A3 GOTO 640
570 IF A = A4 GOTO 610
580 IF A = A5 THEN FL = ABS (FL - 1): RETURN
590 IF A = A6 THEN CALL 54915: FS = 0: FR = 0: GOTO 670
600 TEXT : HOME : END
610 CALL 54915
620 FS = 0
630 FR = 1: TEXT : HOME : VTAB 10: PRINT " SELECT THE ART
  FORM (1 TO "; NF; ") THAT YOU WANT TO RUN CONTINUOUSLY.
  ": GOTO 800
640 CALL 54915
650 FR = 0
660 FS = 1: TEXT : HOME : VTAB 10: PRINT " SELECT THE ART
  FORM (1 TO "; NF; ") THAT YOU WANT TO START THE SEQUENC
  E FROM.": GOTO 800
670 REM ART FORM SELECTION
680 IF FR = 1 GOTO 710
690 IF FS = 1 THEN NN = NN + 1: IF NN = NF + 1 THEN NN =
  1
700 IF FS < > 1 THEN NN = INT ( RND (1) * NF + 1)
710 ON NN (NN / 10 + 1) GOTO 720,740,760,780
720 ON NN GOSUB 4990,3320,2950,1070,1290,1700,3850,4060,3
  320
730 GOTO 670
740 ON NN - 9 GOSUB 1060,1300,1400,2600,3330,4320,1980,16
  90,1080,3340
750 GOTO 670
760 ON NN - 19 GOSUB 2320,3800,1280,3340,1500,1050,1710,3
  330,1970,1300
770 GOTO 670

```

```

780 ON NN - 29 GOSUB 3400,1040,3760,3450
790 GOTO 670
800 REM ART FORM NUMBER INPUT
810 PRINT : PRINT " PRESS <RETURN> WHEN YOU HAVE TYPED I
  N YOUR ENTRY...";
820 IX = POS (0) + 1
830 GET A$: C = ASC (A$): IF C < 49 OR C > 57 THEN HTAB
  IX: GOTO 830
840 C1$ = A$: C$ = A$: PRINT A$: X = POS (0) + 1
850 GET A$: C = ASC (A$): IF C = 13 GOTO 920
860 IF C = 8 THEN X = POS (0): HTAB X: PRINT " "; HTAB
  X: GOTO 830
870 IF C < 48 OR C > 57 THEN HTAB X: GOTO 850
880 C$ = C1$ + A$: PRINT A$: X = POS (0) + 1
890 GET A$: C = ASC (A$): IF C = 13 GOTO 920
900 IF C = 8 THEN X = POS (0): HTAB X: PRINT " "; HTAB
  X: C$ = C1$: GOTO 850
910 GOTO 890
920 NN = VAL (C$): IF NN < 1 OR NF < NN THEN HTAB IX: PRINT
  " "; HTAB IX: GOTO 830
930 IF FS = 1 THEN NN = NN - 1
935 HOME : VTAB 10: HTAB 14: PRINT "I'M THINKING"
940 GOTO 670
950 REM ART FORM NUMBERING SUBROUTINE
960 IF FL = 1 GOTO 980
970 GOTO 1010
980 TEXT : HOME : VTAB 10: HTAB 13: PRINT "ART FORM # "; N
  N
990 CALL 62450: FOR DE = 1 TO 100: GOSUB 500: NEXT DE
1000 REM CHOOSE SCREEN COLOR - WHITE ON BLACK OR BLACK O
  N WHITE
1010 IF RND (1) - .5 > 0 THEN HGR2 : HCOLOR = 7: CS = 4: RETURN
1020 HCOLOR = 7: HGR2 : HPLLOT CX, CY: CALL 62454: HCOLOR = 4
  : CS = 7: RETURN
1030 REM ART FORMS 31,25,10,4,18
1040 FOR DE = 1 TO 200: GOSUB 500: NEXT DE: P = INT ( RND
  (1) * 11 + 10): GOTO 1100
1050 P = INT ( RND (1) * 21 + 90): GOTO 1100
1060 P = INT ( RND (1) * 51 + 20): GOTO 1100
1070 P = INT (( RND (1) * 21 + 20) / 2) * 2: GOTO 1100
1080 P = INT ( RND (1) * 61 + 20)
1090 REM DEFINE POINTS AROUND A CIRCLE
1100 I = 1: AN = 2 * PI / P
1110 P(I,1,1) = CX: P(I,1,2) = CY - RM: K = P
1120 FOR J = 2 TO INT (P / 2 + 1): PX = RM * SIN ((J - 1)
  ) * AN: P(I,J,1) = CX + PX: P(I,J,2) = CY - PX: P(I,J,2)
  = CY - RM * COS ((J - 1) * AN): P(I,K,2) = P(I,J,2)
  : K = K - 1: GOSUB 500: NEXT J
1130 GOSUB 950
1140 REM GRAPH ROUTINE
1150 IF NN < > 31 GOTO 1180
1160 FOR J = 1 TO P - 1: FOR K = J + 1 TO P: GOSUB 1260: NEXT
  K, J
1170 RETURN
1180 J = 1: IF NN = 18 THEN J = INT ( RND (1) * P + 1)
1190 FOR K = 1 TO P: GOSUB 1260: NEXT K: IF NN = 25 THEN
  RETURN
1200 IF NN = 4 OR NN = 18 GOTO 1230
1210 J = 2: FOR K = 3 TO P: GOSUB 1260: NEXT K
1220 J = P: FOR K = 1 TO P: GOSUB 1260: NEXT K: RETURN
1230 IF NN = 18 GOTO 1250
1240 J = P / 2 + 1: FOR K = 1 TO P: GOSUB 1260: NEXT K: RETURN
1250 J = INT ( RND (1) * P + 1): FOR K = 1 TO P: GOSUB 12
  60: NEXT K: RETURN
1260 HPLLOT P(I,J,1), P(I,J,2) TO P(I,K,1), P(I,K,2): GOSUB
  500: RETURN
1270 REM ART FORMS 22,5,11
1280 P = INT ( RND (1) * 8 + 5): C = P: GOTO 1330
1290 P = INT ( RND (1) * 9 + 3): C = INT ( RND (1) * 21 +
  30): GOTO 1330
1300 P = 33
1310 IF NN = 11 THEN C = INT ( RND (1) * 8 + 5)
1320 IF NN = 29 THEN C = INT ( RND (1) * 11 + 20)
1330 FOR DE = 1 TO 200: GOSUB 500: NEXT DE
1340 AN = 2 * PI / P: FOR I = 1 TO P: P(I,1,1) = RP * SIN
  (AN * I): P(I,1,2) = RP * COS (AN * I): GOSUB 500: NEXT
  I
1350 GOSUB 950
1360 REM GRAPH ROUTINE
1370 FOR I = 1 TO C: PY = CY - RP * COS (I * 2 * PI / C):
  PX = CX + RP * SIN (I * 2 * PI / C): HPLLOT PX, PY - R
  P
1380 FOR A = 1 TO P: HPLLOT TO PX + P(1,A,1), PY - P(1,A,2
  ): NEXT A: GOSUB 500: NEXT I: RETURN
1390 REM ART FORM 12
1400 P = INT ( RND (1) * 25 + 6)
1410 FOR DE = 1 TO 200: GOSUB 500: NEXT
  DE
1420 IX = 130 / P: IY = 90 / P: X = 140: Y = 95: PX = 130: PY =
  0
1430 GOSUB 950
1440 REM GRAPH ROUTINE
1450 GC = 2
1460 FOR I = 1 TO P + 1
1470 HPLLOT X, Y + PY TO X + PX, Y TO X, Y - PY TO X - PX, Y TO
  X, Y + PY: GOSUB 500
1480 PX = PX - IX: PY = PY + IY: NEXT I: RETURN
1490 REM ART FORM 24
1500 FOR DE = 1 TO 300: GOSUB 500: NEXT DE
1510 X = INT ( RND (1) * 180): Y = INT ( RND (1) * 192)
1520 GOSUB 2740: GOSUB 950
1530 REM GRAPH ROUTINE
1540 HPLLOT X, Y
1550 FOR I = 1 TO 300
1560 L = INT ( RND (1) * RM - RP)
1570 IX = 0: IY = 0: IF SGN ( RND (1) - .5) > 0 THEN IX =
  L: GOTO 1590
1580 IY = L
1590 X = X + IX: Y = Y + IY
1600 IF X > 279 THEN X = X - 10: GOTO 1600
1610 IF X < 0 THEN X = X + 10: GOTO 1610

```

continued on next page

The Apple Art Gallery (Cont.)

```

1620 IF Y > 191 THEN Y = Y - 10: GOTO 1620
1630 IF Y < 0 THEN Y = Y + 10: GOTO 1630
1640 H$PLOT TO X,Y
1650 POKE 6, INT ((X + Y) / 2): POKE 7, INT (RND (1) * 4
+ 1): CALL 768
1660 GOSUB 500
1670 NEXT I: RETURN
1680 REM ART FORM 17,6,26
1690 P = INT (RND (1) * 91 + 10): PR = INT (RND (1) * 5
+ 10): N = INT (RND (1) * 4 + 3): GOTO 1730
1700 P = INT (RND (1) * 121 + 30): PR = 100: N = 1: GOTO 1
730
1710 N = INT (RND (1) * 5 + 2): R = 130
1720 P = INT (RND (1) * 91 + 10): PR = INT (RND (1) * 5
+ 10): GOTO 1740
1730 R = 130
1740 IF PR = 100 THEN S = INT (P / 2): GOTO 1770
1750 A = 1 - PR / 100
1760 S = INT ((- ATN (A / SQR (- A * A + 1)) + 1.5708
) * P / PI + .5)
1770 I = 1: J = 1
1780 I = I + 1: J = J + S: IF J > P THEN J = J - P
1790 IF I < P AND J = 1 AND P < 150 THEN P = P + 1: GOTO
1750
1800 GOSUB 500: IF I < P GOTO 1780
1810 AN = 2 * PI / P: IR = CY / CX * R: K = P: P(1,1) = CX:
P(1,2) = CY - IR
1820 FOR I = 2 TO INT (P / 2 + 1)
1830 A = (I - 1) * AN: PX = R * SIN (A): P(1,1) = CX + PX
: P(1,2) = CY - IR * COS (A): P(1
,K,2) = P(1,2): K = K - 1
1840 GOSUB 500: NEXT I
1850 IF R = 130 THEN GOSUB 950
1860 REM GRAPH ROUTINE
1870 J = 1: H$PLOT P(1,J,1),P(1,J,2)
1880 FOR I = 1 TO P
1890 J = J + S: IF J > P THEN J = J - P
1900 H$PLOT TO P(1,J,1),P(1,J,2)
1910 GOSUB 500: NEXT I
1920 R = R * COS (S * PI / P) - 3
1930 N = N - 1: IF N = 0 THEN RETURN
1940 IF NN = 26 GOTO 1720
1950 GOTO 1810
1960 REM ART FORMS 28 & 16
1970 PR = 1: GOTO 1990
1980 PR = - 1: FOR DE = 1 TO 200: GOSUB 500: NEXT
1990 P = INT (RND (1) * 12 + 4): PR = PR * (RND (1) * .4
+ .25)
2000 R = 90: PR = R * PR
2010 A = 0: SA = 0: CA = 0: J = 0
2020 FOR I = 1 TO 2 * P
2030 P(1,1) = CX + R * SIN ((I - 1) / P * PI)
2040 P(1,2) = CY - R * COS ((I - 1) / P * PI)
2050 I = I + 1
2060 P(1,1) = CX + PR * SIN ((I - 1) / P * PI)
2070 P(1,2) = CY - PR * COS ((I - 1) / P * PI)
2080 GOSUB 500
2090 NEXT I
2100 GOSUB 950
2110 REM GRAPH ROUTINE
2120 FOR I = 1 TO 2 * P
2130 H$PLOT CX,CY TO P(1,1),P(1,2)
2140 NEXT I
2150 H$PLOT P(1,1),P(1,2)
2160 FOR I = 1 TO 2 * P
2170 H$PLOT TO P(1,1),P(1,2)
2180 GOSUB 500
2190 NEXT I
2200 H$PLOT TO P(1,1),P(1,2)
2210 IF PR < 0 GOTO 2300
2220 FOR I = 1 TO 2 * P
2230 I = I + 1
2240 A = (I - 1) / P * PI: SA = SIN (A): CA = COS (A)
2250 FOR J = 1 TO PR
2260 H$PLOT P(1,I - 1,1),P(1,I - 1,2) TO CX + J * SA,CY -
J * CA
2270 GOSUB 500
2280 NEXT J
2290 NEXT I
2300 RETURN
2310 REM ART FORM 20
2320 L = INT (RND (1) * 6 + 4)
2330 C = INT (RND (1) * 11 + 5)
2340 N = RND (1)
2350 I = 1: GOSUB 2460
2360 FOR J = 1 TO L: FOR K = 1 TO 2: P(3,J,K) = P(1,J,K): NEXT
K,J
2370 FOR DE = 1 TO 300: GOSUB 500: NEXT DE: GOSUB 950
2380 REM GRAPH ROUTINE
2390 FOR N = 2 TO C
2400 FOR J = 1 TO L: FOR K = 1 TO 2: P(2,J,K) = P(3,J,K): NEXT
K,J
2410 I = 3: GOSUB 2460
2420 FOR M = 1 TO L: H$PLOT P(2,M,1),P(2,M,2) TO P(3,M,1),
P(3,M,2): NEXT M
2430 GOSUB 500: NEXT N
2440 FOR M = 1 TO L: H$PLOT P(3,M,1),P(3,M,2) TO P(1,M,1),
P(1,M,2): NEXT M
2450 RETURN
2460 REM POINT LOCATION
2470 X = 20 + RND (N) * 239: Y = 20 + RND (N) * 151
2480 A = RND (N) * 2 * PI: CA = COS (A): SA = SIN (A)
2490 R = RND (N) * 90
2500 IX = X + R * CA
2510 IY = Y + R * SA
2520 GOSUB 500
2530 IF IX < 0 OR IX > 279 OR IY < 0 OR IY > 191 THEN R =
R - 1: GOTO 2500
2540 FOR M = 1 TO L
2550 P(1,M,1) = X + R * M / L * CA
2560 P(1,M,2) = Y + R * M / L * SA

```

```

2570 NEXT M
2580 RETURN
2590 REM ART FORM 13
2600 FOR DE = 1 TO 300: GOSUB 500: NEXT DE
2610 GOSUB 2740
2620 GOSUB 950
2630 REM GRAPH ROUTINE
2640 Y = 191 * RND (1): X = 0
2650 H$PLOT X,Y
2660 I = RND (1) * 4 + 1: X = X + I: IF X > 279 GOTO 2720
2670 Y = 191 * RND (1)
2680 H$PLOT TO X,Y
2690 POKE 6, INT (Y): POKE 7, INT (I): CALL 768
2700 GOSUB 500
2710 GOTO 2660
2720 RETURN
2730 REM 'BOOP' SOUND ROUTINE, POKE 6 AND 7 TO VARY PITC
H AND DURATION
2740 POKE 768,173
2750 POKE 769,48
2760 POKE 770,192
2770 POKE 771,136
2780 POKE 772,208
2790 POKE 773,4
2800 POKE 774,198
2810 POKE 775,7
2820 POKE 776,240
2830 POKE 777,8
2840 POKE 778,202
2850 POKE 779,208
2860 POKE 780,246
2870 POKE 781,166
2880 POKE 782,6
2890 POKE 783,76
2900 POKE 784,0
2910 POKE 785,3
2920 POKE 786,96
2930 RETURN
2940 REM ART FORM 19
2950 FOR DE = 1 TO 300: GOSUB 500: NEXT DE: GOSUB 950
2960 REM GRAPH ROUTINE
2970 IX = 279: IY = 191: A = 100: R = 1: J = 1
2980 GOSUB 3090
2990 N = 50 - 20 * RND (1)
3000 J = J + 1: IF J > 6 THEN RETURN
3010 FOR I = 1 TO N: P(1,I,1) = IX * RND (R): P(1,I,2) = I
Y * RND (R): NEXT
3020 POKE 783, INT (RND (R) * 90 + A)
3030 C = INT (RND (R) * 8): H$COLOR = C: IF (CS = 7 AND (C
= 7 OR C = 3)) OR (CS = 4 AND (C = 0 OR C = 4)) GOTO
3030
3040 X = RND (R) * IX: Y = RND (R) * IY
3050 FOR I = 1 TO N: H$PLOT X,Y TO P(1,I,1),P(1,I,2): CALL
768: NEXT
3060 GOSUB 500
3070 GOTO 2990
3080 REM LAZER SOUND ROUTINE, POKE 783 FOR TONE, POKE 76
9 FOR MULTIPLE TONES
3090 POKE 768,160
3100 POKE 769,1
3110 POKE 770,162
3120 POKE 771,0
3130 POKE 772,138
3140 POKE 773,24
3150 POKE 774,233
3160 POKE 775,1
3170 POKE 776,208
3180 POKE 777,252
3190 POKE 778,141
3200 POKE 779,48
3210 POKE 780,192
3220 POKE 781,232
3230 POKE 782,224
3240 POKE 784,208
3250 POKE 785,242
3260 POKE 786,136
3270 POKE 787,208
3280 POKE 788,237
3290 POKE 789,96
3300 RETURN
3310 REM ART FORMS 20-22
3320 RS = 0: EL = 1: SL = RND (1) * .5: GOTO 3350
3330 SL = RND (1) * .4 + .4: EL = SL: RS = RND (1) * (.8 -
SL): GOTO 3350
3340 SL = RND (1): EL = RND (1): RS = (1 - SL) * RND (1) *
.8: IF ABS (EL - SL) < .3 GOTO 3340
3350 NS = INT (RND (1) * 8 + 5)
3360 OV = INT (RND (1) * 2 + 1)
3370 NA = INT ((RND (1) * 41 + 120) / NS)
3380 NR = OV * NA / ((INT (NA / OV) * OV + 1) * NS)
3390 GOTO 3540
3400 RS = RND (1) * .5: SL = RND (1) * .36 + .04: EL = SL
3410 NS = INT (RND (1) * 6 + 5)
3420 NR = (1 - EL - RS) / (NS * (SL + 4 / RM))
3430 NA = INT (NR * (RND (1) * 41 + 100))
3440 GOTO 3540
3450 IF RND (1) - .5 > 0 THEN RS = 0: GOTO 3470
3460 RS = RND (1) * .2 + .1
3470 SL = RND (1) * .04 + .06
3480 IF RND (1) - .5 > 0 THEN EL = SL: GOTO 3500
3490 EL = RND (1) * SL * .5
3500 NR = RND (1) * 31 + 20
3510 NA = INT (RND (1) * 31 + 40)
3520 NS = INT (500 / NA)
3530 IF ABS (NR / NS - INT (NR / NS)) < .1 GOTO 3500
3540 FOR DE = 1 TO 200: GOSUB 500: NEXT DE: GOSUB 950
3550 REM GRAPH ROUTINE
3560 DI = SGN (RND (1) - .5)
3570 IA = 2 * PI * NR / NA * DI
3580 FOR I = 0 TO NS - 1
3590 ISR = RM * (1 - EL - RS) / NA
3600 ILR = RM * (1 - RS - SL) / NA
3610 IF NN = 9 OR NN = 27 OR NN = 19 THEN ISR = ISR * 2: I
LR = ILR * 2
3620 LR = RM * (RS + SL): SR = RM * RS

```


Apple Art Gallery (Cont.)

```

3630 AN = I * 2 * PI / NS * DI
3640 IF NN = 9 OR NN = 27 OR NN = 19 THEN AN = AN - PI *
      DI / NS
3650 FOR J = 0 TO NA
3660 CA = COS (AN):SA = SIN (AN)
3670 HPLLOT CX + SR * SA,CY - SR * CA TO CX + LR * SA,CY -
      LR * CA
3680 AN = AN + IA:LR = LR + ILR:SR = SR + ISR
3690 IF LR > RM + ILR / 4 AND NA - INT (NA / 2) * 2 = 0 THEN
      ILR = - ILR:LR = LR + 2 * ILR:ISR = - ISR:SR = SR +
      2 * ISR
3700 IF LR > RM + ILR / 4 THEN ILR = - ILR:LR = LR + ILR
      :ISR = - ISR:SR = SR + ISR
3710 GOSUB 500
3720 NEXT J
3730 NEXT I
3740 RETURN
3750 REM ART FORMS 32,21,7
3760 IF RND (1) - .5 > 0 THEN C = RND (1) * 41 + 30: GOTO
      3780
3770 C = 100
3780 NS = INT (RND (1) * C / B + 4)
3790 P = INT (RND (1) * 11 + 5): GOTO 3830
3800 C = 200
3810 NS = INT ((RND (1) * C / B + 4) / 2) * 2
3820 P = INT ((RND (1) * 11 + 2) / 2) * 2 + 1
3830 PR = RND (1) * .6 + .2
3840 GOTO 3890
3850 C = RND (1) * 20 + 40
3860 NS = INT (RND (1) * 3 + 3)
3870 P = INT (RND (1) * 5 + 5)
3880 PR = - 1
3890 R = RM:IA = PI / P
3900 IF NS = 1 THEN IR = 0: GOTO 3920
3910 IR = C * R / (NS - 1) / 100
3920 FOR DE = 1 TO 200: GOSUB 500: NEXT DE
3930 GOSUB 950
3940 REM GRAPH ROUTINE
3950 FOR I = 1 TO NS
3960 IF I = NS AND NN = 32 GOTO 4040
3970 HPLLOT CX,CY - R
3980 AN = 0:RR = R * PR
3990 FOR J = 1 TO P
4000 AN = AN + IA: HPLLOT TO CX + RR * SIN (AN),CY - RR *
      COS (AN)
4010 AN = AN + IA: HPLLOT TO CX + R * SIN (AN),CY - R * COS
      (AN)
4020 GOSUB 500: NEXT J
4030 R = R - IR
4040 NEXT I
4050 RETURN
4060 REM ART FORM B
4070 N = INT (RND (1) * 2 + 2)
4080 HO = RND (1) * .8 + .2
4090 HI = RND (1) * HO + 1 - HO
4100 IF RND (1) - .5 > 0 THEN RO = 0:RI = 0: GOTO 4140
4110 RO = RND (1) * PI * 3 / 18
4120 IF N = 2 THEN RI = 0: IF RND (1) - .5 > 0 THEN RI =
      RO
4130 IF N = 3 THEN RI = RO / 2
4140 P = INT (RND (1) * 18 + 12)
4150 S = INT (P * (RND (1) * .5 + .25))
4160 C = INT (RND (1) * 2 + 1)
4170 GOSUB 4670
4180 GOSUB 4770
4190 GOSUB 4880
4200 GOSUB 950
4210 REM GRAPH ROUTINE
4220 HPLLOT P(1,1,1),P(1,1,2)
4230 I = 1:L = 1:J = 1
4240 FOR K = 1 TO P + 1
4250 I = I + L: IF I > N + 1 THEN L = - 1:I = I - 2
4260 J = J + S: IF J > P THEN J = J - P
4270 HPLLOT TO P(I,J,1),P(I,J,2)
4280 GOSUB 500
4290 IF I < > 1 GOTO 4250
4300 L = 1: NEXT K
4310 RETURN
4320 REM ART FORM 15
4330 N = 1
4340 HO = RND (1) * .5 + .5
4350 HI = RND (1) * (1 - HO) + HO
4360 RO = RND (1) * PI * 1 / 9:RI = RO
4370 P = INT (RND (1) * 18 + 13)
4380 S = INT (P * (RND (1) * .25 + .25))
4390 C = INT (RND (1) * 2 + 1)
4400 GOSUB 4670
4410 GOSUB 4770
4420 P1 = P:S1 = S
4430 P = P1 + 25
4440 S = INT (S1 / P1 * P)
4450 GOSUB 4670
4460 GOSUB 4880
4470 GOSUB 950
4480 REM GRAPH ROUTINE
4490 J = 1
4500 HPLLOT P(1,1,1),P(1,1,2)
4510 FOR K = 1 TO P1 + 1
4520 J = J + S1: IF J > P1 THEN J = J - P1
4530 HPLLOT TO P(2,J,1),P(2,J,2)
4540 J = J + S1: IF J > P1 THEN J = J - P1
4550 HPLLOT TO P(1,J,1),P(1,J,2)
4560 NEXT K
4570 J = 1: HCOLOR = INT (RND (1) * 2 + 5)
4580 HPLLOT P(3,1,1),P(3,1,2)
4590 FOR K = 1 TO P + 1
4600 J = J + S: IF J > P THEN J = J - P
4610 HPLLOT TO P(4,J,1),P(4,J,2)
4620 J = J + S: IF J > P THEN J = J - P
4630 HPLLOT TO P(3,J,1),P(3,J,2)
4640 NEXT K
4650 RETURN
4660 REM INSURE THAT NO LINE DUPLICATION OCCURS

```

```

4670 I = 1:J = 1
4680 I = I + 1
4690 FOR K = 1 TO 2 * N
4700 J = J + S: IF J > P THEN J = J - P
4710 NEXT K
4720 IF I < P AND J = 1 THEN P = P + 1: GOTO 4670
4730 GOSUB 500
4740 IF I < P GOTO 4680
4750 RETURN
4760 REM DEFINE OUTSIDE HOOPS
4770 IF C = 1 THEN IX = 90: IY = 274: J = 1: K = 2: Y = CY
4780 IF C = 2 THEN IX = 130: IY = 186: J = 2: K = 1: Y = CX
4790 HO = HO * IX: RO = HO * SIN (RO): IX = 5 + RO: IY = IY -
      RO
4800 L = N + 1: IA = 2 * PI / P
4810 FOR I = 1 TO P: AN = I * IA: SA = SIN (AN): CA = COS
      (AN)
4820 P(1,I,J) = IX + RO * CA: P(1,I,K) = Y + HO * SA
4830 P(L,I,J) = IY + RO * CA: P(L,I,K) = P(1,I,K)
4840 GOSUB 500
4850 NEXT I
4860 RETURN
4870 REM DEFINE INSIDE HOOP(S)
4880 PX = IY - IX: IA = 2 * PI / P
4890 IF C = 1 THEN IX = 90: IY = CX: J = 1: K = 2: Y = CY
4900 IF C = 2 THEN IX = 130: IY = CY: J = 2: K = 1: Y = CX
4910 HI = HI * IX: RI = HI * SIN (RI)
4920 IF N = 1 THEN PX = RND (1) * PX / 6 + PX / 6: IX = I
      Y - PX: IY = IY + PX: L = 3: M = 4
4930 IF N = 2 THEN IX = IY: L = 2: M = 4
4940 IF N = 3 THEN PX = (RND (1) * .8 - .4) * PX: IX = IY
      - PX: IY = IY + PX: L = 2: M = 3
4950 FOR I = 1 TO P: AN = I * IA: SA = SIN (AN): CA = COS
      (AN)
4960 P(L,I,J) = IX + RI * CA: P(L,I,K) = Y + HI * SA
4970 P(M,I,J) = IY + RI * CA: P(M,I,K) = P(L,I,K)
4980 GOSUB 500: NEXT I: RETURN
4990 REM ART FORM 1
5000 N = INT (RND (1) * 4 + 3)
5010 AN = PI / N
5020 R = (RND (1) * .2 + .3) * RM: RR = 100
5030 M = INT (RND (1) * 30 / N + 10)
5040 IY = R / M
5050 IX = (RR - R) * TAN (AN) / M
5060 REM DEFINE BASIC CURVE
5070 FOR I = 0 TO M: P(0,I,1) = 0: P(0,I,2) = I * IY: NEXT
      I
5080 P(2,0,1) = 0: P(2,0,2) = 0: P(2,1,1) = 0: P(2,1,2) = IY
5090 FOR I = 2 TO M + 10
5100 B1 = (I - 1) * IY
5110 B2 = I * IY
5120 M1 = (RR - B1) / ((I - 1) * IX)
5130 M2 = (RR - B2) / (I * IX)
5140 P(2,I,1) = (B2 - B1) / (M1 - M2)
5150 P(2,I,2) = P(2,I,1) * M1 + B1
5160 GOSUB 500
5170 NEXT I
5180 REM ROTATE AND TRANSLATE BASIC CURVE
5190 CA = COS (AN - PI): SA = SIN (AN - PI)
5200 FOR I = 0 TO M + 10
5210 X = P(2,I,1) * CA - P(2,I,2) * SA
5220 Y = P(2,I,1) * SA + P(2,I,2) * CA + RM
5230 P(2,I,1) = X: P(2,I,2) = Y
5240 GOSUB 500
5250 NEXT I
5260 REM CALCULATE END POINTS
5270 P(1,0,1) = 0: P(1,0,2) = RM
5280 FOR I = 1 TO M
5290 B1 = I * IY: M1 = (RR - B1) / (I * IX)
5300 FOR J = 1 TO M + 10
5310 IF P(2,J,1) = P(2,J - 1,1) THEN M2 = 9999: GOTO 5330
5320 M2 = (P(2,J,2) - P(2,J - 1,2)) / (P(2,J,1) - P(2,J -
      1,1))
5330 X = (P(2,J - 1,2) - P(2,J - 1,1) * M2 - B1) / (M1 - M
      2)
5340 IF X < P(2,J - 1,1) OR X > P(2,J,1) GOTO 5380
5350 P(1,I,1) = X
5360 P(1,I,2) = M1 * X + B1
5370 GOTO 5390
5380 NEXT J
5390 GOSUB 500
5400 NEXT I
5410 GOSUB 950
5420 REM GRAPH ROUTINE
5430 FOR I = 0 TO N - 1
5440 CA = COS (2 * I * AN): SA = SIN (2 * I * AN)
5450 X = 0: Y = 0
5460 GOSUB 5550
5470 NEXT I
5480 FOR I = 0 TO N - 1
5490 CA = COS (2 * I * AN + AN + PI): SA = SIN (2 * I * A
      N + AN + PI)
5500 X = - RM * SIN (2 * I * AN + AN): Y = RM * COS (2 *
      I * AN + AN)
5510 GOSUB 5550
5520 NEXT I
5530 RETURN
5540 REM LINE ROTATE, TRANSLATE, AND PLOT ROUTINE
5550 HPLLOT CX + X + P(1,0,1) * CA - P(1,0,2) * SA,CY - Y -
      P(1,0,1) * SA - P(1,0,2) * CA
5560 HPLLOT TO CX + X + P(0,0,1) * CA - P(0,0,2) * SA,CY -
      Y - P(0,0,1) * SA - P(0,0,2) * CA
5570 FOR J = 1 TO M
5580 HPLLOT CX + X + P(1,J,1) * CA - P(1,J,2) * SA,CY - Y -
      P(1,J,1) * SA - P(1,J,2) * CA
5590 HPLLOT TO CX + X + P(0,J,1) * CA - P(0,J,2) * SA,CY -
      Y - P(0,J,1) * SA - P(0,J,2) * CA
5600 HPLLOT TO CX + X - P(1,J,1) * CA - P(1,J,2) * SA,CY -
      Y + P(1,J,1) * SA - P(1,J,2) * CA
5610 GOSUB 500
5620 NEXT J
5630 RETURN

```



HARDWARE CONSTRUCTION PROJECT

Apple Game I/O Experiments

by Gary B. Little
#214-131 Water Street
Vancouver, B.C.
V6B 4M3

The Game I/O Connector on the Apple II is the socket into which the game paddles are plugged. It is located near the right-rear corner of the motherboard as you look at it from the keyboard end. Although it is referred to as a 'game' connector it can also be used for several other applications that do not involve simply monitoring the game paddle input value.

The pinout for the Game I/O Connector can be found on page 100 of the Apple II Reference Manual and is reproduced here:

+5v	1	16	NC
PB0	2	15	AN0
PB1	3	14	AN1
PB2	4	13	AN2
STROBE	5	12	AN3
GC0	6	11	GC3
GC2	7	10	GC1
Gnd	8	9	NC

Where: PB = push button input
GC = game paddle input
AN = annunciator output
NC = no connection
Gnd = electrical ground
+5v = +5 volt power supply

Note that pin 1 is located at the right front of the Game I/O Connector as viewed from the keyboard end of the Apple II.

When the game paddles are connected to the socket, only the GC0, GC1, PB0, and PB1 pins are used (apart from +5v and Gnd of course). This leaves several pins which are unused, including another push button input, two more game controller inputs, and four one-bit output ports called 'annunciator' outputs.

The purpose of this article is to describe how you can use the four annunciator outputs to drive external devices which are interfaced to the Apple II through the Game I/O Connector.

The original purpose of providing the annunciator outputs on the Apple II was apparently to allow a user to drive a series of control lamps. However, the annunciator output signals are standard 74LS series TTL outputs, so

they can also be used to drive relays, speakers and several other devices. But let's stick to what was originally intended and develop a circuit to drive lamps. In particular, let's use the annunciator outputs to turn the segments of a 7-segment LED (which are arranged in a figure-eight on the chip) on and off in specific patterns so that we can display and recognize the digits from 0 to 9.

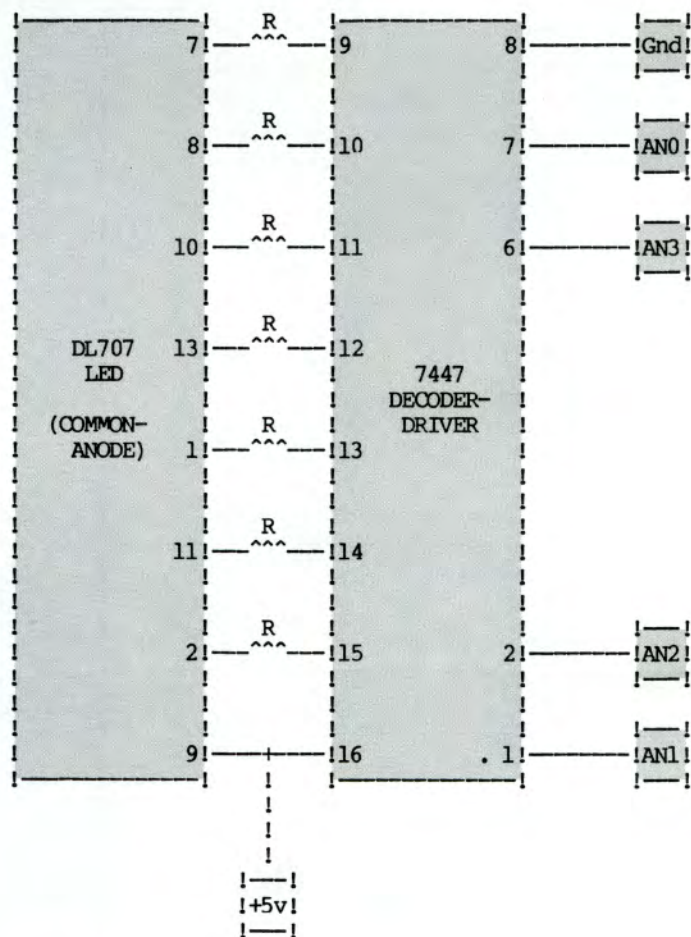
MATERIALS

To conduct this experiment you will need a few parts and materials. The parts that are required are a DL707 common-anode 7-segment LED, a 7447 BCD to 7-segment decoder-driver, and seven 330-ohm resistors. You will also require a proto-board, a few wires, and a 16-pin DIP jumper cable. The jumper cable is used to extend the Game I/O Connector from beneath the cover of the Apple II up to the proto-board so that you have room to wire the circuit. The materials can be purchased from Radio Shack with the

following part numbers:

1. LED 276-053 \$2.29
2. 7447 276-1805 \$1.19
3. Resistor 271-017 \$.19 pack of 2
4. Jumper 276-1976 \$3.99
5. Board 276-1395 \$1.39

Once you have obtained all the parts, insert one end of the jumper cable into the Game I/O Connector and the other end into the proto-board so that the longitudinal center-line of the proto-board is straddled by the two 8-pin halves of the DIP connector. The next step is to check the orientation of the DIP connector so that you can tell which pin on the DIP connector on the proto-board corresponds to which pin on the Game I/O Connector. After you have done this, connect the components on the proto-board in accordance with the circuit diagram in figure 1.



R = 330 ohms

AN0 is Pin 15 on the Game I/O Connector
AN1 is Pin 14 on the Game I/O Connector
AN2 is Pin 13 on the Game I/O Connector
AN3 is Pin 12 on the Game I/O Connector
+5v is Pin 1 on the Game I/O Connector
Gnd is Pin 8 on the Game I/O Connector

Figure 1

Apple Game I/O Experiments (Cont.)

HOW IT WORKS

The decoder-driver is a device which can take 4 TTL inputs and manipulate them to generate 7 on/off outputs, one of which is required for each of the segments on the DL707. The pattern of 7 on/off outputs depends on the pattern of the 4 TTL inputs and since there can be 16 different combinations of input, up to 16 different combinations of output can be generated which is more than enough to handle the display of 10 different digits. In our case, the 4 TTL inputs for the decoder-driver are provided by the annunciator outputs from the Game I/O Connector. The 330-ohm resistors are used to limit the current through the LED segments so that they will not burn out when the circuit is connected.

CONTROLLING THE OUTPUT

Before we try to turn the segments of the DL707 on and off let's look at how you can turn the annunciator outputs on and off in an Applesoft, Integer BASIC, or machine language program.

The four annunciator outputs on the Apple II are numbered from 0 and 3 and each is associated with a pair of memory locations in the 64K address space. These memory locations can be described as 'soft switches' because a reference to the first location in the pair will turn the annunciator off and a reference to the second location in the pair will turn the annunciator on. **When an annunciator is in the 'off' state, the voltage on its pin on the Game I/O Connector goes low (to near 0 volts) and when it is in the 'on' state, the voltage goes high (to near +5 volts).**

The memory locations needed to set the states of each of the annunciators are found in Table 9 on page 24 of the Apple II Refer-

ence Manual. Portions of that Table are reproduced here:

ADDRESS

AN#	STATE	Decimal	Hex
0	off	-16296	\$C058
	on	-16295	\$C059
1	off	-16294	\$C05A
	on	-16293	\$C05B
2	off	-16292	\$C05C
	on	-16291	\$C05D
3	off	-16290	\$C05E
	on	-16289	\$C05F

In general terms, to put **AN#N** (**N=0,1,2,3**) into the 'off' state from Applesoft or Integer BASIC, use the command **POKE -16296+2*N,0**. To put **AN#N** into the 'on' state, use the command **POKE -16295, 2*N,0**. To simplify things even further, you can add a flag to your program, say **F**, where **F=1** if you want the 'on' state and **F=0** if you want the 'off' state, so that the command **POKE -16296+2*N+F,0** will set the desired state on the desired annunciator.

MACHINE LANGUAGE CONTROL

If you want to accomplish the same thing from 6502 machine language, use the following source code in your editor/assembler:

```

LDA ANNUN ;get AN#
ASL A      ;multiply by 2
ORA STATE ;add 1 if 'on' or 0 if 'off'
TAX
STA $C058,X ;trigger the output
RTS
STATE DS 1 ;STATE is 1 if 'on'/0
           ;if 'off'
ANNUN DS 1 ;annunciator number
           ;(0,1,2,3)

```

THE LOGIC TABLE

Although we now know how to turn the annunciator outputs 'on' and 'off', we still need to know what pattern of 'ons' and 'offs' are required to display the digits from 0 to 9 on the DL707. To obtain this information we need what is called a 'logic table'. For the

particular wiring configuration found in the circuit diagram, the following logic table applies:

DIGIT	AN3	AN2	AN1	AN0
ZERO	0	0	0	0
ONE	0	0	0	1
TWO	0	0	1	0
THREE	0	0	1	1
FOUR	0	1	0	0
FIVE	0	1	0	1
SIX	0	1	1	0
SEVEN	0	1	1	1
EIGHT	1	0	0	0
NINE	1	0	0	1

In this table a '0' indicates that **AN** is 'off' and a '1' indicates that **AN** is 'on'. You can see that in this configuration the states of the four annunciators for a given digit follow the binary bit pattern of the digit. For example, the digit '7' in binary is 0111 and the corresponding annunciator sequence is exactly the same.

AN EXAMPLE

To display the digit '4', refer to the previous table to find that you must clear **AN3**, **AN1**, and **AN0** and set **AN2**. The relevant POKES to do this are as follows:

```

POKE -16296,0 (clear AN0)
POKE -16294,0 (clear AN1)
POKE -16291,0 (set AN2)
POKE -16290,0 (clear AN3)

```

The demo programs that are LISTed below are further examples of the programming techniques required to display particular digits. The **first program** repeatedly causes a random digit to be displayed on the DL707. The **second program** can be used to display any digit that is specified.

I hope you have as much fun experimenting with the Game I/O Connector as I have had. If you know of any other simple experiments for the Connector please let me know and I'll try to incorporate them into future articles.



```

100 REM RANDOM DIGIT DISPLAY
110 TEXT : HOME : PRINT "COMMON-A
    NODE LED DIGIT DISPLAY"
120 VTAB 5: CALL - 958
130 Y = INT (9 * RND (1))
140 PRINT "DIGIT = ";Y: PRINT
150 FOR I = 3 TO 0 STEP - 1
160 F = INT (Y / (2 ^ I))
170 POKE - 16296 + 2 * I + F,0
180 PRINT "ANNUNCIATOR #";I;": ";
    : IF F = 1 THEN PRINT "SET":
    GOTO 200
190 PRINT "CLEAR"
200 Y = Y - (2 ^ I) * F
210 NEXT I
220 FOR I = 1 TO 300: NEXT I
230 GOTO 120
240 END

```

```

500 REM FIXED DIGIT DISPLAY
510 TEXT : HOME : PRINT "COMMON-A
    NODE LED DIGIT DISPLAY"
520 VTAB 5: CALL - 958: INPUT "E
    NTER A NUMBER (0-9): ";Y: PRINT
    : IF Y < 0 OR Y > 9 THEN 520

530 FOR I = 3 TO 0 STEP - 1:F =
    INT (Y / (2 ^ I)): POKE - 1
    6296 + 2 * I + F,0:Y = Y -
    (2 ^ I) * F
540 PRINT "ANNUNCIATOR #";I;": ";
    : IF F = 1 THEN PRINT "SET":
    GOTO 560
550 PRINT "CLEAR"
560 NEXT I
570 PRINT : PRINT "PRESS ANY KEY
    : ";GET A$: PRINT A$: GOTO
    520

```



HI-RES DUMP 2X

by Robert M. Delaney
248D Glandore Dr.
Manchester, MO 63011

This program produces a double size dump of the Apple Hi-Res screen to an Epson MX-80 printer equipped with GRAFTRAX. Because of its width, the picture is turned 90 degrees for the printout.

The method used is to replace every dot on the Hi-Res screen by a pattern of four dots printed in a square. The area of the picture is increased by a factor of four, but the relative intensities remain the same.

Since a BASIC program would be too slow, a machine language program, **HIRES DUMP 2X**, does most of the work. An Assembly Language listing of the program accompanies this article. The settings of the dip switches on your Epson may require the program to give a Line Feed (LF) following a Carriage Return (CR). This is the default on the listed program. If you do not require the Line Feed, change address location **\$6001** to **\$00** when entering the program.

I use the machine language program by calling it from the following BASIC program, which does some necessary initializations.

```
10 REM HIRES DUMP 2X.BASIC
15 D$=CHR$(4):I$=CHR$(9):REM
   CONTROL-D AND CONTROL-I
20 PRINT D$;"BLOAD HIRES DUMP 2X":
   REM A$6000,L$10E
25 HOME:VTAB 12:INPUT "TURN ON
   MX-80 AND PRESS RETURN":A$
30 HOME:VTAB 12:HR=32:INPUT "HIRES
   PAGE 1 OR 2?":A$:IF A$="2" THEN
   HR=64
35 HOME:VTAB 12:NI=255:INPUT
   "NORMAL (N) OR INVERSE (I)?":A$:IF
   A$="I" THEN NI=0
40 HOME:VTAB 12:TB=8:INPUT "TAB
   (1-15) = ? (DEFAULT=8 - CENTERED)":
   A$
45 IF A$<>" " THEN TB=VAL(A$):IF
   TB<1 OR TB>15 THEN 40
50 POKE 27,NI:POKE 23,HR
55 PRINT D$;"PR#1"
60 PRINT I$;"80N"
65 PRINT CHR$(27)CHR$(61):REM
   CLEAR MOST SIG BIT
70 PRINT CHR$(27)CHR$(68)CHR$(TB)
   CHR$(0):REM SET TAB
75 PRINT CHR$(27)CHR$(65)CHR$(7):
   REM SET 7/72 INCH LINE SPACING
80 CALL 24576:REM CALL THE MACHINE
   LANGUAGE PROGRAM AT $6000
85 PRINT CHR$(27)CHR$(64):REM RESET
   PRINTER TO DEFAULT SETTINGS
90 PRINT I$;"O"
95 PRINT D$;"PR#0"
96 END
```

See the letters column for information on loading and saving the machine language program HIRES DUMP 2X.

The essential feature of the machine language program is its translation of a Hi-Res Screen Byte into 4 Hi-Res Printer Bytes. To understand this, you should know how a Screen Byte maps to the video screen and how a Printer Byte maps to the needles in the Epson print head.

```
SOURCE FILE: HIRES DUMP 2X
0000: 1 *****
0000: 2 **
0000: 3 ** HIRES DUMP 2X **
0000: 4 ** BY ROBERT M. DELANEY **
0000: 5 **
0000: 6 ** COPYRIGHT (C) 1982 BY **
0000: 7 ** MICRO-SPARC, INC. **
0000: 8 ** APPLE TOOLKIT ASSEMBLER **
0000: 9 **
0000: 10 *****
----- NEXT OBJECT FILE NAME IS HIRES DUMP 2X
6000: 11 ORG $6000
0000: 12 ADDLO EQU $00 HIRES ADDRESS LO
0001: 13 ADDHI EQU $01 HIRES ADDRESS HI
0016: 14 COUNT EQU $16 COUNTER FOR BYTES IN SCREEN LINE
0017: 15 PAGE EQU $17 HIRES PAGE 1 OR 2
0018: 16 HI1 EQU $18 PRINTER BYTE 1
0019: 17 HI2 EQU $19 PRINTER BYTE 2
001A: 18 LINE EQU $1A LINE 0-191 ON SCREEN
001B: 19 NORINV EQU $1B NORMAL (=FF) OR INVERSE (=00) BYTE
001C: 20 DOLF EQU $1C CONTROLS LINE FEED $FF=YES $00=NO
C090: 21 OUT EQU $C090 OUT TO PRINTER
C1C1: 22 TAKEN EQU $C1C1 PLUS WHEN DATA TAKEN
6000: 23 *
6000: 24 * MAIN PROGRAM
6000: 25 *
6000:A9 FF 26 LDA #$FF **FF-LINE FEED **00=NO LINE FEED
6002:85 1C 27 STA DOLF
6004:A9 00 28 LDA $00
6006:85 16 29 STA COUNT INITIALIZE COUNTER
6008:20 F0 60 30 AGAIN JSR BITIMAG SET UP BIT IMAGE GRAPHICS
600B:20 A7 60 31 ODD JSR APLINE CONVERT SCREEN LINE # TO APPLE FORMAT
600E:A4 16 32 LDY COUNT OFFSET TO NEXT BYTE IN SCREEN LINE
6010: 33 *
6010: 34 * ROUTINE TO CONVERT SCREEN BYTE TO PRINTER BYTE
6010: 35 *
6010:B1 00 36 LDA (ADDLO),Y GET SCREEN BYTE
6012:85 18 37 STA HI1
6014:6A 38 ROR A
6015:26 19 39 ROL HI2
6017:66 18 40 ROR HI1
6019:26 19 41 ROL HI2
601B:6A 42 ROR A
601C:26 19 43 ROL HI2
601E:66 18 44 ROR HI1
6020:26 19 45 ROL HI2
6022:6A 46 ROR A
6023:26 19 47 ROL HI2
6025:66 18 48 ROR HI1
6027:26 19 49 ROL HI2
6029:6A 50 ROR A
602A:26 19 51 ROL HI2
602C:A5 19 52 LDA HI2
602E:45 1B 53 EOR NORINV ADJUST FOR NORMAL OR INVERSE GRAPHICS
6030:29 7F 54 AND #$7F MAKE SURE MSBIT IS ZERO
6032:85 18 55 STA HI1
6034:20 D8 60 56 JSR COUT PRINT 7 VERTICAL DOTS
6037:A5 1A 57 LDA LINE
6039:F0 05 58 BEQ NEXT NEXT PRINT LINE IF SCREEN LINE ZERO
603B:A5 18 59 LDA HI1
603D:20 D8 60 60 JSR COUT PRINT DUPLICATE 7 VERTICAL DOTS
6040:A5 1A 61 NEXT LDA LINE
6042:38 62 SEC
6043:E9 01 63 SBC #$01
6045:85 1A 64 STA LINE DEC LINE
6047:B0 C2 65 BCS ODD
6049:20 E1 60 66 JSR CRLF CR WITH OPTIONAL LF
604C:20 F0 60 67 JSR BITIMAG SET UP BIT IMAGE GRAPHICS
604F:20 A7 60 68 EVEN JSR APLINE
6052:A4 16 69 LDY COUNT
6054: 70 *
6054: 71 * 2ND ROUTINE TO CONVERT SCREEN BYTE TO PRINTER BYTE
6054: 72 *
6054:B1 00 73 LDA (ADDLO),Y GET SCREEN BYTE
6056:85 18 74 STA HI1
6058:6A 75 ROR A
6059:66 18 76 ROR HI1
605B:6A 77 ROR A
605C:66 18 78 ROR HI1
605E:6A 79 ROR A
605F:66 18 80 ROR HI1
6061:6A 81 ROR A
6062:66 18 82 ROR HI1
6064:26 19 83 ROL HI2
6066:6A 84 ROR A
6067:26 19 85 ROL HI2
6069:66 18 86 ROR HI1
606B:26 19 87 ROL HI2
606D:6A 88 ROR A
606E:26 19 89 ROL HI2
6070:66 18 90 ROR HI1
6072:26 19 91 ROL HI2
6074:6A 92 ROR A
6075:26 19 93 ROL HI2
6077:66 18 94 ROR HI1
6079:26 19 95 ROL HI2
607B:A5 19 96 LDA HI2
607D:45 1B 97 EOR NORINV ADJUST FOR NORMAL OR INVERSE GRAPHICS
607F:29 7F 98 AND #$7F MAKE SURE MSBIT IS ZERO
6081:85 18 99 STA HI1
6083:20 D8 60 100 JSR COUT
6086:A5 1A 101 LDA LINE
6088:F0 05 102 BEQ NEXT1
608A:A5 18 103 LDA HI1
608C:20 D8 60 104 JSR COUT
608F:A5 1A 105 NEXT1 LDA LINE
6091:38 106 SEC
6092:E9 01 107 SBC #$01
6094:85 1A 108 STA LINE
6096:B0 B7 109 BCS EVEN
609B:20 E1 60 110 JSR CRLF CR WITH OPTIONAL LF
609B:E6 16 111 INC COUNT
609D:A5 16 112 LDA COUNT
609F:C9 28 113 CMP #$2B HAVE DEC 40 SCREEN BYTES BEEN USED?
```



```

60A1:F0 03 114 BEQ RETURN
60A3:4C 0B 60 115 JMP AGAIN IF NOT, DO AGAIN
60A6:60 116 RETURN RTS RETURN TO CALLING PROGRAM
60A7: 117 *
60A7: 118 * SUB TO CONVERT SCREEN LINE NO. TO APPLE FORMAT
60A7: 119 *
60A7:A5 1A 120 APLINE LDA LINE
60A9:85 01 121 STA ADDHI
60AB:2A 122 ROL A
60AC:26 00 123 ROL ADDLO
60AE:2A 124 ROL A
60AF:26 00 125 ROL ADDLO
60B1:2A 126 ROL A
60B2:26 01 127 ROL ADDHI
60B4:26 00 128 ROL ADDLO
60B6:2A 129 ROL A
60B7:26 01 130 ROL ADDHI
60B9:26 00 131 ROL ADDLO
60BB:2A 132 ROL A
60BC:26 01 133 ROL ADDHI
60BE:A5 01 134 LDA ADDHI
60C0:29 3F 135 AND #$3F
60C2:85 01 136 STA ADDHI
60C4:A5 00 137 LDA ADDLO
60C6:0A 138 ASL A
60C7:0A 139 ASL A
60C8:0A 140 ASL A
60C9:0A 141 ASL A
60CA:18 142 CLC
60CB:66 01 143 ROR ADDHI
60CD:6A 144 ROR A
60CE:85 00 145 STA ADDLO
60D0:A5 01 146 LDA ADDHI
60D2:18 147 CLC
60D3:65 17 148 ADC PAGE ADJUST FOR HIRES PAGE 1 OR 2
60D5:85 01 149 STA ADDHI

```

```

60D7:60 150 RTS
60D8: 151 *
60D8: 152 * SUB TO OUTPUT TO PRINTER
60D8: 153 *
60D8:8D 90 C0 154 COUT STA OUT
60DB:AD C1 C1 155 UNTIL LDA TAKEN
60DE:30 FB 156 BMI UNTIL
60E0:60 157 RTS
60E1: 158 *
60E1: 159 * SUB FOR CR WITH OPTIONAL LF
60E1: 160 *
60E1:A9 0D 161 CRLF LDA #$0D CARRIAGE RETURN
60E3:20 D8 60 162 JSR COUT
60E6:A5 1C 163 LDA DOLF
60E8:F0 05 164 BEQ NOLF NO LINE FEED
60EA:A9 0A 165 LDA #$0A
60EC:20 D8 60 166 JSR COUT LINE FEED
60EF:60 167 NOLF RTS
60F0: 168 *
60F0: 169 * SUB TO SET UP BIT IMAGE GRAPHICS
60F0: 170 *
60F0:A9 09 171 BITIMAG LDA #$09 TAB
60F2:20 D8 60 172 JSR COUT
60F5:A9 1B 173 LDA #$1B
60F7:20 D8 60 174 JSR COUT ESCAPE
60FA:A9 4B 175 LDA #$4B
60FC:20 D8 60 176 JSR COUT
60FF:A9 7F 177 LDA #$7F
6101:20 D8 60 178 JSR COUT
6104:A9 01 179 LDA #$01
6106:20 D8 60 180 JSR COUT SET UP FOR 383 COLUMNS OF DOTS
6109:A9 BF 181 LDA #$BF
610B:85 1A 182 STA LINE INITIALIZE SCREEN LINE TO DECIMAL 191
610D:60 183 RTS

```

*** SUCCESSFUL ASSEMBLY: NO ERRORS

A Hi-Res Screen Byte expressed in normal order appears as:

BIT	7	6	5	4	3	2	1	0
	x	g	f	e	d	c	b	a

Bit 7, which controls the Hi-Res color, is for our purposes a "don't care". Any of the other bits when set to 1 give a bright dot, and when set to 0 give a dark dot. The bits map to the screen in reverse order as:

BIT	0	1	2	3	4	5	6
	a	b	c	d	e	f	g

SCREEN BIT MAP

The Epson print head consists of nine needles arranged in a vertical column. Only the middle seven needles can be controlled by the Apple. In the Bit Image Graphics mode, bit 0 of a byte sent to the Epson will affect the bottom needle of the seven, bit 1 the next needle, and so on, with bit 6 affecting the top needle of the seven. Bit 7 is a "don't care". Thus we have the mapping:

Hi-Res Printer Byte	Print Head Needles
BIT	
	• (top - not used)
6	•
5	•
4	•
3	•
2	•
1	•
0	•
	• (bottom - not used)

If a bit is 1 it "fires" the corresponding needle, and a dot is printed. If a bit is 0 no dot results.

Now think of rotating the SCREEN BIT MAP one-quarter turn clockwise, so it is vertical:

Next, map each bit into four identical bits forming a square:

Hi-Res Printer Byte

BIT

6	a	a
5	a	a
4	b	b
3	b	b
2	c	c
1	c	c
0	d	d
6	d	d
5	e	e
4	e	e
3	f	f
2	f	f
1	g	g
0	g	g

ODD PRINT LINE

EVEN PRINT LINE

We see that every Hi-Res Screen Byte maps into 4 Hi-Res Printer Bytes. However, because of the horizontal duplication, there are just 2 distinct types. If we number our print lines 1, 2, ..., we can designate these types as odd or even. In the assembly language listing, the "ROUTINE TO CONVERT SCREEN BYTE TO PRINTER BYTE" converts the odd line bytes. The "2ND ROUTINE TO CONVERT SCREEN BYTE TO PRINTER BYTE" converts the even line bytes.

That's really most of the story. When the machine language program is called by the BASIC program it performs necessary initializations, then sets up Bit Image Graphics for the first print line. The first byte of screen line 191 is converted by the "odd" routine to a Hi-Res Printer Byte, which is then printed twice. The same is done for the first byte of line 190 and so on to line 0.

After a CR, the first byte of screen line 191 is converted by the "even" routine to a Hi-Res Printer Byte, which is then printed twice. This is repeated for the rest of the lines through line 0.

Then the second bytes of all the lines are treated exactly the same way as the first bytes were, and so on for all 40 bytes which are involved with each line of the Hi-Res Screen.

Since APPLE II will not pass the bytes \$09 or \$0C to the printer correctly, these bytes must be transmitted directly. For the interface card in Slot 1, the memory location which transmits to the printer is \$C090, and the location that tells when the data is taken is \$C1C1. Thus I created my own COUT routine, using these addresses, for passing all bytes to the printer.

If an interface card is in another slot, or if it uses different memory locations than \$C090 or \$C1C1 in Slot 1, then it will not work with my program unless the variables OUT and TAKEN in the Assembly Language Program are redefined.

There is one flaw in the program. Since the Apple Hi-Res screen has 192 lines, there should be $2 \times 192 = 384$ dots in a line on the printout. Because the Apple only sends seven bits to the printer, a maximum of 383 dots can be printed at one time in Epson's Bit Image Graphics. Thus every line on the Hi-Res screen is doubled in the printout except for the topmost line, which is printed only once. Since there is usually little information in the top line, I have found this to be no problem.

Super Numeric Keypad

by Alan B. Cohen
14 Candlelight Dr.
Danbury CT 06810

This program enables the user to redefine a section of the Apple II or II Plus keyboard to function as a numeric keypad. The keypad can be turned on and off under software control. Either RAM or ROM Applesoft and Integer Basic can be used, as well as machine language. Both regular monitor and autostart monitor can be used. It should also work with any Disk Operating System that latches into the input and output routines so long as **no disk I/O is attempted while the keypad is active**. I have no information about compatibility with Pascal, Fortran, or other languages.

USING THE KEYPAD

The keypad is turned on by "CALL 768" from either Basic or "JSR \$300" from an assembler. It is turned off by "CALL 785" or "JSR \$311". While active, the program will accept input from designated keys only; other keystrokes will be answered by a beep from the speaker. Left and right arrows, (RETURN), (REPEAT), and (RESET) are the only other keys that will function normally. (See figure 1 for keypad layout.) The input characters can be assigned to any type of variable; integer, real, or string.

The changes in the host program are generally minimal. For example, if a line was originally coded as:

300 INPUT "ENTER THE ZIP CODE";Z

as modified, it would read:

300 CALL 768: INPUT "ENTER THE ZIP CODE";Z:CALL 785

HOW IT WORKS

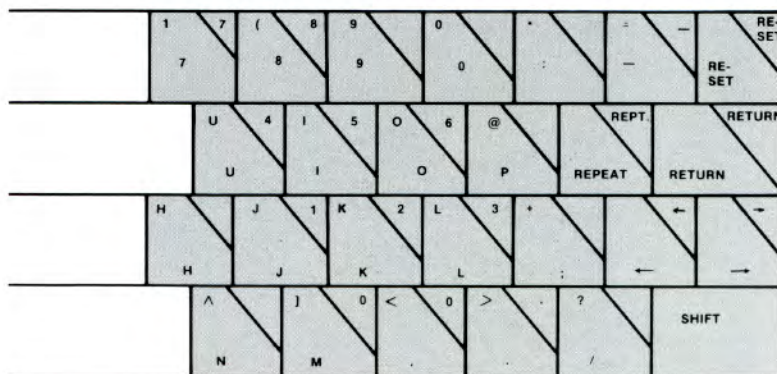
The listing for the program is divided into five sections:

- 1-9 Header, Label assignments, and brief instructions
- 22-43 Routines to turn the keypad on and off
- 45-62 Keypad Routine
- 64-70 Input Keystroke table
- 71-76 Output Character table

The first section has no executable code and exists only to label the listing, assign labels to the five zero page locations used, assign labels to the entry points for the two monitor routines used, and note to the turn on and off commands for the keypad. The second section, lines 22 through 43, contains the "on" and "off" routines; Lines 22 through 27 store the previous input routine hooks, and reinsert them at KSWL and KSWL+1.

The third section, lines 45 through 62, is the heart of the program. It accepts keystrokes, determines if they are valid for the keypad, and outputs the appropriate character for valid keystrokes. Line 47 uses a monitor routine to access the keyboard and get the next keystroke. This routine leaves the ASCII value of the keystroke in the accumulator; line 48 stores the keystroke in "TEMP" for later use. Lines 49 through 50 store the Y register in "YSAV" and set the Y register equal to zero. The Y register is used as a counter. Line 51

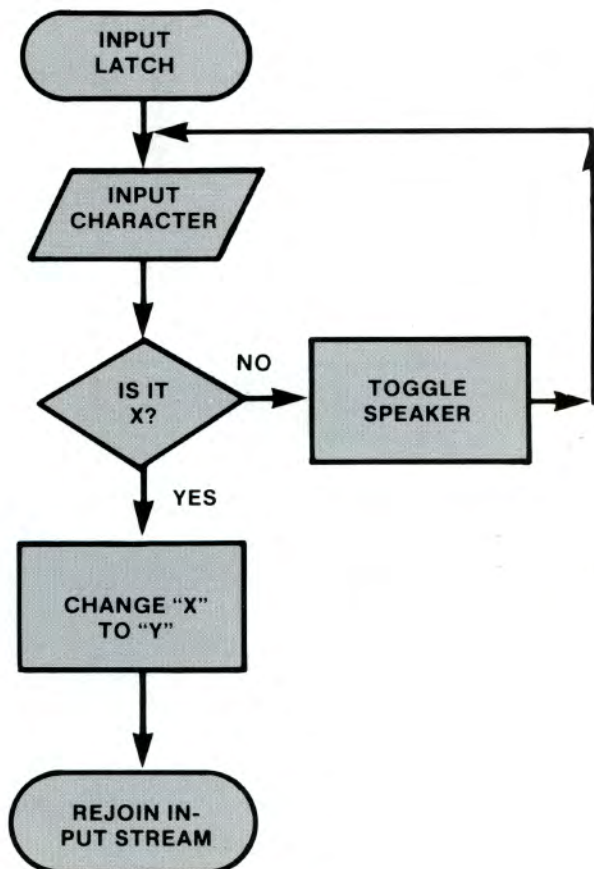
Figure 1. Keypad Layout



NOTE: CHARACTER IN UPPER RIGHT CORNER IS RETURNED WHEN KEY IS PRESSED UNSHIFTED.

Figure 2. Program Logic

x	y
<cr>	<cr>
.	0
,	0
M	1
J	2
K	3
L	4
U	5
I	6
O	7
7	7
8	8
9	9
→	→
←	←
—	—



gets the next value of the input table and stores it in the accumulator. Line 52 tests if the value from the INPUT table (now in the accumulator) is equal to the key pressed (now in TEMP). If they are not equal, the routine skips to Line 57. Lines 54-56 get the replacement value from the output table, restore the old Y register, and return to the normal input routines in the Monitor. Lines 57-59 are accessed only if a match has not yet been found. Line 57 increments the Y register and line 58 checks to see if the end of the input table has been reached. As written, the input table is 16 bytes (\$10) long, begins at location \$343, and ends at \$352. When the Y register value reaches \$10, the program would be looking for the input character at location \$353 (\$343 + \$10), which is beyond the end of the table. As long as the Y register does not equal \$10, the program tries the next item in the table. If the end of the table has been reached, lines 60-62 are executed. These lines beep the speaker and return to line 47 to get another keystroke. The input and output tables at lines 66-76 contain the valid keystrokes and the corresponding replacement characters. See figure 2 for a flowchart of program logic.

SAVING AND LOADING THE PROGRAM

The program, including data tables, can be saved on cassette by the following procedure:

- enter the monitor by either "CALL-151" or pressing the (RESET) key (on non-Autostart-equipped Apples).
- type "300.362W"
- start recording and press the (RETURN) key.

Once saved, the program can be loaded from tape by:

- enter the monitor (as above).
- type "300.362R"
- start playing and press the (RETURN) key.

To save the program on disk, type **BSAVE NUMPAD, A\$300, L\$63**.

LIMITATIONS

The program has one "bug" which, while not interfering with the functioning of the program, should be noted: after the second sequential invalid keystroke, the key pressed will appear in the first print position of the current line.

POTENTIAL APPLICATIONS

The keypad can be used to enter long lists of numbers as encountered in calculation of statistical measures such as mean and standard deviation. Or, expanded to include additional keys, the keypad can be used to support a calculator program. Expansion is easily accomplished by three steps:

1. Add hex ASCII value of input keystroke to the input table.
2. Add hex ASCII value of output character to the output table.
3. Change the "CPY" value on line 610 to the hex length of the input table.

I would appreciate hearing from anyone who makes use of the keypad concept in a common (or uncommon) application. The assembly listing was produced using the Apple DOS Toolkit Assembler.



```

1 REM *****
2 REM * SUPER NUMERIC KEYPAD *
3 REM * DEMO *
4 REM * BY ALAN B. COHEN *
5 REM * COPYRIGHT (C) 1982 *
6 REM * BY MICRO-SPARC INC *
7 REM * LINCOLN, MA. 01773 *
8 REM * ALL RIGHTS RESERVED *
9 REM *****
10 REM NUMPAD DEMO
15 REM IN APPLESOFT
20 TEXT : HOME
25 PRINT CHR$ (4);"BLOAD NUMPAD"
30 PRINT "PLEASE ENTER TWO NUMBERS, SEPARATED": PRINT
   "BY A <CR>". THE KEYPAD IS ON."
40 CALL 768: INPUT N1$: INPUT N2$: CALL 789
50 N1 = VAL (N1$):N2 = VAL (N2$)
65 PRINT "KEYPAD OFF"
70 VTAB 10: PRINT "THANK YOU. THE NUMBERS YOU ENTERED": PRINT "ARE: "
80 PRINT "AS STRINGS ";N1$;" AND ";N2$
90 PRINT "AS NUMBERS ";N1;" AND ";N2
100 PRINT "THE SUM OF THESE TWO NUMBERS IS ": PRINT N
   1 + N2
110 PRINT "THE TWO STRINGS APPENDED ARE ": PRINT N1$ +
   N2$
120 PRINT "YOU DON'T HAVE TO INPUT AS STRINGS,": PRINT
   "THIS WAS JUST TO SHOW YOU IT COULD": PRINT "BE DONE. BYE."
130 END

SOURCE FILE: NUMPAD SOURCE 2.1
0000: 1 ;*****
0000: 2 ; SUPER NUMERIC KEYPAD
0000: 3 ; BY ALAN B. COHEN
0000: 4 ; COPYRIGHT (C) 1982
0000: 5 ; BY MICRO-SPARC, INC.
0000: 6 ; LINCOLN, MA 01773
0000: 7 ; ALL RIGHTS RESERVED
0000: 8 ;*****
0000: 9 ;
----- NEXT OBJECT FILE NAME IS NUMPAD SOURCE 2.1.OBJ0
0300: 10 ORG $300
0006: 11 KLSAV EQU $06 ;INSTRUCTIONS:
0007: 12 KHSAB EQU $07 ;TO TURN ON:
0008: 13 TEMP EQU $08 ;JSR $300 / CALL 768
0034: 14 YSAV EQU $34 ;TO TURN OFF:
0038: 15 KSWL EQU $38 ;JSR $315 / CALL 789
FD18: 16 KEYIN EQU $FD18 ;WORKS WITH BOTH
FF3A: 17 BELL EQU $FF3A ;APPLESOFT & INTEGER
AB51: 18 DOS EQU $AB51
AA55: 19 DOSKL EQU $AA55
0300: 20 ;
0300: 21 ;
0300: 22 ; STORE INITIAL HOOKS
0300: 23 ;
0300:AD 55 AA 24 LDA DOSKL
0303:85 06 25 STA KLSAV
0305:AD 56 AA 26 LDA DOSKL+1
0308:85 07 27 STA KHSAB

```

```

030A: 28 ;
030A: 29 ; SET HOOKS TO KEYPAD START
030A: 30 ;
030A:A9 20 31 LDA $>START
030C:85 38 32 STA KSWL
030E:A9 03 33 LDA $<START
0310:85 39 34 STA KSWL+1
0312:4C 51 A8 35 JMP DOS ;SAVE DOS LATCHES

0315: 36 ;
0315: 37 ; RESET HOOKS TO SHUT OFF KEYPAD
0315: 38 ;
0315:A5 06 39 LDA KLSAV
0317:85 38 40 STA KSWL
0319:A5 07 41 LDA KHSAB
031B:85 39 42 STA KSWL+1
031D:4C 51 A8 43 JMP DOS
0320: 44 ;
0320: 45 ; MAIN KEYPAD ROUTINE
0320: 46 ;
0320:20 1B FD 47 START JSR KEYIN ;GET KEY
0323:85 08 48 STA TEMP ;STORE KEY
0325:84 34 49 STY YSAV ;STORE YREGISTER
0327:A0 00 50 LDY #00 ;ZERO COUNTER
0329:B9 43 03 51 LOOP LDA INPUT,Y ;LOAD YTH ITEM
032C:C5 08 52 CMP TEMP ;SAME?

032E:D0 06 53 BNE NEXT ;NO, NEXT ITEM
0330:B9 53 03 54 LDA OUTPUT,Y ;YES,GET REPLACEMENT
0333:A4 34 55 LDY YSAV ;RESTORE YREGISTER
0335:60 56 RTS ;CONTINUE INPUT RTNE
0336:C8 57 INY ;INCREMENT COUNTER
0337:C0 10 58 CPY #$10 ;END OF TEST LIST?

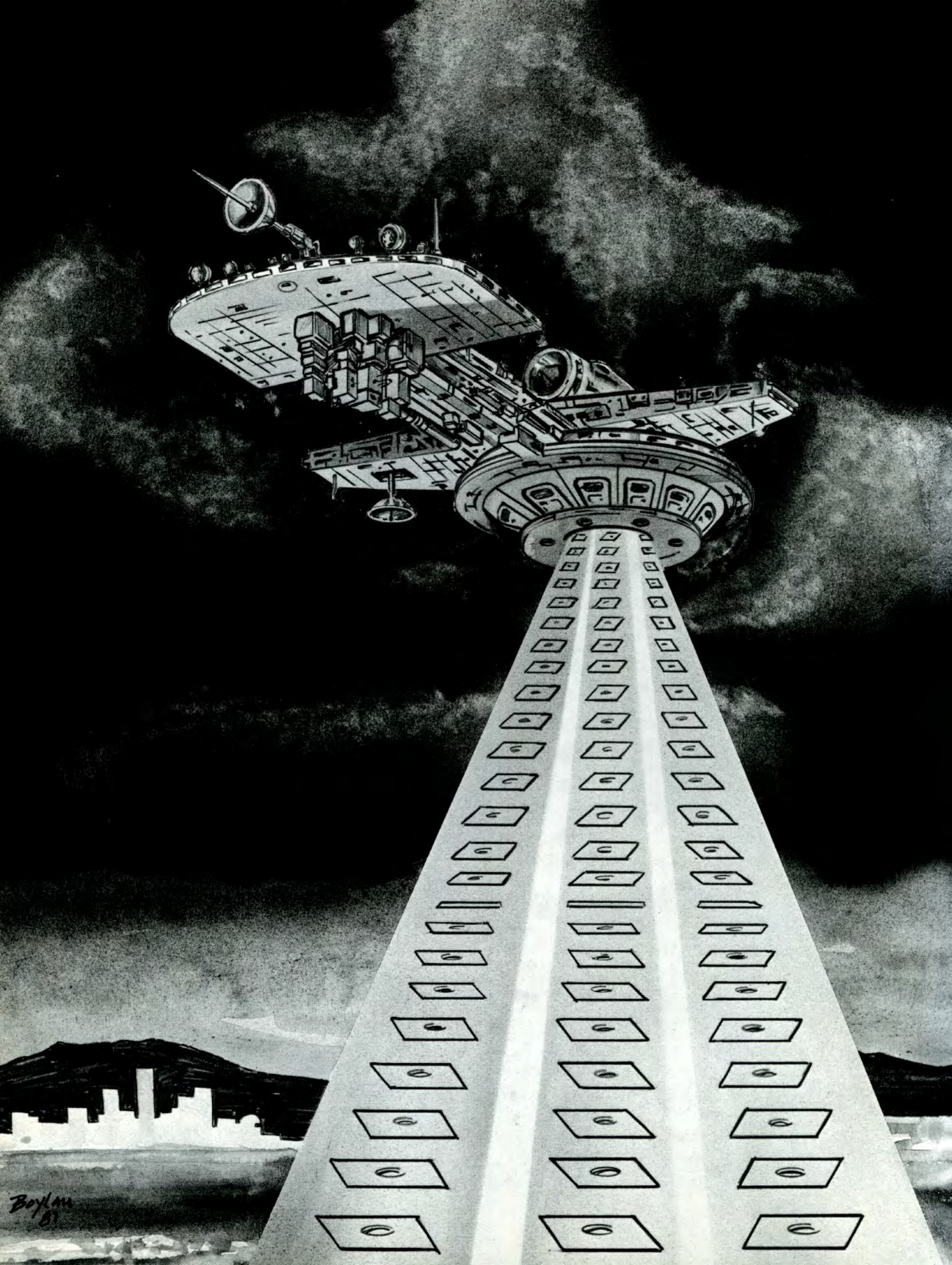
0339:D0 EE 59 BNE LOOP ;NO,TEST NEXT ITEM
033B:20 3A FF 60 JSR BELL ;YES,BEEP!!
033E:A4 34 61 LDY YSAV ;RESTORE YREGISTER
0340:4C 20 03 62 JMP START ;TRY AGAIN

0343: 63 ;
0343: 64 ; VALID INPUT KEYSTROKES
0343: 65 ;
0343:8D AE AC 66 INPUT DFB $8D,$AE,$AC,$CD,$CA
0346:CD CA 67 DFB $CB,$CC,$D5,$C9,$CF
0348:CB CC D5 68 DFB $B7,$B8,$B9,$B8,$95
034B:C9 CF 69 DFB $AD
034D:B7 B8 B9 70 ;
0350:88 95 71 ; OUTPUT KEYCODE TABLE
0352:AD 72 ;
0353: 73 OUTPUT DFB $8D,$AE,$B0,$B0,$B1
0356:B0 B1 74 DFB $B2,$B3,$B4,$B5,$B6
0358:B2 B3 B4 75 DFB $B7,$B8,$B9,$B8,$95
035D:B7 B8 B9 76 DFB $AD
0360:88 95 77 ;
0362:AD 78 ; THE END
0363: 79 ;
0363: 79 ;

```

*** SUCCESSFUL ASSEMBLY: NO ERRORS





Boylan
01

DISK COMMANDER

Disk COMMANDER

by Preston R. Black, M.D.
16 Durham St.
Boston, MA 02115

Recently, while organizing my diskettes, I noticed that many of the files I use most often seem to have names like **MY FASTEST SORTING ROUTINE** or **SUPER-DUPER DISKETTE UTILITY**. I also realized that I was just taking advantage of the ability of the Apple II Disk Operating System (DOS) to handle file names up to thirty characters long. With this inherent flexibility, I frequently try to give my programs descriptive names which hopefully will give anyone using the program an idea of what the program was written to do. Unfortunately, long names can be cumbersome to use and for those of us who are not expert typists, typing file names that are thirty characters long can become very frustrating.

Most Apple II users soon discover that one way to circumvent their lack of typing skills is to perform a CATALOG and use the Apple's screen editing capabilities, i.e. the right arrow (→), the left arrow (←), and the **REPEAT** key, to enter the file name. But more often than not the list of files, at least on my most frequently used diskettes, exceeds the capacity of the text screen, causing half of the Diskette Directory to scroll off the screen during the Catalog. As luck would have it, the file we are interested in is usually among those scrolling off the text screen. Moreover, very few key strokes are saved using this technique.

Obviously what we need is a program which will allow us to use descriptive (i.e. long) file names while also giving us the ability to use functional (i.e. short) representations of the files to access them.

The diskette utility DISK COMMANDER is just what the doctor ordered. First, DISK COMMANDER **sequentially numbers the files** in the Diskette Directory. Then by using the number assigned to a given file, the various DOS functions can be performed on that file. To reduce typing even further, **only the first three characters of any DOS command are required for proper function**. DISK COMMANDER also supports **four commands** used only by the utility.

"C" — Print the numbered DISK COMMANDER catalog

"Dx" — Change the disk drive number to x

"Sx" — Change the slot number to x

"Q" — Quit DISK COMMANDER

With these features DISK COMMANDER makes accessing files with names of any length quick and efficient.

HOW DISK COMMANDER WORKS

DISK COMMANDER is divided into four parts:

1. The Initializer (lines 76-93)
2. The Command Processor (lines 98-447)
3. The Directory Reader (lines 454-526)
4. The Catalog Printer (lines 531-638)

These four modules can be thought of as four independent programs which are linked together to perform a unified task.

INITIALIZER

The Initializer performs **four tasks**. First, it resets **HIMEM** to hide DISK COMMANDER and thereby prevent the utility from being overwritten. **Secondly**, this module resets the input hooks at **KSWL (\$38)** and **KSWH (\$39)** to point to the short routine at **EASE?**. This routine is responsible for the **third** function of the Initializer, namely to scan the keyboard for a **control-F**, the character used to initialize DISK COMMANDER. If any other key is entered, the normal monitor input and output routines are used. However, when a **control-F** is detected the program jumps to **EASE** where the fourth and final function is performed. **EASE** reads the Diskette Directory into RAM, outputs a percent sign, a new prompt signaling that DISK COMMANDER is in operation, and then falls into the Command Processor.

COMMAND PROCESSOR

The heart of DISK COMMANDER is the Command Processor. This module handles the input and interpretation of the legal commands as well as checking for errors. The monitor routine **GETLN1 (\$FD6F)** is used for entering commands. This routine stores all characters entered from the keyboard into a buffer beginning at **\$200**. Since a **RETURN** delimits the string being entered, on exit from **GETLN1** the **X-register** holds the length of the character string **plus one**. DISK COMMANDER is designed to recognize short commands and therefore the space for command storage has been limited. If a command is overly long, an error message is printed.

The routine initially looks at the first character of the input string searching for the special DISK COMMANDER commands. If any of these commands is detected, the specific operation is performed. In addition, a search is made for the **BSAVE** and **RENAME** commands because they need special attention. **BSAVE** and **RENAME** are the only two DOS commands which **require information in addition to the command** and the file name to function properly. Consequently provision must be made to provide this information. For example, **BSAVE** needs the starting address and the length of the binary file to be saved. **RENAME** requires the name to which the file is being changed. When DISK COMMANDER detects that either of these commands is being issued, it jumps to the proper routine where a string containing the required information is constructed.

DISK COMMANDER now begins to process the command that the user has entered. All DISK COMMANDER commands are divided into **two fields**: the **command field** and the **number field**. **These two fields must be separated by a space**. This character is used by DISK COMMANDER to define and demarcate the two fields. The processor now places the two fields into separate buffers in preparation for interpretation. Before the number field is placed into its buffer, a check is made to make certain that all the entries are indeed numeric.

COMMAND FIELD

The command field is processed first. A table of the legal three letter commands used by DISK COMMANDER is stored at **CMDLST**. The string in the command buffer is compared to the entries in the command list to see

if a match can be made. If there is no match, an error message is printed. If a match is found, a pointer to the full length command (found in another table) is stored for later use.

NUMBER FIELD

The interpretation of the **number field** begins by converting each character in that field from its ASCII representation into its hexadecimal equivalent. This number is used in conjunction with the table of addresses pointing to the files in the diskette directory to indicate the file we wish to use. This address is also stored for later use.

CONSTRUCTING COMMANDS

The Command Processor then constructs the command string which will then be used to issue the command. The command string begins with a **RETURN (\$8D)** to make certain that the command is issued on a new line. Following this is a **control-D (\$84)** indicating that a DOS command is about to follow. The **RETURN** is needed because the **control-D must be the first character on the line to function properly**. The processor now uses the previously stored pointer to locate the full DOS command which is then placed into the command string. The name of the file is similarly added to the command string.

If **BSAVE** or **RENAME** is the command, then the string holding the necessary additional information is also added to the command string.

Finally, another **RETURN** and the end of string delimiter (**\$00**) is added to complete the command string. The entire string is then output using the monitor output routine at **COU (\$FDED)**, thereby passing the command to DOS to be performed.

DIRECTORY READER

The Directory Reader uses the **RWTS** subroutine to read the Diskette Directory and stores it in a RAM buffer. In addition a table is constructed which contains the addresses of the start of each file entry in the buffer. Access to and manipulation of each file is thereby made much easier.

CATALOG PRINTER

The Catalog Printer is used to print the special DISK COMMANDER catalog. Before each file is printed, it is assigned a special number which is then used to access each file. This number replaces the number used by the DOS CATALOG to indicate how many sectors are needed to save the file. The file type, whether the file is locked or unlocked, and the file name follows.

HOW TO USE DISK COMMANDER

DISK COMMANDER is about 3.5K long including space reserved for storage. In a 48K system it should be assembled starting at **\$8200 (Hex)**. For smaller systems appropriate adjustments should be made. See the instructions for entering machine language programs into memory in the letters section of this issue. When the program has been completely entered, you can save it to disk with the command:

BSAVE DISK COMMANDER,A\$8200,L\$4AC

continued on next page

Disk COMMANDER (Cont.)

Once the program has been BSAVed, BRUNning DISK COMMANDER will reset HIMEM to protect the program and reset the monitor hooks to make the program operative. Thereafter entering control-F will put the user into **DISK COMMANDER** indicated by the new prompt, the **percent sign (%)**. Now all the special DISK COMMANDER commands are operational.

SOME RULES

Several rules must be observed when using DISK COMMANDER. You can press 'Q' to quit and return to BASIC. The "C" command will cause the special catalog to be printed. Like the DOS CATALOG command, the listing will pause after a proscribed number of files have been listed. Pressing the space bar will continue the listing of both catalogs. But unlike the DOS CATALOG, the **DISK COMMANDER catalog will stop if any key other than the space bar is pressed**. The special DISK COMMANDER commands "D" and "S" require that there be **no space** between the command and the number of the new drive or slot number. For example, **D2** and **S6** are legal commands whereas **D 1** and **S 5** are not.

OTHER FORMATS

Entering other commands also requires special formats. As noted before only the **first three letters** of DOS commands are needed. The full command will of course be accepted. Furthermore, the **command field MUST be separated from the number field by at least one space**. Otherwise an error message will

be printed. However, only numbers will be accepted in the number field.

Three of the commands require special attention. First of all the **SAVE** command can only be used with a program which has been **previously SAVED** since only numbers will be accepted in the number field.

TABLE I

A COMPARISON OF DOS AND DISK COMMANDER COMMANDS

DOS COMMAND	DISK COMMANDER COMMAND
BSAVE A,A\$800,L\$FF	BSA 1,800,FF
BLOAD A	BLO 1
RENAME A,B	REN 1,B
LOAD A	LOA 1
SAVE A	SAV 1
EXEC A	EXE 1
RUN A	RUN 1
BRUN A	BRU 1
DELETE A	DEL 1
LOCK A	LOC 1
UNLOCK A	UNL 1

TABLE II

LEGAL COMMANDS	ILLEGAL COMMANDS
RUN 14	RUN14
BLO 10	BLO10
LOAD 5	LOAD5
REN 14,NEW NAME	REN 14 NEW NAME
BSA 11,800,\$FF	BSA 11,\$800,\$FF
SAV 13	SAV PROGRAM

```

SOURCE FILE: DISK COMMANDER
0000: 1 *****
0000: 2 **
0000: 3 ** DISK COMMANDER
0000: 4 **
0000: 5 ** BY PRESTON BLACK
0000: 6 **
0000: 7 **
0000: 8 ** COPYRIGHT (C) 1982 BY
0000: 9 ** MICRO-SPARC, INC.
0000: 10 ** APPLE TOOLKIT ASSEMBLER
0000: 11 **
0000: 12 *****
0000: 13 **
0000: 14 ** EQUATES
0000: 15 **
000B: 16 BUFFS EQU *0B
0009: 17 CMMND EQU *09
000B: 18 PTRI EQU *0B
000D: 19 BUFFM EQU *0D
000F: 20 ERRSAV EQU *0F
003B: 21 KSWL EQU *3B
0039: 22 KSWH EQU *39
0073: 23 HIMEML EQU *73
0074: 24 HIMEMH EQU *74
00A0: 25 SPACE EQU *A0
00AC: 26 COMMA EQU *AC
0200: 27 BUFFER EQU *200
03D0: 28 DOS EQU *3D0
03EA: 29 DOSHOK EQU *3EA
AA6B: 30 DOSDRV EQU *AA6B
AA6A: 31 DOSLOT EQU *AA6A
B7B5: 32 RWTS EQU *B7B5
FDOC: 33 RDKEY EQU *FDOC
FD1B: 34 KEYIN EQU *FD1B
FD6F: 35 GETLN1 EQU *FD6F
FDBE: 36 CROUT EQU *FDBE
FDED: 37 COUT EQU *FDED
0000: 38 *
----- NEXT OBJECT FILE NAME IS DISK COMMANDER
8200: 39 ORG *8200
0000: 40 OBJ *800
8200: 41 *
8200: 42 * CHANGE THE INPUT HOOKS TO OUR ROUTINE
8200: 43 * AT 'EASE?' AND RESET HIMEM SO OUR
8200: 44 * ROUTINE IS NOT OVERWRITTEN
8200: 45 *
8200:A9 14 46 HOOKS LDA *EASE?
8202:B5 38 47 STA KSWL
8204:A9 82 48 LDA *EASE?
8206:B5 39 49 STA KSWH
820B:A9 80 50 LDA *80
820A:B5 73 51 STA HIMEML
820C:A9 82 52 LDA *82
820E:B5 74 53 STA HIMEMH

```

Secondly, the **RENAME** command uses the same format as the DOS command except that the file **number replaces the file name** to be changed (see table for format of this command).

Finally, the **BSAVE** command has the same constraints as the **SAVE** command and **only a file already present in the Diskette Directory can be saved**. This command is simplified somewhat in that the "A\$" and "L\$" are not needed when using DISK COMMANDER. All that is required is the starting address and length of the file to be BSAVED separated from each other and the file number by commas. The "A\$" and "L\$" are automatically included by DISK COMMANDER. Note that both the file's starting address and length **must be in hexadecimal**.

RESET AND RESTART

Should the reset key be hit when using DISK COMMANDER or the output hooks changed by a program in use, the hooks can be re-enabled to point to DISK COMMANDER by typing **CALL 33280 from Apple-soft or 8200G from the monitor**. If during this time, DISK COMMANDER is accidentally overwritten by an assembly language program, it must be reloaded and reactivated. And finally, **when files are added or DELETED**, always review the revised special catalog to see an update of the number of each file. This will prevent you from using the wrong file.

I have found that this program has saved me a considerable amount of time when using DOS. I still can use the descriptive names I like and not pay the penalty for not taking typing lessons when I was in school. I hope that the reader will find it as useful as I have. (Disk COMMANDER was assembled using the Apple Toolkit Editor/Assembler).




```

8210:20 EA 03 54 JSR DOSHOK
8213:60 55 RTS
8214: 56 #
8214: 57 # SCAN THE KEYBOARD INPUT FOR A CTRL-F
8214: 58 #
8214:20 1B FD 59 EASE? JSR KEYIN
8217:C9 B6 60 CMP #B6; CTRL-F
8219: 61 #
8219: 62 # IF IT IS TYPED THEN GOTO 'EASY'
8219: 63 # OTHERWISE CONTINUE AS BEFORE
8219: 64 #
8219:F0 01 65 BEQ EASY
821B:60 66 RTS
821C: 67 #
821C: 68 #
821C: 69 # FIRST MAKE SURE WE ARE READING THE SLOT
821C: 70 # AND DRIVE WE CALLED THE ROUTINE FROM AND THEN
821C: 71 # READ THE FILE DIRECTORY INTO RAM
821C: 72 #
821C:20 BE FD 73 EASY JSR CROUT
821F:20 BE FD 74 JSR CROUT
8222:AD 68 AA 75 LDA DOSDRV
8225:8D 99 B6 76 STA IBDRVN
8228:AD 6A AA 77 LDA DOSLOT
822B:0A 78 ASL A
822C:0A 79 ASL A
822D:0A 80 ASL A
822E:0A 81 ASL A
822F:8D 98 B6 82 STA IBLSLOT
8232:4C 2B B4 83 JMP RDCAT
8235: 84 #
8235: 85 # OUTPUT OUR PROMPT
8235: 86 #
8235:A9 A5 87 EASE1 LDA #'X'
8237:20 ED FD 88 JSR COUT
823A:A9 00 89 LDA #00
823C:85 08 90 STA BUFFS
823E: 91 #
823E: 92 # USE MONITOR ROUTINE 'GETLN1' FOR
823E: 93 # INPUTTING DISK EASE COMMANDS
823E: 94 #
823E:20 6F FD 95 GETCMD JSR GETLN1
8241:CA 96 DEX
8242:8E 8B B6 97 STX COUNT
8245:E0 30 98 CPX #30
8247:90 03 99 BCC GETCMD1
8249:4C 6F B5 100 JMP ERROR
824C:A2 00 101 GETCMD1 LDX #00
824E:BD 00 02 102 LDA BUFFER,X
8251: 103 #
8251: 104 # FIRST CHECK FOR SPECIAL DISK EASE
8251: 105 # COMMANDS 'Q' FOR 'QUIT', 'D' FOR DRIVE, 'S' FOR SLOT,
8251: 106 # AND 'C' FOR SPECIAL 'CATALOG'
8251: 107 # ALSO CHECK FOR 'RENAME' AND
8251: 108 # 'BSAVE' BECAUSE THEY NEED
8251: 109 # SPECIAL ATTENTION
8251: 110 #
8251:C9 C2 111 CMP #'B'
8253:D0 0F 112 BNE GETCMD3
8255:E8 113 INX
8256:BD 00 02 114 LDA BUFFER,X
8259:C9 D3 115 CMP #'S'
825B:D0 03 116 BNE GETCMD2
825D:4C E5 B3 117 JMP BSAVE
8260:CA 118 GETCMD2 DEX
8261:BD 00 02 119 LDA BUFFER,X
8264:C9 D2 120 GETCMD3 CMP #'R'
8266:D0 08 121 BNE GETCMD4
8268:E8 122 INX
8269:BD 00 02 123 LDA BUFFER,X
826C:C9 C5 124 CMP #'E'
826E:D0 03 125 BNE GETCMD4
8270:4C C9 B3 126 JMP RENAME
8273:A2 00 127 GETCMD4 LDX #00
8275:BD 00 02 128 LDA BUFFER,X
8278:C9 D1 129 CMP #'Q'
827A:D0 03 130 BNE GETCMD5
827C:4C D0 03 131 JMP DOS
827F:C9 C4 132 GETCMD5 CMP #'D'
8281:D0 16 133 BNE GETCMD6
8283: 134 #
8283: 135 # MAKE SURE CHARACTER AFTER 'D' IS 1 OR 2
8283: 136 #
8283:A2 01 137 LDX #01
8285:BD 00 02 138 LDA BUFFER,X
8288:C9 B1 139 CMP #B1
828A:90 34 140 BLT GETCMD8
828C:C9 B3 141 CMP #B3
828E:B0 30 142 BGE GETCMD8
8290:38 143 SEC
8291:E9 B0 144 SBC #B0
8293: 145 #
8293: 146 # STORE NEW DRIVE NUMBER INTO DOS
8293: 147 #
8293:BD 68 AA 148 STA DOSDRV
8296:4C 1C B2 149 JMP EASY
8299:C9 D3 150 GETCMD6 CMP #'S'
829B:D0 16 151 BNE GETCMD7
829D: 152 #
829D: 153 # MAKE SURE NEXT CHARACTER IS A NUMBER BETWEEN 0 AND 7
829D: 154 #
829D:A2 01 155 LDX #01
829F:BD 00 02 156 LDA BUFFER,X
82A2:C9 B0 157 CMP #B0
82A4:90 1A 158 BLT GETCMD8
82A6:C9 B8 159 CMP #B8
82A8:B0 16 160 BGE GETCMD8
82AA:38 161 SEC
82AB:E9 B0 162 SBC #B0
82AD: 163 #
82AD: 164 # PLACE NEW SLOT NUMBER INTO DOS
82AD: 165 #

```

```

82AD:BD 6A AA 166 STA DOSLOT
82B0:4C 1C B2 167 JMP EASY
82B3:C9 C3 168 GETCMD7 CMP #'C'
82B5:D0 09 169 BNE GETCMD8
82B7:20 BE FD 170 JSR CROUT
82BA:20 BE FD 171 JSR CROUT
82BD:4C A6 B4 172 JMP PRTCAT
82C0: 173 #
82C0: 174 # STORE THE COMMAND IN A BUFFER
82C0: 175 # NOTE THAT A SPACE IS USED
82C0: 176 # AS THE DELIMITER FOR THE COMMAND
82C0: 177 #
82C0:A2 00 178 GETCMD8 LDX #00
82C2:BD 00 02 179 GETCMD9 LDA BUFFER,X
82C5:9D 91 B5 180 STA CMDBUF,X
82C8:C9 A0 181 CMP #>SPACE
82CA:F0 03 182 BEQ GETCMDA
82CC:E8 183 INX
82CD:D0 F3 184 BNE GETCMD9
82CF:AE 8B B6 185 GETCMDA LDX COUNT
82D2:A0 00 186 LDY #00
82D4: 187 #
82D4: 188 # NOW STORE THE NUMBER IN A BUFFER.
82D4: 189 # AGAIN A SPACE IS USED AS A DELIMITER
82D4: 190 #
82D4:BD 00 02 191 GETCMD8 LDA BUFFER,X
82D7:C9 A0 192 CMP #>SPACE
82D9:F0 18 193 BEQ GETCMDE
82DB: 194 #
82DB: 195 # IF EACH NUMBER IS NOT BETWEEN
82DB: 196 # 0 AND 9 THEN PRINT ERROR MESSAGE
82DB: 197 #
82DB:C9 B0 198 CMP #B0
82DD:B0 03 199 BCS GETCMDC
82DF:4C 6F B5 200 JMP ERROR
82E2:C9 BA 201 GETCMDC CMP #BA
82E4:90 03 202 BCC GETCMDD
82E6:4C 6F B5 203 JMP ERROR
82E9: 204 #
82E9: 205 # CONVERT THE ASCII CODE FOR 0 TO 9
82E9: 206 # TO THE HEXIDECIMAL EQUIVALENT
82E9: 207 # BEFORE PLACING INTO THE BUFFER
82E9: 208 #
82E9:38 209 GETCMDD SEC
82EA:E9 B0 210 SBC #B0
82EC:99 98 B5 211 STA FILBUF,Y
82EF:CA 212 DEX
82F0:C8 213 INY
82F1:D0 E1 214 BNE GETCMD8
82F3:88 215 GETCMDE DEY
82F4: 216 #
82F4: 217 # STORE THE NUMBER OF PLACES IN
82F4: 218 # OUR NUMBER IN 'COUNT'
82F4: 219 #
82F4:BC 8B B6 220 STY COUNT
82F7:A0 00 221 LDY #00
82F9:A9 FF 222 LDA #FF
82FB:8D 8C B6 223 STA COUNT1
82FE: 224 #
82FE: 225 # NOW PARSE THROUGH THE COMMAND LIST
82FE: 226 # UNTIL WE FIND THE COMMAND WE WANT
82FE: 227 #
82FE:A2 00 228 GETCMDF LDX #00
8300:EE 8C B6 229 GETCMDG INC COUNT1
8303:C0 24 230 CPY #24
8305:D0 03 231 BNE GETCMDH
8307: 232 #
8307: 233 # IF NO MATCH THEN PRINT ERROR
8307: 234 #
8307:4C 6F B5 235 JMP ERROR
830A: 236 #
830A: 237 # CHECK TO SEE IF FIRST LETTERS MATCH
830A: 238 #
830A:BD 91 B5 239 GETCMDH LDA CMDBUF,X
830D:D9 08 B6 240 CMP CMDLST,Y
8310:F0 05 241 BEQ GETCMDI
8312: 242 #
8312: 243 # IF NOT THEN GO TO NEXT COMMAND IN LIST
8312: 244 #
8312:C8 245 INY
8313:C8 246 INY
8314:C8 247 INY
8315:D0 E9 248 BNE GETCMDG
8317: 249 #
8317: 250 # IF THERE IS A MATCH THEN CHECK FOR
8317: 251 # A COMPLETE MATCH OF THE COMMAND
8317: 252 #
8317:E8 253 GETCMDI INX
8318:C8 254 INY
8319:E0 03 255 CPX #03
831B:F0 10 256 BEQ GETCMDK
831D:BD 91 B5 257 LDA CMDBUF,X
8320:D9 08 B6 258 CMP CMDLST,Y
8323:F0 F2 259 BEQ GETCMDI
8325: 260 #
8325: 261 # IF THERE IS NO MATCH THEN CONTINUE
8325: 262 #
8325:E0 03 263 GETCMDJ CPX #03
8327:F0 D5 264 BEQ GETCMDF
8329:E8 265 INX
832A:C8 266 INY
832B:D0 F8 267 BNE GETCMDJ
832D: 268 #
832D: 269 # IF THERE IS A MATCH THEN STORE
832D: 270 # THE ADDRESS OF THE FULL COMMAND
832D: 271 # IN CMND ON ZERO PAGE
832D: 272 #
832D:0E 8C B6 273 GETCMDK ASL COUNT1
8330:AE 8C B6 274 LDX COUNT1
8333:BD 2C B6 275 LDA CMDFTR,X
8336:85 09 276 STA CMND
8338:E8 277 INX

```

continued on next page

Disk COMMANDER (Cont.)

```

8339:BD 2C B6 278 LDA CMDPTR,X
833C:85 0A 279 STA CMND+1
833E: 280 *
833E: 281 * NOW CONVERT THE DECIMAL NUMBERS OF THE FILE
833E: 282 * NUMBER TO ITS HEXIDECIMAL EQUIVALENT
833E: 283 *
833E:A9 00 284 GETFIL LDA #00
8340:BD BD B6 285 STA ALPHA
8343:AC BB B6 286 LDY COUNT
8346:B9 9B B5 287 GTFIL1 LDA FILBUF,Y
8349:F0 0E 288 BEQ GTFIL3
834B:AA 289 TAX
834C:1B 290 CLC
834D:B9 0B B6 291 GTFIL2 LDA TIMES,Y
8350:6D BD B6 292 ADC ALPHA
8353:BD BD B6 293 STA ALPHA
8356:CA 294 DEX
8357:D0 F4 295 BNE GTFIL2
8359:BB 296 GTFIL3 DEY
835A:10 EA 297 BPL GTFIL1
835C: 298 *
835C: 299 * THEN MULTIPLY THAT NUMBER BY TWO
835C: 300 * TO GET THE POSITION OF THE FILE'S
835C: 301 * ADDRESS IN THE ADDRESS BUFFER
835C: 302 *
835C:0E BD B6 303 ASL ALPHA
835F:AE BD B6 304 LDX ALPHA
8362:BD AC B6 305 LDA ADDR,X
8365:85 0B 306 STA PTRI
8367:EB 307 INX
8368:BD AC B6 308 LDA ADDR,X
836B:85 0C 309 STA PTRI+1
836D:A0 00 310 CMDFIL LDY #00
836F:A2 00 311 LDX #00
8371: 312 *
8371: 313 * NOW CONSTRUCT THE COMMAND STRING
8371: 314 * STARTING WITH A RETURN ($BD)
8371: 315 * AND A CONTROL-D ($B4)
8371: 316 *
8371:A9 BD 317 LDA #BD
8373:9D 9C B5 318 STA CMND,X
8376:EB 319 INX
8377:A9 B4 320 LDA #B4
8379:9D 9C B5 321 STA CMND,X
837C:EB 322 INX
837D: 323 *
837D: 324 * THEN PLACE THE FULL COMMAND INTO
837D: 325 * THE COMMAND STRING
837D: 326 *
837D:B1 09 327 CMDFL1 LDA (CMND),Y
837F:F0 07 328 BEQ CMDFL2
8381:9D 9C B5 329 STA CMND,X
8384:EB 330 INX
8385:C8 331 INY
8386:D0 F5 332 BNE CMDFL1
8388: 333 *
8388: 334 * FINALLY PLACE THE NAME OF THE FILE
8388: 335 * INTO THE COMMAND STRING
8388: 336 *
8388:A0 03 337 CMDFL2 LDY #03
838A:B1 0B 338 CMDFL3 LDA (PTRI),Y
838C:9D 9C B5 339 STA CMND,X
838F:C8 340 INY
8390:EB 341 INX
8391:C0 21 342 CPY #21
8393:D0 F5 343 BNE CMDFL3
8395:A5 0B 344 LDA BUFFS
8397:F0 0B 345 BEQ CMDFIL5
8399:AB 346 TAY
839A:B9 DC B5 347 CMDFIL4 LDA STRING,Y
839D:9D 9C B5 348 STA CMND,X
83A0:EB 349 INX
83A1:BB 350 DEY
83A2:10 F6 351 BPL CMDFIL4
83A4: 352 *
83A4: 353 * WE COMPLETE THE COMMAND STRING
83A4: 354 * WITH A RETURN AND THE DELIMITER
83A4: 355 * OF THE STRING $00
83A4: 356 *
83A4:A9 BD 357 CMDFIL5 LDA #BD
83A6:9D 9C B5 358 STA CMND,X
83A9:EB 359 INX
83AA:A9 00 360 LDA #00
83AC:85 0B 361 STA BUFFS
83AE:9D 9C B5 362 STA CMND,X
83B1:20 EA 03 363 JSR DSHOK
83B4:A2 00 364 LDX #00
83B6: 365 *
83B6: 366 * WE NOW OUTPUT THE ENTIRE COMMAND
83B6: 367 * STRING USING THE MONITOR ROUTINE
83B6: 368 * 'COUT' UNTIL WE REACH THE DELIMITER
83B6: 369 *
83B6:BD 9C B5 370 CMDFL4 LDA CMND,X
83B9:F0 06 371 BEQ CMDFL5
83BB:20 ED FD 372 JSR COUT
83BE:EB 373 INX
83BF:D0 F5 374 BNE CMDFL4
83C1:A9 BD 375 CMDFL5 LDA #BD
83C3:20 00 B2 376 JSR HOOKS
83C6:4C 1C B2 377 JMP EASY
83C9: 378 *
83C9: 379 * THIS ROUTINE HANDLES THE STRING
83C9: 380 * CONTAINING THE NEW NAME FOR THE
83C9: 381 * RENAME COMMAND
83C9: 382 *
83C9:A0 00 383 RENAME LDY #00
83CB:AE BB B6 384 LDX COUNT
83CE:BD 00 02 385 RNAME1 LDA BUFFER,X

```

```

83D1:99 DC B5 386 STA STRING,Y
83D4:C9 AC 387 CMP #>COMMA
83D6:F0 04 388 BEQ RNAME2
83D8:CA 389 DEX
83D9:C8 390 INY
83DA:D0 F2 391 BNE RNAME1
83DC:CA 392 RNAME2 DEX
83DD:8E BB B6 393 STX COUNT
83E0:84 0B 394 STY BUFFS
83E2:4C C0 B2 395 JMP GETCMD8
83E5: 396 *
83E5: 397 * THIS ROUTINE CONSTRUCTS THE
83E5: 398 * PROPER STRING FOR THE BSAVE
83E5: 399 * COMMAND
83E5: 400 *
83E5:A0 00 401 BSAVE LDY #00
83E7:84 0D 402 STY BUFFM
83E9:AE BB B6 403 LDX COUNT
83EC: 404 *
83EC: 405 * PUT LENGTH IN STRING
83EC: 406 *
83EC:BD 00 02 407 BSAVE1 LDA BUFFER,X
83EF:C9 AC 408 CMP #>COMMA
83F1:F0 07 409 BEQ BSAVE2
83F3:99 DC B5 410 STA STRING,Y
83F6:CA 411 DEX
83F7:C8 412 INY
83F8:D0 F2 413 BNE BSAVE1
83FA: 414 *
83FA: 415 * ADD 'L*' TO STRING AS WELL AS COMMA
83FA: 416 *
83FA:CA 417 BSAVE2 DEX
83FB:E6 0D 418 INC BUFFM
83FD:A9 A4 419 LDA #'*
83FF:99 DC B5 420 STA STRING,Y
8402:C8 421 INY
8403:A5 0D 422 LDA BUFFM
8405:C9 01 423 CMP #01
8407:D0 0F 424 BNE BSAVE3
8409:A9 CC 425 LDA #'L'
840B:99 DC B5 426 STA STRING,Y
840E:A9 AC 427 LDA #>COMMA
8410:C8 428 INY
8411:99 DC B5 429 STA STRING,Y
8414:C8 430 INY
8415:4C EC B3 431 JMP BSAVE1
8418: 432 *
8418: 433 * AFTER STARTING ADDRESS, ADD 'A*'
8418: 434 * TO THE STRING AND A COMMA
8418: 435 *
8418:A9 C1 436 BSAVE3 LDA #'A'
841A:99 DC B5 437 STA STRING,Y
841D:A9 AC 438 LDA #>COMMA
841F:C8 439 INY
8420:99 DC B5 440 STA STRING,Y
8423:84 0B 441 STY BUFFS
8425:8E BB B6 442 STX COUNT
8428:4C C0 B2 443 JMP GETCMD8
8428: 444 *
8428: 445 * THIS SUBROUTINE READS THE DISKETTE
8428: 446 * DIRECTORY INTO MEMORY AND PLACES
8428: 447 * INTO MEMORY A DIRECTORY OF THE
8428: 448 * STARTING ADDRESS OF EACH FILE
8428: 449 *
8428:A9 0B 450 RDCAT LDA #0B
842D:BD BD B6 451 STA ALPHA
8430:A9 23 452 LDA #23
8432:BD BE B6 453 STA BETA
8435:A9 00 454 LDA #>BUFFP
8437:BD 9F B6 455 STA IBUFFP
843A:85 0D 456 STA BUFFM
843C:A9 B7 457 LDA #<BUFFP
843E:BD A0 B6 458 STA IBUFFP+1
8441:85 0E 459 STA BUFFM+1
8443: 460 *
8443: 461 * IBTRK IS INITIALIZED WITH $11(17)
8443: 462 * AND IBSECT IS INITIALIZED WITH $0F(15)
8443: 463 *
8443:A9 11 464 RDCAT1 LDA #11
8445:BD 9B B6 465 STA IBTRK
8448:A9 0F 466 LDA #0F ; CHANGE #0F TO #0C FOR DOS 3.
844A:BD 9C B6 467 STA IBSECT
844D: 468 *
844D: 469 * THEN USE THE RWTS SUBROUTINE TO READ
844D: 470 * EACH SECTOR OF THE DIRECTORY INTO
844D: 471 * MEMORY SEQUENTIALLY
844D: 472 *
844D:20 BF B6 473 RDCAT2 JSR CONTRL
8450:EE A0 B6 474 INC IBUFFP+1
8453:CE 9C B6 475 DEC IBSECT
8456:D0 F5 476 BNE RDCAT2
8458:A2 00 477 LDX #00
845A:86 0D 478 STX BUFFM
845C:AD BD B6 479 LDA ALPHA
845F:9D AC B6 480 STA ADDR,X
8462:EB 481 INX
8463:A5 0E 482 LDA BUFFM+1
8465:9D AC B6 483 STA ADDR,X
8468: 484 *
8468: 485 * NOW SCAN THE NEWLY READ DIRECTORY
8468: 486 * AND PLACE THE ADDRESS OF THE START
8468: 487 * OF EACH FILE ENTRY INTO THE ADDRESS BUFFER
8468: 488 *
8468:AC BD B6 489 RDCAT3 LDY ALPHA
846B:B1 0D 490 RDCAT4 LDA (BUFFM),Y
846D:F0 34 491 BEQ RDCAT7
846F: 492 *
846F: 493 * IGNORE DELETED FILES
846F: 494 *
846F:C9 FF 495 CMP #FF
8471:F0 02 496 BEQ RDCAT5
8473:EB 497 INX
8474:EB 498 INX

```



```

8475:CA 499 RDCAT5 DEX
8476:9B 500 TYA
8477:1B 501 CLC
8478:6D 8E B6 502 ADC BETA
8479:AB 503 TAY
847C:C0 00 504 CPY ##00
847E:F0 0D 505 BEQ RDCAT6
8480:9D AC B6 506 STA ADDRA,X
8483:EB 507 INX
8484:A5 0E 508 LDA BUFFM+1
8486:9D AC B6 509 STA ADDRA,X
8489:C0 00 510 CPY ##00
848B:D0 DE 511 BNE RDCAT4
848D:A9 0B 512 RDCAT6 LDA ##0B
848F:8D 8D B6 513 STA ALPHA
8492:9D AC B6 514 STA ADDRA,X
8495:A9 00 515 LDA ##00
8497:85 0D 516 STA BUFFM
8499:EB 517 INX
849A:E6 0E 518 INC BUFFM+1
849C:A5 0E 519 LDA BUFFM+1
849E:9D AC B6 520 STA ADDRA,X
84A1:D0 C5 521 BNE RDCAT3
84A3:4C 35 B2 522 RDCAT7 JMP EASE1
84A6: 523 #
84A6: 524 # THIS SUBROUTINE PRINTS OUR SPECIAL
84A6: 525 # CATALOG WITH EACH FILE NUMBERED
84A6: 526 #
84A6:A2 00 527 PRICAT LDX ##00
84AB:8E 8D B6 528 STX ALPHA
84AB:BD AC B6 529 PRINT1 LDA ADDRA,X
84AE:85 0B 530 STA PTRI
84B0:EB 531 INX
84B1:BD AC B6 532 LDA ADDRA,X
84B4:85 0C 533 STA PTRI+1
84B6:EB 534 INX
84B7:8E 8E B6 535 STX BETA
84BA:A0 00 536 LDY ##00
84BC:B1 0B 537 LDA (PTRI),Y
84BE:D0 03 538 BNE PRINT2
84C0:4C 3B B5 539 JMP PRINTB
84C3: 540 #
84C3: 541 # AGAIN IGNORE DELETED FILES
84C3: 542 #
84C3:C9 FF 543 PRINT2 CMP ##FF
84C5:F0 E4 544 BEQ PRINT1
84C7:EE 8D B6 545 INC ALPHA
84CA:A2 00 546 LDX ##00
84CC: 547 #
84CC: 548 # FIRST PRINT THE FILE NUMBER
84CC: 549 #
84CC:BD 04 B6 550 PRINT3 LDA NUMB,X
84CF:20 ED FD 551 JSR COUT
84D2:EB 552 INX
84D3:E0 04 553 CPX ##04
84D5:D0 F5 554 BNE PRINT3
84D7:20 4D B5 555 JSR NUMCHK
84DA: 556 #
84DA: 557 # THEN CHECK TO SEE IF THE FILE IS
84DA: 558 # LOCKED OR UNLOCKED
84DA: 559 #
84DA:A0 02 560 LDY ##02
84DC:B1 0B 561 LDA (PTRI),Y
84DE:0A 562 ASL A
84DF:4B 563 PHA
84E0:B0 04 564 BCS PRINT4
84E2:A9 A0 565 LDA >>SPACE
84E4:D0 02 566 BNE PRINT5
84E6: 567 #
84E6: 568 # PRINT '*' IF IT IS ELSE PRINT A SPACE
84E6: 569 #
84E6:A9 AA 570 PRINT4 LDA #'*'
84EB: 571 #
84EB: 572 # THEN PRINT THE FILE TYPE
84EB: 573 #
84EB:20 ED FD 574 PRINT5 JSR COUT
84EB:6B 575 PLA
84EC:4A 576 LSR A
84ED:C9 04 577 CMP ##04
84EF:D0 04 578 BNE PRINT6
84F1:A9 C2 579 LDA #'B'
84F3:D0 12 580 BNE PRINT9
84F5:C9 02 581 PRINT6 CMP ##02
84F7:D0 04 582 BNE PRINT7
84F9:A9 C1 583 LDA #'A'
84FB:D0 0A 584 BNE PRINT9
84FD:C9 01 585 PRINT7 CMP ##01
84FF:D0 04 586 BNE PRINT8
8501:A9 C9 587 LDA #'I'
8503:D0 02 588 BNE PRINT9
8505:A9 D4 589 PRINT8 LDA #'T'
8507:20 ED FD 590 PRINT9 JSR COUT
850A:A9 A0 591 LDA >>SPACE
850C:20 ED FD 592 JSR COUT
850F:CB 593 INY
8510: 594 #
8510: 595 # NOW PRINT THE FILE NAME
8510: 596 #
8510:B1 0B 597 PRINTA LDA (PTRI),Y
8512:F0 27 598 BEQ PRINTB
8514:20 ED FD 599 JSR COUT
8517:CB 600 INY
8518:C0 21 601 CPY ##21
851A:D0 F4 602 BNE PRINTA
851C:20 BE FD 603 JSR CROUT
851F:AE 8E B6 604 LDX BETA
8522:AD 8D B6 605 LDA ALPHA
8525: 606 #
8525: 607 # PAUSE UNTIL KEYBOARD INPUT AFTER
8525: 608 # TWENTY FILE NAMES HAVE BEEN LISTED
8525: 609 #
8525:C9 14 610 CMP ##14
8527:D0 B2 611 BNE PRINT1

```

```

8529:20 0C FD 612 JSR RDKEY
852C: 613 #
852C: 614 # IF KEYBOARD INPUT NOT A 'SPACE'
852C: 615 # THEN DISCONTINUE 'CATALOG'
852C: 616 #
852C:C9 A0 617 CMP >>SPACE
852E:D0 0B 618 BNE PRINTB
8530:AE 8E B6 619 LDX BETA
8533:A9 00 620 LDA ##00
8535:8D 8D B6 621 STA ALPHA
8538:4C AB B4 622 JMP PRINT1
853B:A2 00 623 PRINTB LDX ##00
853D:A9 B0 624 LDA ##B0
853F: 625 #
853F: 626 # RESTORE NUMB TO BOBOBO (000)
853F: 627 # BEFORE RETURNING TO MAIN PROGRAM
853F: 628 #
853F:9D 04 B6 629 PRINTC STA NUMB,X
8542:EB 630 INX
8543:E0 03 631 CPX ##03
8545:D0 F8 632 BNE PRINTC
8547:20 BE FD 633 JSR CROUT
854A:4C 35 B2 634 JMP EASE1
854D: 635 #
854D: 636 # THIS ROUTINE IS USED TO INCREMENT
854D: 637 # THE BUFFER HOLDING THE ASCII CODE
854D: 638 # FOR THE FILE NUMBERS
854D: 639 #
854D:EE 06 B6 640 NUMCHK INC NUMB+2
8550:AD 06 B6 641 LDA NUMB+2
8553:C9 BA 642 CMP ##BA
8555:90 17 643 BCC RTURN
8557:A9 B0 644 LDA ##B0
8559:8D 06 B6 645 STA NUMB+2
855C:EE 05 B6 646 INC NUMB+1
855F:AD 05 B6 647 LDA NUMB+1
8562:C9 BA 648 CMP ##BA
8564:90 0B 649 BCC RTURN
8566:A9 B0 650 LDA ##B0
8568:8D 05 B6 651 STA NUMB+1
856B:EE 04 B6 652 INC NUMB
856E:60 653 RTURN RTS
856F: 654 #
856F: 655 # THIS ROUTINE PRINTS THE ERROR MESSAGE
856F: 656 #
856F:20 BE FD 657 ERROR JSR CROUT
8572:A2 00 658 LDX ##00
8574:BD B2 B5 659 ERROR1 LDA ERRMSG,X
8577:F0 06 660 BEQ ERROR2
8579:20 ED FD 661 JSR COUT
857C:EB 662 INX
857D:D0 F5 663 BNE ERROR1
857F:4C 35 B2 664 ERROR2 JMP EASE1
8582: 665 #
8582:87 C2 C1 666 ERRMSG ASC "BAD COMMAND"
8585:C4 A0 C3
8588:CF CD CD
858B:C1 CE C4
858E:8D 8D 00 667 DFB $BD,$BD,$00
8591: 668 #
8591: 669 # BUFFERS
8591: 670 #
8591: 671 CMDBUF DS $7
8598: 672 FILBUF DS $4
859C: 673 COMMAND DS $40
85DC: 674 STRING DS $2B
8604:B0 B0 B0 675 NUMB DFB $B0,$B0,$B0,$A0
8607:A0
8608:01 0A 64 676 TIMES DFB $01,$0A,$64
860B: 677 #
860B: 678 # ABBREVIATED COMMAND LIST
860B: 679 #
860B:CC CF C1 680 CMDLST ASC "LOA"
860E:D5 CE CC 681 ASC "UNL"
8611:C4 C5 CC 682 ASC "DEL"
8614:D2 D5 CE 683 ASC "RUN"
8617:C5 D8 C5 684 ASC "EXE"
861A:C2 CC CF 685 ASC "BLO"
861D:C2 D2 D5 686 ASC "BRU"
8620:CC CF C3 687 ASC "LOC"
8623:D3 C1 D6 688 ASC "SAV"
8626:D2 C5 CE 689 ASC 'REN'
8629:C2 D3 C1 690 ASC 'BSA'
862C: 691 #
862C: 692 # BUFFER HOLDING ADDRESSES OF FULL
862C: 693 # COMMAND LIST
862C: 694 #
862C:42 B6 695 CMDPTR DW WCMD1
862E:4B B6 696 DW WCMD2
8630:50 B6 697 DW WCMD3
8632:58 B6 698 DW WCMD4
8634:5D B6 699 DW WCMD5
8636:63 B6 700 DW WCMD6
8638:6A B6 701 DW WCMD7
863A:70 B6 702 DW WCMD8
863C:76 B6 703 DW WCMD9
863E:7C B6 704 DW WCMDA
8640:84 B6 705 DW WCMD8

```

continued on page 191

LOWER CASE INPUT/OUTPUT

Lower Case Input/Output

by Ben E. Colley
Rt. 1, Box 67B
Hallsville, MO 65255

People who do a great deal of software development soon fall into the habit of collecting subroutine modules. These modules aid development in numerous ways, by saving coding time and debugging effort for new programs. And in some cases, machine language routines can perform a function much faster than their BASIC counterparts. This article shares with you two such subroutines that I have developed and subsequently collected for future use in developing and enhancing programs.

LC INPUT

If you have a printer that supports lower case printing, then this subroutine (LIST 1) offers you a generalized starting point for the entry of upper and lower case text. It uses (internally only) the ASCII codes for Backspace (8), Carriage Return (13), and Escape (27). The ASCII code 0 cannot be entered successfully into this routine. The Applesoft command ASC disallows this character, but you could add the checks to include it at the expense of increasing the execution time.

LC INPUT can be used in either of two ways. **One way** is to simply include the code in-line, that is, not as a subroutine, but simply in the flow of your programs. The **second way**

(which I prefer) is to add a RETURN statement at the end and then use a GOSUB statement whenever upper and lower case text is required. If you wish to use certain control codes as special functions, they can easily be handled in the routine prior to being concatenated to the string LC\$.

I should point out a couple of techniques in LC INPUT because they address some of the problems people face when learning to program. **The first technique is avoiding the use of GOTOs.** Notice that the entire loop is controlled by **FOR QL = 1 TO 2 STEP 0**. I try to structure programs as much as possible; the truth is, I dislike using GOTO at all, but sometimes it is a necessary evil. Limiting GOTO statements in an interpretive language like Applesoft is a fairly easy way to enhance your program's speed.

The **second technique** is to try to limit the use of built-in functions. Here the routine sets **AC = ASC(A\$)** and then checks for AC=8, AC>31, etc. Built-in functions introduce overhead that can slow program execution.

LC OUTPUT

The complement of text input is (obviously) text output. LC OUTPUT is a machine language subroutine (LIST 2) that displays the inverse/normal text that is so typical of text editing programs for the Apple. And it does so at a speed far superior to its counterpart written in Applesoft. In addition to inverse/normal output, LC OUTPUT displays control codes as flashing output. Adding this routine

to the T.O.U.G.H. processor makes the display of text fairly fly!

To use LC OUTPUT, key in the machine code at \$300, then save it with:

BSAVE LC OUTPUT, A\$300, L\$53

After saving it, BLOAD it from your BASIC program and use **CALL 768, var**, where **var** is a string variable, either simple (**A\$**) or array (**A\$(I)**). If you need to use location \$300 for another machine language program, LC OUTPUT is completely relocatable. BLOAD it anywhere in high memory and call the address where it was BLOADED. (Be sure to protect it with HIMEM if you do, otherwise it may get clobbered.)

LC OUTPUT was written to display true ASCII character strings, that is, character values between 0 and 127 inclusive. If you use it to display a regular Applesoft string, all the alphabetic characters will be displayed in inverse (upper case) mode.

While I have written these two routines to complement one another, they need not be supported by each other. For example, I use the output routine in the enhanced T.O.U.G.H. word processor and the input routine in some label producing programs.

Try running the demonstration program (LIST 3) and I think you will find these two subroutines worthy of inclusion in your library of subroutines.

NOTE: The assembly listing was produced using Apple's DOS Toolkit Assembler.

```
1 REM *****
2 REM * LOWER CASE INPUT/OUTPUT *
3 REM * BY BEN E. COLLEY *
4 REM * COPYRIGHT (C) 1982 *
5 REM * BY MICRO-SPARC INC *
6 REM * LINCOLN, MA. 01773 *
7 REM * ALL RIGHTS RESERVED *
8 REM *****
10 REM LC INPUT
20 LC$ = "": UC = 0
30 FOR QL = 1 TO 2 STEP 0
40 GET A$: AC = ASC (A$)
50 IF (AC > 31 AND AC < 65) OR AC >
90 THEN PRINT A$; UC = 0
60 IF AC = 27 THEN UC = 1
70 IF NOT UC AND AC > 64 AND AC <
91 THEN AC = AC + 32: PRINT A$;
80 IF UC AND AC > 64 AND AC < 91 THEN
INVERSE : PRINT A$; NORMAL : UC
C = 0
90 IF AC < > 27 AND AC < > 13 AND
AC < > 8 THEN A$ = CHR$ (AC);
LC$ = LC$ + A$
100 QM = LEN (LC$)
110 IF AC = 8 AND QM = 1 THEN IF ASC
(LC$) > 31 THEN PRINT A$; CALL
- 86B
120 IF AC = 8 AND QM = 1 THEN LC$ =
""
130 IF AC = 8 AND QM > 1 THEN IF ASC
( RIGHT$ (LC$, 1)) > 31 THEN PRINT
A$; CALL - 86B
140 IF AC = 8 AND QM > 1 THEN LC$ =
LEFT$ (LC$, QM - 1)
150 IF AC = 13 THEN PRINT : QL = 2
160 NEXT
170 REM ON EXIT LC$ IS UPPER/LOWER
CASE STRING
```

```
1 REM *****
2 REM * LOWER CASE DEMO *
3 REM * BY BEN E. COLLEY *
4 REM * COPYRIGHT (C) 1982 *
5 REM * BY MICRO-SPARC INC *
6 REM * LINCOLN, MA. 01773 *
7 REM * ALL RIGHTS RESERVED *
8 REM *****
10 :
20 :
30 REM UPPER - LOWER CASE DEMONST
RATION
40 :
50 :
60 TEXT 4: HOME
70 HTAB 8: PRINT "UPPER CASE - LOWE
R CASE"
80 HTAB 13: PRINT "DEMONSTRATION"
90 PRINT CHR$ (4); "BLOAD LC OUTPUT
"
100 VTAB 5: HTAB 10: PRINT "ASCII C
HARACTER SET"
110 VTAB 7
120 FOR I = 0 TO 127: LC$ = LC$ + CHR$
(I): NEXT
130 CALL 768, LC$
140 FOR I = 1 TO 4000: NEXT
150 HTAB 1: VTAB 5: CALL - 95B
160 PRINT : PRINT "NOW YOU MAY TRY
THE INPUT ROUTINE"
170 PRINT "ESC KEY SIGNALS CAPITAL
LETTERS"
180 DIM IN$(5)
190 FOR I = 1 TO 5
200 PRINT I; " ";
210 GOSUB 400
220 IN$(I) = LC$
230 NEXT
240 REM PRINT THE MATRIX
250 VTAB 14
```

```
260 HTAB 9: PRINT "HERE IS WHAT YOU
KEYED"
270 FOR I = 1 TO 5
280 PRINT I; " ";
290 CALL 768, IN$(I): PRINT
300 NEXT
310 VTAB 23: PRINT "THAT'S ALL!";
320 END
400 REM LC INPUT
410 LC$ = "": UC = 0
420 FOR QL = 1 TO 2 STEP 0
430 GET A$: AC = ASC (A$)
440 IF (AC > 31 AND AC < 65) OR AC >
90 THEN PRINT A$; UC = 0
450 IF AC = 27 THEN UC = 1
460 IF NOT UC AND AC > 64 AND AC <
91 THEN AC = AC + 32: PRINT A$;
470 IF UC AND AC > 64 AND AC < 91 THEN
INVERSE : PRINT A$; NORMAL : UC
C = 0
480 IF AC < > 27 AND AC < > 13 AND
AC < > 8 THEN A$ = CHR$ (AC);
LC$ = LC$ + A$
490 QM = LEN (LC$)
500 IF AC = 8 AND QM = 1 THEN IF ASC
(LC$) > 31 THEN PRINT A$; CALL
- 86B
510 IF AC = 8 AND QM = 1 THEN LC$ =
""
520 IF AC = 8 AND QM > 1 THEN IF ASC
( RIGHT$ (LC$, 1)) > 31 THEN PRINT
A$; CALL - 86B
530 IF AC = 8 AND QM > 1 THEN LC$ =
LEFT$ (LC$, QM - 1)
540 IF AC = 13 THEN PRINT : QL = 2
550 NEXT
560 REM ON EXIT LC$ IS UPPER/LOWER
CASE STRING
570 RETURN
```


Lower Case Input/Output (Cont.)

```

0000: 2 * * * * *
0000: 3 * * * * *
0000: 4 * UPPER/LOWER CASE OUTPUT
0000: 5 * BY BEN E. COLLEY
0000: 6 * COPYRIGHT (C) 1982
0000: 7 * BY MICRO-SPARC INC
0000: 8 * THIS SUBROUTINE OUTPUTS THE
0000: 9 * INVERSE/NORMAL CHARACTER
0000: 10 * STRING TYPICAL OF MOST TEXT
0000: 11 * EDITING PROGRAMS.
0000: 12 *
0000: 13 * TO USE:
0000: 14 * CALL NNNN,STR
0000: 15 *
0000: 16 * WHERE:
0000: 17 * NNNN = BLOAD ADDRESS
0000: 18 * STR = STRING VARIABLE
0000: 19 *
0000: 20 * FOR EXAMPLE:
0000: 21 * CALL 768,LC*
0000: 22 *
0000: 23 * * * * *

```

```

00A3: 25 BADTYP EQU #A3 TYPE MISMATCH ERR CODE
DEBE: 26 CHKCOM EQU #DEBE CHECK FOR A COMMA
FDF6: 27 COUTZ EQU #FDF6 VIDEO OUTPUT
D412: 28 ERROR EQU #D412 APPLESOFT ERROR RTN
DFE3: 29 PTRGET EQU #DFE3 GET VARIABLE ADDR
0011: 30 VALTYP EQU #11 APPLESOFT VARIABLE TYPE
0000: 31 *
0040: 32 MFLASH EQU #40 MASK FOR FLASH OUTPUT
003F: 33 MINVRS EQU #3F MASK FOR INVERSE OUTPUT
00B0: 34 MNORML EQU #B0 MASK FOR NORMAL OUTPUT
00DF: 35 MBIT2 EQU #DF MASK BIT 2
0000: 36 *
0006: 37 STRADR EQU #6 ADDR OF STRING
000B: 38 STRDSC EQU #B ADDR OF STRING DESCRIPTOR
000B: 39 STRLEN EQU #B LENGTH OF STRING

```

```

----- NEXT OBJECT FILE NAME IS LC OUTPUT.S.OBJO
0300: 41 ORG $300
0300:20 BE DE 43 JSR CHKCOM GO FIND THE COMMA
0303:20 E3 DF 44 JSR PTRGET AND THEN THE STR ADDR

```

```

0306:24 11 45 BIT VALTYP INSURE VARIABLE IS A STRING
030B:D0 05 46 BNE LCSTRT VARIABLE OK, GO BEGIN
030A:A2 A3 47 LDX #BADTYP SETUP ERROR CODE
030C:4C 12 D4 48 JMP ERROR
030F:85 08 49 LCSTRT STA STRDSC SAVE DESCRIPTOR ADDR
0311:84 09 50 STY STRDSC+1
0313:A0 00 51 LDY #0
0315:B1 08 52 LDA (STRDSC),Y GET THE STRING LENGTH
0317:F0 39 53 BEQ LCRTS LEN=0, GOBACK
0319:AA 54 TAX CAN'T SAVE (YET)
031A:CB 55 INY
031B:B1 08 56 LDA (STRDSC),Y NOW GET THE ADDR
031D:85 06 57 STA STRADR
031F:CB 58 INY
0320:B1 08 59 LDA (STRDSC),Y GET THE HIGH PART TOO!
0322:85 07 60 STA STRADR+1
0324:B6 08 61 STX STRLEN SAVE LENGTH
0326: 62 *
0326: 63 * SETUP DONE, PRINT THE STRING
0326: 64 *
0326:A0 00 65 LDY #0 INIT INDEX REG
032B:B1 06 66 LCLLOOP LDA (STRADR),Y GET CHAR TO OUTPUT
032A:C9 20 67 CMP #20 VAL < 32?
032C:B0 04 68 BGE LCCHK2 NO, TRY NEXT
032E:09 40 69 ORA #MFLASH YES, SET FLASH
0330:D0 18 70 BNE LCDSPL AND GO DISPLAY
0332:C9 41 71 LCCHK2 CMP #41 VAL < 65?
0334:90 08 72 BLT LCNORM YES, GO DISPLAY AS IS
0336:C9 5B 73 CMP #5B VAL >= 91?
0338:B0 04 74 BGE LCNORM YES, GO DISPLAY AS IS
033A:29 3F 75 BNE #MINVRS NO, SET INVERSE RANGE
033C:D0 0C 76 AND LCDSPL AND GO DISPLAY
033E:C9 61 77 LCNORM CMP #61 CHECK FOR
0340:90 06 78 BLT LCNRM2 PSEUDO LOWER CASE
0342:C9 7B 79 CMP #7B MUST MASK THIS DATA
0344:B0 02 80 BGE LCNRM2
0346:29 DF 81 AND #MBIT2 CLEAR BIT (SUBTRACT 32)
0348:09 80 82 LCNRM2 ORA #MNORML SET NORMAL RANGE
034A:D0 F6 FD 83 LCDSPL JSR COUTZ AND SEND TO VIDEO
034D:CB 84 INY INCR TO NEXT CHAR
034E:C4 08 85 CPY STRLEN MORE?
0350:90 D6 86 BLT LCLLOOP YES, GO AGAIN
0352:60 87 LCRTS RTS NO, GOBACK

```

```

0353: 89 *
0353: 90 * END
0353: 91 *
*** SUCCESSFUL ASSEMBLY: NO ERRORS

```

Compare Basic Programs for Differences

by Charles Boody
317 17th Ave North
Hopkins, MN 55343

I often use programs that are published in various journals, and almost always find they lack some small (and sometimes some large) pieces that will make them more useful for me. Once I have made changes or additions, I often like to pass the changes on to others in the form of an article or list of changes. Unfortunately, it is difficult to remember what changes have been made in which lines of what parts of which subroutines, and how those changes affect other parts of the program. I also often find myself with more than one version of a program: my own or public domain ones, that seem to have small differences that are very hard to track down. Sometimes I find that I have kept several versions of a program under development, but am not sure about the detailed differences in them. Other computer users often want to copy my enhancements, but not by hand.

CAPTURE AND COMPARE

To alleviate these problems, I wrote the COMPARE AND CAPTURE programs. To use them, you must first make a text file copy of the two programs you want to compare. I use the COMPARE programs to do this. They are adaptations of program found in the Apple D.O.S. manual. A user should key in MAKE CAPTURE, save it BEFORE running; and then RUN it. The program will create text files named CAPTURE.A and CAPTURE.I

which can be EXECed. The command EXEC CAPTURE.A will add CAPTURE as line 63999 of your Applesoft program. Similarly, the command EXEC CAPTURE.I will add lines 32766 and 32767 to Integer Basic programs. If you have such line numbers in the program, it will be necessary to renumber them before EXECuting either version of CAPTURE. Once you have EXECuted CAPTURE, a RUN 63999 command (RUN 32766 in Integer) will give you prompts to name the text file you are about to create, and then create the text file. Be careful NOT to give the text file the same name as the program, or you will have problems and will get a FILE TYPE MISMATCH error, or worse, lose your program! If the program is very large, you will probably want to save the text file on a rather empty disk, for it will be very long. (A 65 sector program converted to a text file is 95 sectors long.)

THE COMPARISON

Once you have created appropriately named text files for the two programs you need to compare (I have fallen into the habit of calling them OLD and GNU for obvious reasons), a RUN COMPARE will produce a listing showing each Changed statement, each Deleted statement, and each Added statement.

In the instances where a statement was changed, the old and new versions will both be printed with appropriate FROM and TO remarks. If you wish, it will also create a text file that can be EXECuted to update earlier versions conveniently.

LIMITATIONS

There are some limitations to the program that could cause minor difficulties for the

user. The inclusion of some special control characters in a program might cause problems for the printer. A program line containing more than 255 characters will get broken into two lines. Normally this is not a problem, but if by chance the point of break was immediately followed by a number greater than the line number currently being processed, your output could be quite scrambled. Neither of these problems has occurred in my use of the program. Finally, both versions of the program should have been similarly numbered (line numbers) to begin with, or the results, though accurate, will be rather meaningless.

MODIFICATIONS

Users will undoubtedly want to adapt the program to their own printers, and it should be easy to do so. The printer on/off routines are clearly marked and should be adjusted to conform to your printer. Another change you may want to make is clearly marked by a REM statement: the variable LL which controls the length of the printed line.

There are many other uses for the CAPTURE programs. One can enter changes to a program that appear in a journal, use CAPTURE to turn the changes into a text file, and EXEC that text file to put the changes into the original program without the usual laborious searching for lines and making the changes one by one.

I hope others will find these programs as useful as I have for keeping track of changes in programs. The whole procedure could also be programmed in Integer BASIC without much difficulty. Perhaps at a later date someone will do so.

listed on next page

Compare Basic Programs for Differences (Cont.)

```

1 REM *****
2 REM # APPLESOFT COMPARE #
3 REM # BY CHARLES BOODY #
4 REM # COPYRIGHT (C) 1982 #
5 REM # BY MICRO-SPARC INC #
6 REM # LINCOLN, MA. 01773 #
7 REM *****
10 REM ** LENGTH OF PROGRAM LINE TO BE
20 REM ** PRINTED ON A SINGLE LINE IS
30 REM ** SET BY LL IN THE STATEMENT
40 REM ** BELOW
50 PRINT D$;"MAXFILES4"
60 LL = 50
70 TEXT : HOME : VTAB 5
80 REM
90 REM ** ENTER THE NAMES FILES TO BE
100 REM ** COMPARED—MUST BE TEXT FILES
110 REM ** CREATED BY EXEC OF CAPTURE
120 REM ** GIVE NAME OF CHANGE FILE
130 REM ** IF ONE IS TO BE CREATED
140 D$ = CHR$(4): INPUT "OLD PROGRAM TEXT FILE NAME
";O$: INPUT "NEW PROGRAM TEXT FILE NAME ";N$
150 VTAB 12: INPUT "MAKE AN EXECUTE FILE OF THE CHANG
ES? (1=YES, 0=NO)—PRESS RETURN ";CK$: IF CK$ <
> 0 AND CK$ < > 1 THEN PRINT CHR$(7): GOTO
150
160 IF CK$ = 1 THEN INPUT "NAME THE EXEC FILE: ";C$:
PRINT D$;"OPEN";C$: PRINT D$;"DELETE";C$: PRINT
D$;"OPEN";C$
170 GOSUB 1070: REM PRINTER ON
180 ONERR GOTO 760
190 GOTO 440
200 REM
210 REM ** GETS A STATEMENT FROM THE
220 REM ** "OLD" FILE
230 REM
240 OS$ = "": PRINT D$;"READ ";O$
250 GET A$: IF A$ < > CHR$(13) OR LEN (OS$) = 0 THEN
IF LEN (OS$) < 255 THEN OS$ = OS$ + A$: GOTO 25
0
260 PRINT : RETURN
270 REM
280 REM ** GETS A STATEMENT FROM THE
290 REM ** "NEW" FILE
300 REM
310 NS$ = "": PRINT D$;"READ ";N$
320 GET A$: IF A$ < > CHR$(13) OR LEN (NS$) = 0 THEN
IF LEN (NS$) < 255 THEN NS$ = NS$ + A$: GOTO 32
0
330 PRINT : RETURN
340 REM
350 REM ** PRINTS A STATEMENT
360 REM ** IN LL LONG LINES
370 REM
380 II = 0
390 IO = II + 1: II = II + LL: PRINT TAB(12); MID$(J
$,IO,LL): IF II < LEN (J$) THEN 390
400 RETURN
410 REM
420 REM ** MAIN PROGRAM LOOP
430 REM
440 PRINT D$;"OPEN "O$: PRINT D$;"OPEN ";N$
450 GOSUB 240: GOSUB 310
460 REM
470 REM ** IF STATEMENT # SAME AND
480 REM ** CONTENTS SAME IGNORE
490 REM
500 OS = VAL (OS$): NS = VAL (NS$): IF VAL (OS$) = VAL
(NS$) AND OS$ = NS$ THEN 450
510 REM
520 REM ** IF STATEMENT #S SAME BUT
530 REM ** CONTENTS NOT PRINT "CHANGED"
540 REM ** AND GET NEXT STATEMENT
550 REM
560 IF VAL (OS$) = VAL (NS$) AND OS$ < > NS$ THEN
PRINT "CHANGED ": PRINT TAB(5); J$ = OS$: GOSUB
380: PRINT "TO: "; PRINT TAB(5); J$ = NS$: GOSUB
380: GOSUB 860: GOTO 450
570 REM
580 REM ** IF OLD LINE# < NEW THEN
590 REM ** OLD LINE WAS DELETED
600 REM ** PRINT FACT AND GET NEXT
610 REM
620 IF VAL (OS$) < VAL (NS$) THEN PRINT "DELETED "
: PRINT TAB(5); J$ = OS$: GOSUB 380: GOSUB 240:
GOSUB 950: GOTO 500
630 REM
640 REM ** IF NONE OF ABOVE THEN
650 REM ** NEW LINE IS ADDED.
660 REM ** PRINT FACT AND GET NEXT.
670 REM
680 PRINT "ADDED ": PRINT TAB(5); J$ = NS$: GOSUB 3
80: GOSUB 310: GOSUB 860: GOTO 500
690 REM
700 REM ** WHEN A FILE IS EMPTY
710 REM ** DETERMINE WHICH AND
720 REM ** ALL IN REMAINING FILE
730 REM ** ARE EITHER ADDED OR
740 REM ** DELETED.
750 REM
760 X = PEEK (218) + PEEK (219) * 256: POKE 216,0: ONERR
GOTO 1020
770 IF X > 310 THEN 790
780 PRINT "ADDED ": GOSUB 310: J$ = NS$: GOSUB 380: GOSUB
860: GOTO 780
790 IF VAL (OS$) < > NS THEN PRINT "DELETED ": J$ =
OS$: GOSUB 380

```

```

800 PRINT "DELETED ": GOSUB 240: J$ = OS$: GOSUB 380: GOSUB
950: GOTO 800
810 REM
820 REM ** IF CREATING EXEC FILE
830 REM ** STORE CHANGED OR ADDED
840 REM ** LINE.
850 REM
860 IF CK$ = 0 THEN RETURN
870 PRINT D$;"WRITE ";C$: PRINT J$
880 PRINT
890 PRINT D$;"READ";C$: RETURN
900 REM
910 REM ** IF CREATING EXEC FILE
920 REM ** AND STATEMENT IS DELETED
930 REM ** STORE STATEMENT NUMBER
940 REM
950 IF CK$ = 0 THEN RETURN
960 PRINT D$;"WRITE ";C$: PRINT VAL (J$)
970 PRINT D$;"READ";C$: RETURN
980 REM
990 REM ** END OF BOTH FILES REACHED
1000 REM ** SO PROGRAM ENDS
1010 REM
1020 PRINT : PRINT "END OF COMPARISONS": PRINT D$;"CL
OSE": GOSUB 1080
1030 END
1040 REM
1050 REM ***PRINTER ON AND OFF
1060 REM
1070 PRINT D$;"PR#1": RETURN
1080 PRINT D$;"PR#0": RETURN

```

```

1 REM *****
2 REM # MAKE CAPTURE #
3 REM # BY CHARLES BOODY #
4 REM # COPYRIGHT (C) 1982 #
5 REM # BY MICRO-SPARC INC #
6 REM # LINCOLN, MA. 01773 #
7 REM *****
10 D$ = CHR$(4): A$ = "CAPTURE.A": PRINT D$;"OPEN";A$
: PRINT D$;"WRITE ";A$: POKE 33,30: LIST 63999: PRINT
D$;"CLOSE";A$: TEXT
20 PRINT D$;"LOCK";A$
30 D$ = CHR$(4): A$ = "CAPTURE.I": PRINT D$;"OPEN";A$
: PRINT D$;"WRITE ";A$: POKE 33,30: LIST 32766,327
67: PRINT D$;"CLOSE";A$: TEXT
40 PRINT D$;"LOCK";A$: END
32765 REM IN THE NEXT STATEMENT, INSERT A CONTROL-D
BETWEEN THE QUOTES IN D$ = ""
32766 DIM A$(20): D$ = "": INPUT "TEXTFILE NAME ",A$: PRINT
D$;"OPEN";A$
32767 PRINT D$;"WRITE ";A$: POKE 33,30: LIST 0,32765:
PRINT D$;"CLOSE";A$: TEXT : END
63999 D$ = CHR$(4): INPUT "TEXTFILE NAME ";A$: PRINT
D$;"OPEN";A$: PRINT D$;"WRITE ";A$: POKE 33,30: LIST
0,63998: PRINT D$;"CLOSE";A$: TEXT : END

```



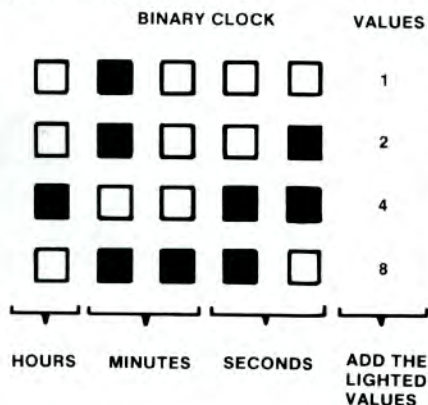
Apple Binary Clock

by Rip Toren
401 Cherry Lane E206
Laurel, MD 20810

How many times have you finished work on a project and turned off your display, but left your Apple on? Not very interesting, is it? I do that to save wear and tear on the power switch, and a big repair bill. Well, one day I was sitting in an office, and noticed a small square box with blinking lights. After a few moments of reflection, I realized that it was a binary clock. The display was much more enjoyable to watch while I waited for my appointment than a wall clock. At that point, I decided to write a binary clock program for my Apple. It was going to be an interesting exercise in cycle counting, I thought.

THE PROGRAM

The program is set up to run a binary clock, using LO-RES graphics for the display. The actual timing comes from a number of delay loops, each with successively finer resolution. The columns are read from left to right, with the first column being hours (1-12). Then follows tens of minutes, then minutes, etc. The values read top to bottom, with the top row being 2**0.



Examine the values to the right in Illustration 1 above. The bottom row lights each have a value of "8". The next row has a value of "4"... the next is "2"... and the top row is "1".

In Illustration 1, the correct time is 11:47:39 (by reading the white — lighted — blocks).

ENTERING THE PROGRAM

The machine language program can be entered directly into your Apple's memory using the instructions on the Letters page of this issue. Your first entries from the Monitor will be: **0800: 4C 23 08 47** etc. When the program has been entered into memory, press **CTRL C** to reenter Applesoft and type: **BSAVE BCLOCK, A\$800, L\$209** to save it to disk. You can then **BRUN BCLOCK** to run it.

IN OPERATION

When you **BRUN** the program, the screen will clear, and you will be asked for the initial setting. You enter the values for each column, and then press the **RETURN** key. The screen will clear, and then nothing will happen. The program is waiting for you to press another key to get it started. That is to allow it to sync as closely as possible to the initial values.

CYCLE COUNTING

As it turned out, the cycle counting was not as bad as I had anticipated. But let me digress for a moment. I have been doing 90% of my programming in assembly language since I purchased my Apple. Many times I have seen the claims of some package for how fast it runs due to the use of assembly language. Well, the first version of this program was written in Integer Basic. Unfortunately, I could not get it to run fast enough. It was always losing time. So, I was rather amazed at the magnitude of the delay routines that are used in the assembly language version. As the code stands now, there is approximately **0.31 second** of delay looping. That would appear to give it about a 50% edge in speed.

Rather than do a critical cycle analysis, I used a bracketing technique to find the optimal

settings. This would involve starting the process, measuring the error after 10 minutes, and making adjustments.

THE SETTINGS

The current settings give good results, at least on my system, but I don't believe that all Apples run at exactly the same number of microseconds per cycle. I have found values from **1.023** to **.9799269 microseconds** in different sources. So you may have to adjust the values to suit your own machine. The diagram below shows the structure of the timing loop itself:

```

loop on %X for n times (GROSS)
call DELAY with value (DELAY0)
end loop
call DELAY with value (DELAY1)
loop on %X for (DELAY2) times
    
```

When you are adjusting your clock, you will probably want it to run for 12 hours or so to be sure that you are within acceptable limits. As you can see from the listing, all the values that you will be using are located from **\$803-\$806**, so if you type in the machine code, you will only have to change these few once it is entered.

To find out how much change will take place when you adjust the value passed to the **DELAY** routine, the following formula is provided:

$$\text{CYCLE DELAY} = (2.5 * \%A * \%A) + (13.5 * \%A) + 13$$

As a humorous aside, we had the clock running one evening while we had company visiting. Someone asked what time it was, and since I had the Apple hooked to the TV set, I just changed the channel, read off the time and went back to the show we were watching. Our friends became very curious as to just what channel it was, and what the display had been.

Have Fun!



```

1000 * *****
1010 * BINARY CLOCK *
1020 * BY RIP TOREN *
1030 * COPYRIGHT (C) 1982 *
1040 * BY MICRO-SPARC INC *
1050 * S-C ASSEMBLER *
1060 * *****
1070 *
1080 *----- EQUATES -----
1090 Z.00 .EQ $00
002C- 1100 H2 .EQ $2C
002E- 1110 MASK .EQ $2E
0030- 1120 COLOR .EQ $30
00A2- 1130 Z.A2 .EQ $A2
00FD- 1140 Z.FD .EQ $FD SAVE LOC
00FE- 1150 Z.FE .EQ $FE SAVE LOC
00FF- 1160 Z.FF .EQ $FF SAVE LOC
1170 *
0200- 1180 INBUFF .EQ $0200
C000- 1190 KEYBD .EQ $C000
C030- 1200 SPKR .EQ $C030
FB1C- 1210 HLIN .EQ $FB1C
FB47- 1220 GBACAL .EQ $FB47
FB40- 1230 SETGR .EQ $FB40
FCAB- 1240 WAIT .EQ $FCAB
FD67- 1250 GETLNZ .EQ $FD67
FDED- 1260 COUT .EQ $FDED
1270 *
1280 *-----
1290 * START OF PROGRAM
1300 *
1310 .OR $800
0800- 4C 23 08 1320 JMP START
1330 *
1340 * THE FOLLOWING ARE THE DELAY
1350 * LOOP PARAMETERS
1360 *
0803- 47 1370 GROSS .DA $47
0804- 48 1380 DELAY0 .DA $48
0805- 01 1390 DELAY1 .DA $01
0806- 01 1400 DELAY2 .DA $01
1410 *
1420 * LOCATION FOR INDIVIDUAL DIGITS
0807- 00 1430 TBASE .DA $0
0808- 00 1440 HRS .DA $0
0809- 00 1450 MIN10 .DA $0
080A- 00 1460 MIN01 .DA $0
080B- 00 1470 SEC10 .DA $0
080C- 00 1480 SEC01 .DA $0
1490 *
1500 * THESE ARE THE FUDGE VALUES FOR
1510 * LEAVING THE CASCADE OF VALUES
    
```

```

1520 * EARLY
1530 *
080D- 10 1540 DELAY3 .DA $16
080E- 0C 1550 DELAY4 .DA $12
080F- 08 1560 DELAY5 .DA $08
0810- 04 1570 DELAY6 .DA $04
0811- 02 1580 DELAY7 .DA $02
0812- A0 C8 C8
0815- A0 CD A0
0818- CD A0 D3
081B- A0 D3 A0
081E- D4 C9 CD
0821- C5 BF
1590 MSG .AS -/ HH MM SS TIME??
1600 *****
0823- 20 5B FC 1610 START JSR $FC5B MON.HOME
1620 *-----
1630 * PRINT OUT THE TIME SET MSG
1640
0826- A2 00 1650 LDX $0
0828- BD 12 0B 1660 .1 LDA MSG,X
082B- F0 07 1670 BEQ SET.T
082D- 20 ED FD 1680 JSR COUT
0830- E8 1690 INX
0831- 4C 2B 0B 1700 JMP .1
1710 *
1720 * PROCESS THE TIME SET INPUT
0834- 20 67 FD 1730 SET.T JSR GETLNZ
0837- A0 00 1740 LDY $00
0839- 20 BF 09 1750 JSR ASD2B
083C- AD 8B 09 1760 LDA ASDLSB
083F- BD 0B 0B 1770 STA HRS
0842- 20 BF 09 1780 JSR ASD2B
0845- AD 8B 09 1790 LDA ASDLSB
0848- BD 09 0B 1800 STA MIN10
084B- 20 BF 09 1810 JSR ASD2B
084E- AD 8B 09 1820 LDA ASDLSB
0851- BD 0A 0B 1830 STA MIN01
0854- 20 BF 09 1840 JSR ASD2B
0857- AD 8B 09 1850 LDA ASDLSB
085A- BD 0B 0B 1860 STA SEC10
085D- 20 BF 09 1870 JSR ASD2B
0860- AD 8B 09 1880 LDA ASDLSB
0863- BD 0C 0B 1890 STA SEC01
0866- 20 40 FB 1900 JSR SETGR
1910 *
1920 * WAIT FOR KEYPRESS TO START
0869- 2C 00 C0 1930 KPRESS BIT KEYBD
086C- 10 FB 1940 BPL KPRESS
1950 *
    
```

continued on next page

Apple Binary Clock (Cont.)

```

1960 * INCREMENT SECONDS UNITS
086E- BD 30 C0 1970 SECO STA SPKR
0871- EE 0C 0B 1980 INC SECO1
0874- AD 0C 0B 1990 LDA SECO1
0877- C9 0A 2000 CMP #0A
0879- F0 04 2010 BEQ SETS10
087B- AD 0D 0B 2020 LDA DELAY3
087E- 4C DD 0B 2030 JMP DELAY
2040 *
2050 * INCREMENT SECONDS TENS
0881- A9 00 2060 SETS10 LDA #00
0883- BD 0C 0B 2070 STA SECO1
0886- EE 0B 0B 2080 INC SEC10
0889- AD 0B 0B 2090 LDA SEC10
088C- C9 04 2100 CMP #04
088E- F0 04 2110 BEQ MINO
0890- AD 0E 0B 2120 LDA DELAY4
0893- 4C DD 0B 2130 JMP DELAY
2140 *
2150 * INCREMENT MINUTES
0896- A9 00 2160 MINO LDA #00
0898- BD 0B 0B 2170 STA SEC10
089B- EE 0A 0B 2180 INC MIN01
089E- AD 0A 0B 2190 LDA MIN01
08A1- C9 0A 2200 CMP #0A
08A3- F0 04 2210 BEQ SETM10
08A5- AD 0F 0B 2220 LDA DELAY5
08AB- 4C DD 0B 2230 JMP DELAY
2240 *
2250 * INCREMENT MINUTES TENS
08AB- A9 00 2260 SETM10 LDA #00
08AD- BD 0A 0B 2270 STA MIN01
08B0- EE 09 0B 2280 INC MIN10
08B3- AD 09 0B 2290 LDA MIN10
08B6- C9 04 2300 CMP #04
08B8- F0 04 2310 BEQ HOURS
08BA- AD 10 0B 2320 LDA DELAY6
08BD- 4C DD 0B 2330 JMP DELAY
2340 *
2350 * INCREMENT HOURS
08C0- A9 00 2360 HOURS LDA #00
08C2- BD 09 0B 2370 STA MIN10
08C5- EE 0B 0B 2380 INC HRS
08C8- AD 0B 0B 2390 LDA HRS
08CB- C9 0D 2400 CMP #0D
08CD- F0 04 2410 BEQ CYCLEH
08CF- AD 11 0B 2420 LDA DELAY7
08D2- 4C DD 0B 2430 JMP DELAY
2440 *
2450 * RESET HOURS TO 1
08D5- A9 01 2460 CYCLEH LDA #01
08D7- BD 0B 0B 2470 STA HRS
08DA- 4C DD 0B 2480 JMP DELAY
2490 *-----
2500 * DELAY LOOPS
2510 *-----
2520 * DELAY FOR CASCADE FUDGE
2530 DELAY
2540 TAX
08DE- CA 2550 .1 DEX
08DF- D0 FD 2560 BNE .1
2570 *
2580 * MAJOR DELAY LOOP
08E1- AE 03 0B 2590 LDX GROSS
08E4- AD 04 0B 2600 .2 LDA DELAY0
08E7- 20 AB FC 2610 JSR WAIT
08EA- CA 2620 DEX
08EB- D0 F7 2630 BNE .2
2640 *
2650 * MINOR DELAY LOOP
08ED- AD 05 0B 2660 LDA DELAY1
08F0- 20 AB FC 2670 JSR WAIT
2680 *
2690 * FINE TUNE DELAY
08F3- AE 06 0B 2700 LDX DELAY2
08F6- CA 2710 .3 DEX
08F7- D0 FD 2720 BNE .3
08F9- 4C 00 09 2730 JMP PROCESS
2740 *-----
08FC- 00 2750 S.COL .DA #0 START COLUMN
08FD- 00 2760 S.ROW .DA #0 START ROW
08FE- 00 2770 TH.ROW .DA #0 THIS ROW BEING PROCESSED
08FF- 00 2780 VALUE .DA #0 VALUE FOR THIS COLUMN
2790 *-----
0900- A9 29 2800 PROCESS LDA #29
0902- BD FC 0B 2810 STA S.COL
0905- A2 05 2820 LDX #05
0907- BD 07 0B 2830 .1 LDA TBASE,X
090A- BD FF 0B 2840 STA VALUE
090D- 38 2850 SEC SET THE COL.
090E- AD FC 0B 2860 LDA S.COL VALUE FOR
0911- E9 0B 2870 SBC #08 THIS VALUE
0913- BD FC 0B 2880 STA S.COL
0916- 20 1F 09 2890 JSR PROC.V BREAK INTO
0919- CA 2900 DEX BITS
091A- D0 EB 2910 BNE .1
091C- 4C 6E 0B 2920 JMP SECO
2930 *
2940 *
2950 * PROCESS ONE VALUE A BIT AT A TIME
091F- B6 FF 2960 PROC.V STX Z.FF
0921- A9 02 2970 LDA #02 START ROW FOR
0923- BD FD 0B 2980 STA S.ROW EACH VALUE
0926- A2 04 2990 LDX #04
3000 *
3010 * GET EACH BIT VALUE
0928- 20 43 09 3020 PROC.B JSR SET.COLOR
092B- AD FD 0B 3030 LDA S.ROW
092E- BD FE 0B 3040 STA TH.ROW
0931- 20 58 09 3050 JSR DISP
0934- 18 3060 CLC
0935- AD FD 0B 3070 LDA S.ROW
0938- 69 05 3080 ADC #05
093A- BD FD 0B 3090 STA S.ROW
093D- CA 3100 DEX
093E- D0 EB 3110 BNE PROC.B

```

```

0940- A6 FF 3120 LDX Z.FF
0942- 60 3130 RTS
3140 *
3150 SET.COLOR
0943- 6E FF 0B 3160 ROR VALUE
0946- 90 0B 3170 BCC .1
0948- EA 3180 NOP FOR TIMING
0949- A9 77 3190 LDA #77
094B- B5 30 3200 STA COLOR
094D- 4C 57 09 3210 JMP .2
3220 *
0950- A9 11 3230 .1 LDA #11
0952- B5 30 3240 STA COLOR
0954- 4C 57 09 3250 JMP .2
3260 *
0957- 60 3270 .2 RTS
3280 *
3290 * ROUTINE TO INTERFACE TO THE
3300 * LO-RES GRAPHICS IN MONITOR
3310 *
095B- B6 FD 3320 DISP STX Z.FD SAVE %X
095A- AD FC 0B 3330 LDA S.COL
095D- 18 3340 CLC COMPUTE END
095E- 69 04 3350 ADC #04 COL.
0960- B5 2C 3360 STA H2
0962- A2 03 3370 LDX #03 LOOP COUNT
0964- AD FE 0B 3380 .1 LDA TH.ROW
0967- B6 FE 3390 STX Z.FE
0969- 20 47 FB 3400 JSR GBACAL
096C- A9 F0 3410 LDA #F0
096E- B5 2E 3420 STA MASK TOP HALF OF
0970- AC FC 0B 3430 LDY S.COL ROW
0973- 20 1C FB 3440 JSR HLIN
0976- A9 0F 3450 LDA #0F
0978- B5 2E 3460 STA MASK BOTTOM HALF
097A- AC FC 0B 3470 LDY S.COL OF ROW
097D- 20 1C FB 3480 JSR HLIN
0980- EE FE 0B 3490 INC TH.ROW
0983- A6 FE 3500 LDX Z.FE
0985- CA 3510 DEX
0986- D0 DC 3520 BNE .1 END LOOP
0988- A6 FD 3530 LDX Z.FD RESTORE %X
098A- 60 3540 RTS
3550 *
3560 * THIS ROUTINE INTERPRETS THE
3570 * INPUT LINE WITH THE INITIAL
3580 * SETTINGS
3590 * IT WILL CONVERT A 5 DIGIT ASCII
3600 * VALUE SEPERATED BY BLANKS INTO
3610 * A BINARY NUMBER IN ASDLSB,ASMSB.
3620 * MULTIPLE CALLS WILL MOVE DOWN THE
3630 * INPUT BUFFER.
3640 * CARRY WILL BE SET IF GREATER
3650 * THAN 32767.
3660 *
098B- 00 3670 ASDLSB .DA #0
098C- 00 3680 ASMSB .DA #0
098D- 00 3690 ASDTMP .DA #0
098E- 00 3700 .DA #0
098F- B9 00 02 3710 ASD2B LDA INBUFF,Y
0992- C9 A0 3720 CMP #A0
0994- D0 03 3730 BNE ASDG0
0996- C8 3740 INY
0997- D0 F6 3750 BNE ASD2B
0999- A9 00 3760 ASDG0 LDA #00
099B- BD BC 09 3770 STA ASMSB
099E- 20 BB 09 3780 JSR NU.DGT
09A1- B0 11 3790 BCS 1.09AE
09A3- BD BB 09 3800 STA ASDLSB
09A6- A2 04 3810 LDX #04
09AB- 20 BB 09 3820 NX.DGT JSR NU.DGT
09AB- B0 06 3830 BCS RTSS
09AD- 20 CC 09 3840 JSR MULT.10
09B0- CA 3850 DEX
09B1- D0 F5 3860 BNE NX.DGT
09B3- 60 3870 RTSS
3880 *
3890 * 1 IS THE VALUE OF BAD INPUT
09B4- A9 01 3900 1.09AE LDA #01
09B6- BD BB 09 3910 STA ASDLSB
09B9- D0 FB 3920 BNE RTSS
09BB- B9 00 02 3930 NU.DGT LDA INBUFF,Y
09BE- C9 B0 3940 CMP #B0
09C0- 90 0B 3950 BCC ASDRTN
09C2- C9 BA 3960 CMP #BA
09C4- B0 04 3970 BCS ASDRTN
09C6- 29 0F 3980 AND #0F
09C8- CB 3990 INY
09C9- 60 4000 RTS
4010 *
09CA- 38 4020 ASDRTN SEC
09CB- 60 4030 RTS
4040 *
4050 MULT.10
09CC- BD BD 09 4060 STA ASDTMP
09CF- AD BC 09 4070 LDA ASMSB
09D2- 48 4080 PHA
09D3- AD BB 09 4090 LDA ASDLSB
09D6- 48 4100 PHA
09D7- 0E BB 09 4110 ASL ASDLSB
09DA- 2E BC 09 4120 ROL ASMSB
09DD- 0E BB 09 4130 ASL ASDLSB
09E0- 2E BC 09 4140 ROL ASMSB
09E3- 68 4150 PLA
09E4- 6D BB 09 4160 ADC ASDLSB
09E7- BD BB 09 4170 STA ASDLSB
09EA- 68 4180 PLA
09EB- 6D BC 09 4190 ADC ASMSB
09EE- BD BC 09 4200 STA ASMSB
09F1- 0E BB 09 4210 ASL ASDLSB
09F4- 2E BC 09 4220 ROL ASMSB
09F7- AD BD 09 4230 LDA ASDTMP
09FA- 6D BB 09 4240 ADC ASDLSB
09FD- BD BB 09 4250 STA ASDLSB
0A00- A9 00 4260 LDA #00
0A02- 6D BC 09 4270 ADC ASMSB
0A05- BD BC 09 4280 STA ASMSB
0A0B- 60 4290 RTS

```



Amp-L-Soft

Amp-L-Soft

Paul Irwin
Dynamic Business Systems
P.O. Box 1264, Station B
Ottawa, Ontario
K1P 5R3

Have you ever wished that Applesoft had a few choice commands to make your programming easier? Like a tone generator, or an INPUT that lets you enter commas, or a fast substring search? And while we're at it, why not have an easier way to call system routines in the Monitor or DOS? The program should be just as easy to use as Tag Sort (Nibble Vol. 2 #6); in fact we want Tag Sort to be included.

Well, you can have it working for you on your own Apple II so read on!

To use Amp-L-Soft, you need an Apple II with at least 32K, Applesoft in ROM (a RAM card with FPBASIC is great!), and a disk drive.

HOW TO USE Amp-L-Soft

The entire package comes in a binary file called Amp-L-Soft. From the keyboard, you can type BRUN Amp-L-Soft. From a program, you need three statements:

```
10 DEF FN AD(X) = PEEK(X) +  
    256*PEEK(X+1)  
50 PRINT CHR$(4); "BLOAD AMP-L-SOFT"  
60 CALL FN AD(FN AD(999)-79)
```

Be sure you load and call Amp-L-Soft before any DIMension statements. Do it at the beginning of the program, or better yet, in the HELLO program.

If you want to remove Amp-L-Soft from your memory and reclaim its space, then use the FP command.

When you load and run this package, it is located at \$4750 where it can keep below the DOS of a 32K system. Amp-L-Soft relocates itself forward in memory if you have a 48K system, and then moves its routines to \$80C and sets the ampersand hook at \$3F5.3F7.

You can use any of the ampersand routines present by a simple statement from Applesoft. You always use the same form:

&x(parameter list)

where **x** is the letter code for the routine:

```
TAGSORT... &A($$, T%, F, L)  
SUBSTRING... &B(A$, B$, P%)  
SYSCALL... &C(A, X, Y, R, C%)  
INPUT... &I(IN$)  
TONE... &N(P, D)
```

Currently, you have **A**, **B**, **C**, **I**, and **N** defined for your use. Just give each routine the parameters it needs, and it will do what you want.

Any **parameter** you give a routine is one of three kinds. If it is a **value**, you can give it any legal Applesoft expression like **5** or **FN L0(C1)**. If it is a **variable**, then you can give it either a simple variable like **A\$** or **C%**, or else an array variable like **T%(5)** or **D(34, 2)**. But, if

it is an **array name**, then just give it **A\$**, say, for an array DIMensioned as **A\$(N)**. Keep values within the ranges given. With variables, use only the type shown: **integer**, **real**, or **string**. You can use any routine by passing it the parameters — values, variables, and names — it requires.

HOW TO LOAD Amp-L-Soft

Amp-L-Soft is made up of 2 assembly language program segments and 1 Basic program called **BOOTER**. If you are entering the machine language directly into memory, here's how to **a) Enter and Save each segment**; and **b) Merge the two segments into the final program**.

The entry of each program segment is begun by typing: **CALL-151** to enter the Monitor. At that point, the screen will display an asterisk (*).

Now **look for the first place in the program where machine code appears** in the second/third/fourth columns immediately following the colon. For example, in **LOADER**, the machine code begins at memory location **\$4750 (hex)** with the codes **A9 0F**, and continues with **85 51 A9 00** etc.

Now, **type in the starting memory address** of the code; in the example, this is memory address **\$4750 (hex)**. Now you can begin entering the code by typing:

4750:A9 0F 85 51 A9 00 85 50 A5 67 etc.

For additional information on how to enter machine language code, refer to the instructions on the Letters page of this issue.

When you have finished typing in the code, you can save it to disk with the following command:

BSAVE LOADER, A\$4750, L\$AC

In like fashion, enter the other segment and save it with the command:

**BSAVE AMP-L-SOFT UTILITIES,
A\$80C, L\$642**

Now, you can **Merge** the program segments with the following steps. They're a little complicated so please follow them exactly. With the two routines now stored on disk, type:

**BLOAD LOADER, A\$4750
BLOAD AMP-L-SOFT UTILITIES, A\$480C**

Now, all the routines are merged in memory and must be saved as a block. This can be done with the command:

BSAVE AMP-L-SOFT, A\$4750, L\$6FE

It is this last program, Amp-L-Soft, which will be operational when you run the Applesoft Basic Loader called **BOOTER**. It is this program which you would transfer to other disks in order to build turn-key applications.

For the moment, let Amp-L-Soft stay comfortably on the disk while you type in the Applesoft **BOOTER** program (Listing #1). This program loads Amp-L-Soft, relocates it to lower memory, and then sets up the Apple-

soft Pointers to accept your own programs. It is suggested that this **Booter** program be made the **HELLO** program on any disk which you wish to use for automatically loading Amp-L-Soft.

Having typed in **BOOTER**, now save it on disk.

Now you can run **BOOTER** and it will set everything up to try out the program with the Amp-L-Soft demonstration (Listing #2).

Let's look at each of these routines and see what you can do with them.

SYSCALL

How often have you seen a routine in Monitor, Applesoft, or DOS and wished you could have used it without having to program the machine language needed to call it? For instance, to print a number in hex, the monitor has a couple of neat routines, but you have to load the byte you wish to convert into the A-register first. But you couldn't do that from Applesoft, so you had to do some tricky programming.

A good trick is to have one **calling routine** written in machine language to **load the registers, make the system call**, and then return the carry flag to your Applesoft program. Why the carry flag? Because some system calls will tell you if an error occurred or not by using the carry flag. A famous example of this is **RWTS** in the heart of DOS. **By having a call routine like this, you can use many of the system calls that are unavailable from Applesoft.**

A syscall routine can load the A-register, the X-register, and the Y-register with the values passed to it from your Applesoft program; then **call the memory address** you also passed to it from the Applesoft program; and finally **pass the carry flag** back to your program. So, when using **SYSCALL**, you give it five parameters: **A-reg**, **X-reg**, **Y-reg**, **call address**, and the **carry flag**. The first four are values, so any Applesoft expression can be used. But the fifth parameter — the carry flag — must always be an integer variable. As an example, here's how you can **print a hex number**:

&C(127, 0, 0, 64986, C%)

This prints **7F** — the hex equivalent of **127** — for you. In this example, the **A-reg** was set to **127**, the **X-reg** to **zero**, the **Y-reg** to **zero**, and the call address was set to **64986** in the Monitor. The integer variable — **C%** here — must always be provided to capture the carry flag, regardless.

You can see a number of **SYSCALL sequences** in Table 1. Most of the routines are in the Monitor, but the **RWTS** sequence is in DOS, and the three bit handling routines are included with **SYSCALL**.

These are the three little routines we call **Little Bit Pickers**. Whenever you want to get just one bit from a byte, or put one bit into a byte, these are the calls to make. The first one **reads a bit** from a byte into an integer variable and the other two **write a bit** into a byte. Now you can work quickly and easily with the **track bit maps** in DOS, or the most significant bit — bit 7 — of character bytes. The bit handling that we found so awkward from Applesoft is

continued on next page

Amp-L-Soft (Cont.)

now much easier with SYSCALLs to the Little Bit Pickers. See the program **AMP-L-SOFT DEMO 2** for some examples of their use.

Use SYSCALL whenever you want to call a routine that passes parameters via the 6502 registers. Set the A-reg, the X-reg, the Y-reg, and the call address. Always provide an integer variable to capture the carry flag. Use the Little Bit Pickers to get and put bits of a byte according to the examples in **Table 1**.

SUBSTRING

This is a function that **searches one string and tells you where a particular substring is located**. You can use it to **edit input strings** from disk files and keyboard, to **lookup code numbers**, and to **format output strings**. It doesn't do anything that the **MID\$** function in a loop can't do, but with it you can do the job easier and faster.

Just give Amp-L-Soft three variables: a **string to be searched**, a **target substring**, and the integer position:

&B(OB\$, TA\$, I%)

The search starts from the position given in **I%**, usually **one**. When the target string **TA\$** is first found in the object string **OB\$**, the function returns to you with its position in **I%**. But, if the target string can't be found, the position **I%** is returned to you as **zero**.

For example:

```
I% = 1
A$ = "THE QUICK BROWN FOX JUMPED."
B$ = "O"
&B(A$, B$, I%)
PRINT I%
13
```

Now you can continue the search by entering a new starting value for **I%**.

```
I% = 14
&B(A$, B$, I%)
PRINT I%
18
```

And again

```
I% = 19
&B(A$, B$, I%)
PRINT I%
0
```

We start with the pointer at the first character of the string: **I% = 1**. The strings are given values: **A\$** is the object of the search, and **B\$** is the target. The target here is just one character — "O". After the first call, you can see that the pointer is at the thirteenth character; we found the **first "O"** in **A\$**. By changing **I%** from **13** to **14**, we continued the search. On the second call, we found that the **second "O"** is the **nineteenth** character. Again to continue, we increased **I%** by one — to **20**. On the third call, you can see that the pointer became zero because there is **no third "O"** in **A\$**. Our search has ended.

So, to start a search, set the pointer to **one**. To continue the search, increment the pointer by one after each "hit". Whenever a search results in a zero pointer, the search is completed.

Table 1
SOME USEFUL SYSCALLS

Read bit from a byte:

```
BR = FN AD(FN AD(1014)+47)+72
B% = byte to read: 0 . . 255
N% = bit number: 7 . . 0 (0 = LSB)
&C(B%, N%, 0, BR, C%)
result (0 or 1) = C%
```

Turn ON bit in a byte:

```
BN = FN AD(FN AD(1014)+47)+77
B% = byte to be changed: 0 . . 255
N% = bit number: 7 . . 0 (0 = LSB)
POKE 60, B%
&C(0, N%, 0, BN, C%)
B% = PEEK(60)
```

Turn OFF bit in a byte:

```
BF = FN AD(FN AD(1014)+47)+87
B% = byte to be changed: 0 . . 255
N% = bit number: 7 . . 0 (0 = LSB)
POKE 60, B%
&C(255, N%, 0, BF, C%)
B% = PEEK(60)
```

Print a nybble in hex:

```
B% = nybble to print: 0 . . 15
&C(B%, 0, 0, 64995, C%)
```

Print a byte in hex:

```
B% = byte to print: 0 . . 255
&C(B%, 0, 0, 64986, C%)
```

Print an address in hex:

```
B = address: 0 . . 65535
&C(FN HI(B), FN LO(B), 0, 63809, C%)
```

Print a number of spaces:

```
SP% = number of spaces: 0 . . 230
&C(0, SP%, 0, 63818, C%)
```

Print a character via COUT:

```
A$ = character to print, in ASCII
A = ASC(A$) + 128
&C(A, 0, 0, 65005, C%)
```

Print a character to screen, via COUT1:

```
A$ = character to print, in ASCII
A = ASC(A$) + 128
&C(A, 0, 0, 65008, C%)
```

Direct disk access:

This uses the work space in **AMP-L-SOFT** as a buffer, but remember it is clobbered by using **TAGSORT**. The Input-Output Block in DOS is used. The SYSCALL uses **RWTS** (DOS manual, p.94). Any error will set **E% > 0** with the **RWTS** code.

```
WS = FN AD(1014)+1508
IB = FN AD(FN AD(999)):
REM START OF IOB
POKE WS+B, X%:
REM PUT any byte B: 0 . . 255
POKE IB+1, SL*16:
REM SLOT: 1 . . 6
POKE IB+2, DR:
REM DRIVE: 1 or 2
POKE IB+3, V:
REM VOLUME: 0 . . 254
POKE IB+4, TR:
REM TRACK: 0 . . 34
POKE IB+5, SE:
REM SECTOR: 0 . . 15
POKE IB+8, FN LO(WS):
REM BUFFER-lo
POKE IB+9, FN HI(WS):
REM BUFFER-hi
POKE IB+12, OP:
REM 1=read, 2=write, 4=format
&C(FN HI(IB), 0, FN LO(IB), 985, C%)
E%=0: IF C%>0 THEN
E%=PEEK(IB+13)
X%= PEEK(WS+B):
REM GET any byte B: 0 . . 255
The error codes of E% are:
0= OK, no error.
16= Write protected,
can't write.
32= Volume Mismatch:
PEEK(IB+14) found.
64= Drive error. Could be
anything.
128= Read error. Tried 48
times.
```

SUBSTRING APPLICATIONS

One place where **SUBSTRING** is very useful is in handling Menus. After you display the menu choices on the screen, GET each key-stroke from the keyboard with a subroutine:

```
3100 C$ = "IJKMQ" : REM MENU CHOICES
3110 GET A$
3120 I%=1: &B(C$, A$, I%)
3130 IF I% = 0 THEN 3110
3140 IF I% = 5 THEN RETURN
3150 ON I% GOSUB 100, 200, 300, 400
3160 GOTO 3110
```

You could also use this for **HIRES** cursor control. For example, the four cursor keys could each call its own subroutine and the fifth choice — "Q" — will return to the caller.

In another example, when **editing input strings** from the keyboard or disk, you can use Amp-L-Soft to search for commands, keywords, or delimiters. For instance, you could write an Adventure game that lets the player enter English-language sentences and then parses them quickly (with **SUBSTRING** calls) to extract the valid commands.

The **SUBSTRING** also lets you do **fast character code conversion**. For example, you can make up a string in a code sequence,

such as the "IJKMQ" sequence above. Now searching for a single ASCII character will return the code in the pointer. To convert back to ASCII from the code, just use the **MID\$** function. The **MID\$** is the inverse of **SUBSTRING**:

```
&B(OB$, TA$, P%) : REM GIVEN P%
TA$ = MID$(OB$, P%, L%) : REM GIVES TA$,
LENGTH L%
```

where **OB\$** is the object string, **TA\$** is the target string, and **P%** is the pointer. This makes the two functions an ideal match for your enciphering and deciphering needs.

The **SUBSTRING** complements the **MID\$** function in Applesoft. With the **MID\$**, you create a target string by giving the object string and the pointers to the substring. But with **SUBSTRING**, you find the pointer to the substring by giving it the object string and the target string. The pointer must be in an integer variable and it must have a value of one for a complete search. The returned value of the pointer must be incremented by one to continue the search for more substrings. A returned pointer value of zero ends any search for substrings and indicates that none were found.

TONE

You can make many different sounds on the Apple II, many more than just the **CTRL/G** beep. If you have the so-called Red Book — the original Apple II Reference manual — then you probably know about Lutas' algorithm given on page 45. It uses two parameters — **period and duration** — to control the sound of a single tone. Now you can use Lutas' algorithm to make your own sounds simply by calling Amp-L-Soft's **TONE** routine:

&N(P, D)

where **P** is the **period** value and **D** is the **duration** value.

The **period** you give it must be a value from **zero to 255**. If you pick zero, then the Monitor **WAIT** routine will kill some time. Otherwise with a period greater than zero, you will hear a tone. Using a small period gives you a high-pitched tone, while using a large period gives you a low-pitched one.

You can control the **length** of the tone with the **duration**. Give it a value from **one to 255**: the longer the duration, the longer the tone. If you want a **rest note**, use a zero period with a suitable duration. You will find that any duration you give doesn't always give the same length of a rest as the length of a tone.

Experiment with different periods and durations. You can find sounds to use for special effects in **games, alarms, and tunes**. Use different tones for different prompts: a ready tone, an error tone, a warning tone, and an in-process (busy) tone. Play with some loops, some iterations, or anything else you can think of; discover new sounds! Put some personality into your programs!

INPUT

?EXTRA IGNORED. Look familiar? The string **INPUT** statement in Applesoft just won't accept commas, quotes, or colons. Do not enter them. They do not compute. Bah, humbug!

The routine that Applesoft uses to input a string is called **INLIN** and is located in Applesoft at **\$D52C**. By calling this routine **directly**, we can get the input string **before** Applesoft edits it for delimiters — commas, quotes, or colons — and then assign the characters to the string variable ourselves. We have to create space for the string in memory and move it there from the input buffer. Then we can hook up the descriptors in the string variable. This way, we do just what Applesoft's string **INPUT** does, but without the extra unwanted editing.

Amp-L-Soft **INPUT** is easy to use. Just pass one string variable:

&I(INS)

which is returned to you with the string from the system input. Because you are using the **system** input, it also **works with disk or any other device**, just the Applesoft's **INPUT** statement.

Now we can **INPUT** strings such as, **"THIS, IS GREAT FOR : TEXT ENTRY; ETC."** But why, you may ask, does Applesoft's **INPUT** behave in such a manner? What method can there be in this madness?

Well, Applesoft lets you enter **several** variables on one input line. You can write a statement such as:

100 INPUT A\$, B, C\$

and Applesoft uses the commas in the input buffer to tell it which variables go where. In this example, Applesoft expects only **two** commas (for three variables) in the buffer. The variable preceding the first comma is assigned to **A\$**; the variable **between** the commas is assigned to **B** (with an implied **VAL** function); and the variable **after** the second comma is assigned to **C\$**.

When Applesoft reads from disk for example, all the fields of a record can be assigned with a single **INPUT** statement. If you are learning about computers, this is a great way to work with the fields and records of files relatively painlessly.

With Amp-L-Soft's **INPUT** that allows commas, we can accept complete strings without the dreaded **?EXTRA IGNORED** message. Use Amp-L-Soft's **INPUT** routine for keyboard string inputs and complete record inputs from disk or other devices. Then just use Applesoft's **INPUT** statement for numeric input from the keyboard.

TAGSORT

Sooner or later, you'll need a good, fast sort. You may need to **sort a disk file**; to **search key fields** of a random access file, or just to properly **sequence an array** for fast lookup. When you do, you'll find that Amp-L-Soft's **TAGSORT** is a powerful solution for all these needs.

You can find a more complete description of **TAGSORT** in a companion article **Amper Jump & TSORT** (Nibble Vol. 2 #6). There, you can see how to call the sort and review the main ideas of its use. This version of **TSORT** has been enhanced to handle arrays which are larger than 800 elements (it is limited only by memory capacity of your Apple). It also properly sequences strings where the shorter string is identical to the same number of beginning characters in a longer string.

To sort a file or list, you must put the elements to be sorted into a **one-dimensional string array**. If it is a disk file that you are sorting, put only the **sort field** of each record into the string array. The array size is limited by the memory available, so if you find yourself running short of memory, pick up only enough characters (from the selected field) to do the job.

You will need a **second, one-dimensional array**, **DIMensioned the same size** as the string array. Make this second array an **integer** array, and load it with the **physical sequence numbers** of the records which you wish to sort. This integer array is your **Tag Array**. To initialize it, you put in the physical element numbers, (which usually range from one to the last element number, i.e.)

FOR I=1 TO N : T%(I) = I : NEXT

With disk files, you can use the **record numbers**.

After the sort is completed, you will have the physical record numbers in the final sorted sequence in the Tag Array. You can use these tags to access the records in the proper, sorted sequence. A tag array must be **DIMensioned**, regardless of whether you actually use it.

With the string and tag arrays loaded, you can sort whenever you wish, just by the call:

&A(SS, T%, F, L)

Give it the names of the arrays (**SS** and **T%**) as the first two parameters. Then give the value

of the **first element (F)** you want sorted; usually **one**. Finally, give it the value of the **last element (L)** you want sorted; usually the number of records in the file or elements in the array.

After the sort, you'll find the string array in ascending (sorted) sequence. If you are working with a disk file, you can find the physical record numbers (in the same sort sequence as the strings) in the Tag Array. The first tag **contains the old record number** of the first record to write to the new (sorted) file; the second tag contains the number of the second record to write to the new file; and so on.

If you are working with a random access file, you can read the old records using random access tags, and then write the new file sequentially (in the new sorted sequence). Regardless of which data structure you are sorting however, you'll find that tags can often be used to speed things up.

HOW AMP-L-SOFT WORKS

JUMP

The heart of Amp-L-Soft is a routine called **JUMP**. When you load and call Amp-L-Soft, this routine together with the five working routines are loaded into the **\$80C**. **EFF** area **below your program**. Whenever you use an **'&'**, this routine looks at your command code and finds the routine you requested. The **'&'** that you used tells Applesoft to pass control to memory location **\$3F5**. At that location, there is a **JMP** instruction to **\$81C** in **JUMP**. Depending on the code you used, one of the Amp-L-Soft routines is then selected for execution.

JUMP TO TONE

For instance, let's follow a call to the **TONE** routine. Suppose you type

& N(76, 224)

Applesoft sees your ampersand and jumps to **\$3F5**. There, the instruction

JMP \$81C

passes control to the **AMPER** routine in **JUMP**. If you look at the listing for **JUMP**, you will see that it **checks for a letter from Applesoft text**. The letter must be from **"A"** to **"Z"**, otherwise you get a **?SYNTAX ERROR** message. We have an **"N"**, so it's OK. It does the same thing for an open bracket — **"("**. That's OK, too. Then it remembers the letter **"N"** it accepted before and **converts it into a number — a 13**. (An **"A"** would have become a **zero**, a **"B"** would have become a **one**, and so on.)

This number is **doubled** (with an **ASLA** instruction) to a **26** and **used as an index (X-reg)** to **fetch the third address from the table** (at **\$84B**. **84C**) to the **JMP** instruction. **When the JMP instruction is reached, it has the address of the tone routine — \$972 — to jump to**. The **TONE** routine is just where we wanted to direct the program.

The **TONE** routine picks up its parameters (more about that below) and makes the sound you wanted. When finished, **TONE** jumps to an Applesoft routine called **DATA**. **DATA** makes sure the current statement is finished before returning to Applesoft's interpreter to await your next instruction.

continued on next page

Amp-L-Soft (Cont.)

So, JUMP receives control from the **& Jump** (at **\$3F5**) in Page Three; figures out which routine you want according to the letter code; and then jumps to your chosen routine. **It expands one ampersand instruction into 26 possible commands.** Any commands that we don't use, like the **D** or **E**, have Applesoft's **?SYNTAX ERROR** message routine selected; only the routines we actually have loaded can be selected by the letter code.

LOADING YOUR OWN ROUTINES

If you want to use some other ampersand routine at the same time you are using Amp-L-Soft, you can do so. Just choose one of the **unused letters**, **Z**, for example, and **POKE** the address of your routine into the **Jump Table** in JUMP. You can find the POKE addresses to use in the program listing. You then call the routine by:

&Z(any parameters)

assuming you chose "**Z**" as its letter code. **The opening bracket is a must.** Follow the bracket with any symbols and parameters required by the routine, **(except for the ampersand itself)**. Don't repeat "&", since you only need one & symbol to activate a routine.

WHY USE THE "&"?

You may wonder why the ampersand feature is used at all, or why we would go to the trouble of extending it with a table. After all, you could call the TONE routine directly with:

CALL 2418, P, D

if TONE had a **JSR CHKCOM** statement at the beginning to gobble the first comma. The ampersand method of calling machine language subroutines appears more complicated (something to avoid!) at first.

But, you can use ampersand calls a lot easier than calling them through Applesoft. All you need is a **single letter** for each routine — a letter that's easier to remember than a memory location number when you're programming.

And don't forget, you would probably have to convert the memory location number from hex to a decimal number in order to CALL it. Then you'd have a problem if you changed or relocated anything. You'd have to search through all your programs and change one address for another at each CALL affected. It's much better to be able to make all the changes in the Amp-L-Soft files only, and leave all the programs alone. Then, just load the Amp-L-Soft program on each disk, without worrying about what the programs are doing. For these reasons, you will find Amp-L-Soft easy to use, expand and maintain.

APPLESOFT ROUTINES — HOW IT WORKS

Remember the three kinds of parameters we looked at (Values, Variables, and Array Names) way back at the beginning? If you want to know how the routines access them, then this section's for you. It's not too difficult to follow if you have been doing any machine language programming. You can even customize your Amp-L-Soft package to include your own routines.

Here's how to use Applesoft in accessing parameters.

If you do a disassembly list starting at **\$00B1** in Page zero, you will find a short routine that **fetches a byte** from somewhere, using the address at **\$00B8. 00B9**, to the accumulator. You can see that it advances the pointer at **\$00B8. 00B9** first. Then it gets the first printable character (byte), and finally converts it if it is numeric. The routine (**\$00B1**) is called **CHRGET**, and the pointer it uses (**\$00B8. 00B9**) is called **TXTPTR**.

Now, if you are using Applesoft in its command mode (from the keyboard) **TXTPTR** puts data into Page Two (where it is found when you list **CHRGET**). Whenever you are executing a **program** however, **TXTPTR** is used by Applesoft to point into your program while Applesoft reads the program text and interprets what it finds.

When a **machine language routine** is called (in an Applesoft statement), **TXTPTR** points to the next character in your program that follows the **CALL** address. If that call is an ampersand, then **TXTPTR** contains the **address of the first character following "&" in your program**. The character itself is in the **A-register**. The trick to handling parameters in an Applesoft program is to know where **TXTPTR** is at all times.

We can use **existing** Applesoft routines to edit the bytes we pick up at **TXTPTR**. We can exit through a routine called **SNERR** if we don't like what we get. **SNERR** prints the dreaded **"?SYNTAX ERROR"** message and goes to Applesoft's command mode, halting the program.

You can see the **JUMP** routine do this. The first thing **JUMP** does when executed is to save the **A-register** and then perform a **JSR ISLETC**. This routine checks that the **A-register** has a letter — "**A**" to "**Z**" — in it. If not, the carry is cleared and we know to exit through the **SNERR** routine. If the routine being called is valid, we use **CHRGET** to get the next character. Whenever **CHRGET** returns to us, it has the next character in the **A-register**. Now **JUMP** does the same thing again, but it uses **CHKOPN** to check that the current character is an open bracket. This time, **CHKOPN** will exit through **SNERR** or **GETCHR** depending on what it finds.

Now let's see what **JUMP** does when it is ready to jump to your chosen routine. It executes a **JSR CHRGOT** immediately before the **JMP**. **CHRGOT** is a secondary entry point to **CHRGET**. **CHRGOT** does not advance **TXTPTR**, so it gets the **same** character that the previous **CHRGET** fetched. Then the jump is made to your chosen routine with the **A-register** containing the current character — the one immediately following "(" in your program — and **TXTPTR** points to it as well.

Any ampersand routine, whether it is reached via **JUMP** or directly from **\$3F5**, has **TXTPTR** pointing to the first character in the parameter list, and that character — edited by **CHRGET** — is in the **A-register**. **JUMP** is transparent to any ampersand routine.

You pick up a parameter by executing a **JSR** to an Applesoft routine that has its own **JSR CHRGET** and puts some value or pointer in a convenient location for you to use. Most of these routines require that **TXTPTR** be set to the first character of the parameter, and then they will exit through **CHRGET** to leave **TXTPTR** set to the delimiter (like a comma) which follows. This makes things easy for us.

We have already used one of these routines, **CHKOPN**. It would have exited to

SNERR if it didn't accept the parameters it was given. It left **TXTPTR** pointing to the first byte following the "(" . Between the parameters we used a similar one — **JSR CHKCOM**. If this routine is called with **TXTPTR** on a **comma**, it returns via **GETCHR**, ready for the next parameter, but it will go to **SNERR** if a comma isn't present. So, you can use **CHKCOM** between parameters to get from the end of one parameter to the beginning of another.

THE SYSCALL EXAMPLE

Look at the **SYSCALL** listing; it has a few **JSR CHKCOM**'s in it.

Okay, let's get some parameters.

GETBYT picks up an expression. When you want a **value between zero and 255**, **GETBYT** will get it from the expression and deliver it in the **X-register**. **SYSCALL** uses three of them to pick up the register values. **GETBYT** is at **\$E6F8**.

FRMNUM also picks up an expression but it delivers the expression to the **floating-point accumulator — FAC**. So, you can get a value of any size, not just a single byte value. If you immediately follow **JSR FRMNUM** by **JSR GETADR**, then your value in **FAC** is converted to an address in **LINNUM** (**\$50. 51**) in **LO. HI** format. This is how **SYSCALL** picks up the call address as its **fourth** parameter. **FRMNUM** is at **\$DD67**; **GETADR** is at **\$E752**.

PTRGET is used to access strings and parameters. It will pick up any variable and put its address in **VARPNT** (**\$83. 84**). At that address you will find the **floating point number, integer, or string descriptor** of the variable named at **TXTPTR**. In **SYSCALL** for example, **PTRGET** is used to get the address in which to put the value of the carry flag. It assumes an integer variable, so it puts a zero in the first location (integer high byte) and then a zero or one in the next location (integer low byte). **PTRGET** is at **\$DFE3**.

Look at some examples of using **PTRGET** to read and write to strings. See **SUBSTRING** for examples of reading strings. See the **INPUT** routine as an example of writing a string by first creating space for it. You use **PTRGET** to find the address of the string's descriptor: length, address-low, address-high. Pick up the descriptor using **VARPNT** and then move it to page zero. Then you can use the string address to access the characters in the string.

WORKING WITH ARRAYS

You can also work with **entire arrays** after you have accessed single variables. The key routine is **GETARYPT** at **\$F7D9**. Call it as you would call **PTRGET**, but it won't give you a variable. It gives the **location of the array's name**. The address of the name is in **LOWTR**, at **\$9B. 9C**. The **TAGSORT** routine uses **GETARYPT** twice: once for the name of the **string array**, and once for the address of the **tag array**. It assumes their type and order, so it calculates the offset to the first entry with a simple formula. To do the same thing yourself, work out the distance, in bytes, using the information on page 137 of your Applesoft manual (for the array you want to access).

When writing an ampersand routine with any **numeric** parameters that you don't have to save, pick them up as **values, not as variables**. Use **GETBYT** for any **single** byte values and **FRMNUM** for any **larger** values. Follow **FRMNUM** with **GETADR** if it's an address.

You will need a **variable** for a string or a numeric parameter that must be returned; **PTRGET** will find it for you. And good old **CHKCOM** between the parameters "gobbles" commas for you.

EXIT AMPERSAND

It's easy to exit from an ampersand routine; just **JMP DATA**. The routine at **\$D995** in Applesoft makes sure that **TXTPTR** is at the end of the statement. So, Applesoft will be at the end of a line or at a colon when **DATA** returns.

LOADING YOUR OWN ROUTINE

If your routine is small enough to fit into the space at **\$D38. DFF**, then assemble it there. **BLOAD AMP-L-SOFT** and **move the new routine up to \$4D38. 4DFF**. Use the **JUMP** listing to find the addresses for the letter code you have chosen. **Set the table in page \$48 rather than in page 8**. (Remember that Amp-L-Soft first loads beginning at **\$4750**, then relocates its routines downstairs to **\$80C** for use.) For example, the code for "Z" is used by entering the routine's address in as

*4879: 38 0D

After these two changes, just

BSAVE AMPERSANDS, A\$4750, L\$6FE

and you have a custom version of Amp-L-Soft containing your own routine.

Amp-L-Soft LOADER

For most applications, Amp-L-Soft loads and relocates itself for you without any trouble. However, should you have a large program that clobbers **LOADER** while relocating, or you want to use some package that also uses the low memory area, then you may consider modifying **LOADER**. You can fix the first problem — big programs — easier than the second one.

If you find programs **overwriting** **LOADER** during relocation, then re-assemble **LOADER** at a higher address. **The present address is only needed if you have a 32K system**. If yours is a 48K system then move it higher in memory. Leave a little headroom, though, because some programs may re-assign a string or two before loading Amp-L-Soft. All of Amp-L-Soft except **LOADER** has been assembled to execute at its relocated position

in low memory. So, only **LOADER** needs to be re-assembled. Just change the **ORG**, assemble, and copy it in over the top of the old **LOADER** at **\$4750. 47FB**. Then **BSAVE** the newly entered **LOADER**.

If you have the second type of problem — conflicting packages — then you're on your own. You could re-assemble all of the Amp-L-Soft routines to run elsewhere, and then re-write **AMPERLOAD**. Depending on where you decide to load the programs, this could be easy or difficult. Sticking everything just below the DOS buffers and then resetting **HIMEM**: to protect the programs is perhaps the easiest. But be careful not to use **MAX-FILES**. And remember, new **RUN** commands clear memory, so the **Load** statements must be in each program, not just the **HELLO** program. See the **BOOTER** listing and the figures for the details.

If you do make any changes that you feel may be of use to others, please write and share them. And let us know where you are using Amp-L-Soft. It would be great to see some neat new program that exploits its features.

continued on next page

Table 2
REFERENCE SUMMARY

Uses functions:

```
DEF FN AD(X) = PEEK(X) +
256*PEEK(X+1)
DEF FN LO(X) = INT(X/256)
DEF FN HI(X) = X - 256*INT(X/256)
```

To install: **BLOAD AMP-L-SOFT**

CALL FN

AD(FN AD(999)-79)

To remove: **FP**

TAGSORT

&A(s, t, f, l)

sort string array name

tag integer array name

value of first element

value of last element

INPUT

&I (s)

input string variable

SUBSTRING

&B(o, t, p)

object string variable

target string variable

pointer integer variable

TONE

&N(p, d)

period value

duration value

SYSCALL

&C(a, x, y, r, c)

A-reg set value

X-reg set value

Y-reg set value

routine address value

carry flag integer variable

The parameters are passed in the directions shown.



Amp-L-Soft (Cont.)

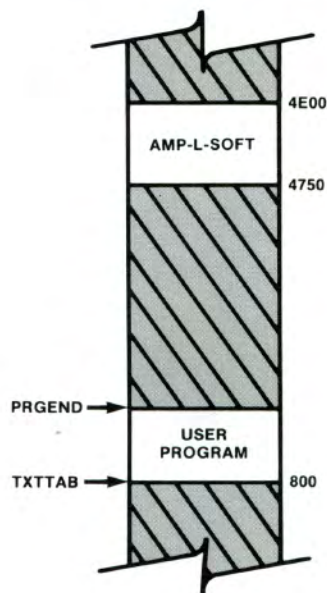


Figure 1(a). AMPL-L-SOFT loads from disk to memory below the DOS of a 32K system. From there, it must be called at \$4750.

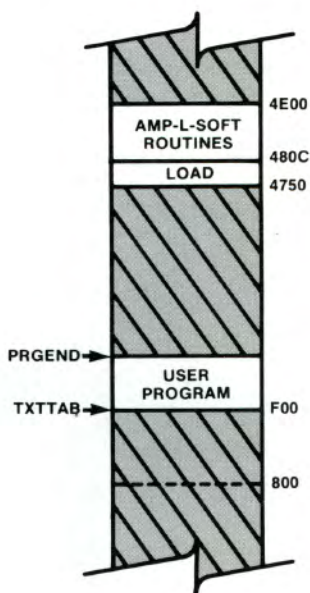


Figure 1(b). The first part called LOAD executes when run. It relocates the user program upward in memory to make room for AMP-L-SOFT ROUTINES.

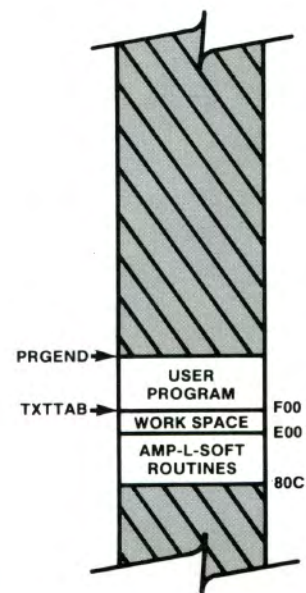


Figure 1(c). Finally, LOAD moves AMP-L-SOFT ROUTINES into low memory where it is safe from MAX-FILES, NEW, or RUN commands. Only an FP command can clobber it now. LOAD calls it to set its "&" hook in Page Three, completing the load.

Figure 2. The Memory Map of AMP-L-SOFT. After being moved to low memory, JUMP and the five routines are ready for use with the "&" hook set. The one Page work space is used by TAGSORT, but can also be used as a temporary buffer.

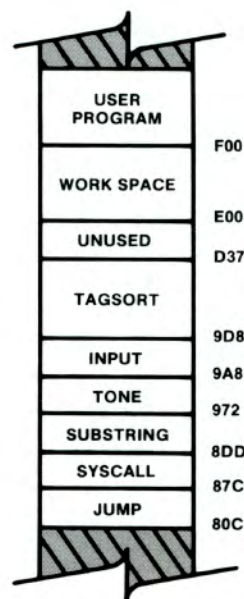


Figure 3. Any of the five routines are called using the "&" hook in Page Three. JUMP gains control and looks at the two characters following the "&". It expects a letter followed by an open bracket — "(". If one of the five codes is found, it jumps to the chosen routine, which in turn returns to Applesoft when finished by a jump to DATA. If one of the five pairs is not found, the SNERR routine in Applesoft bombs the statement.

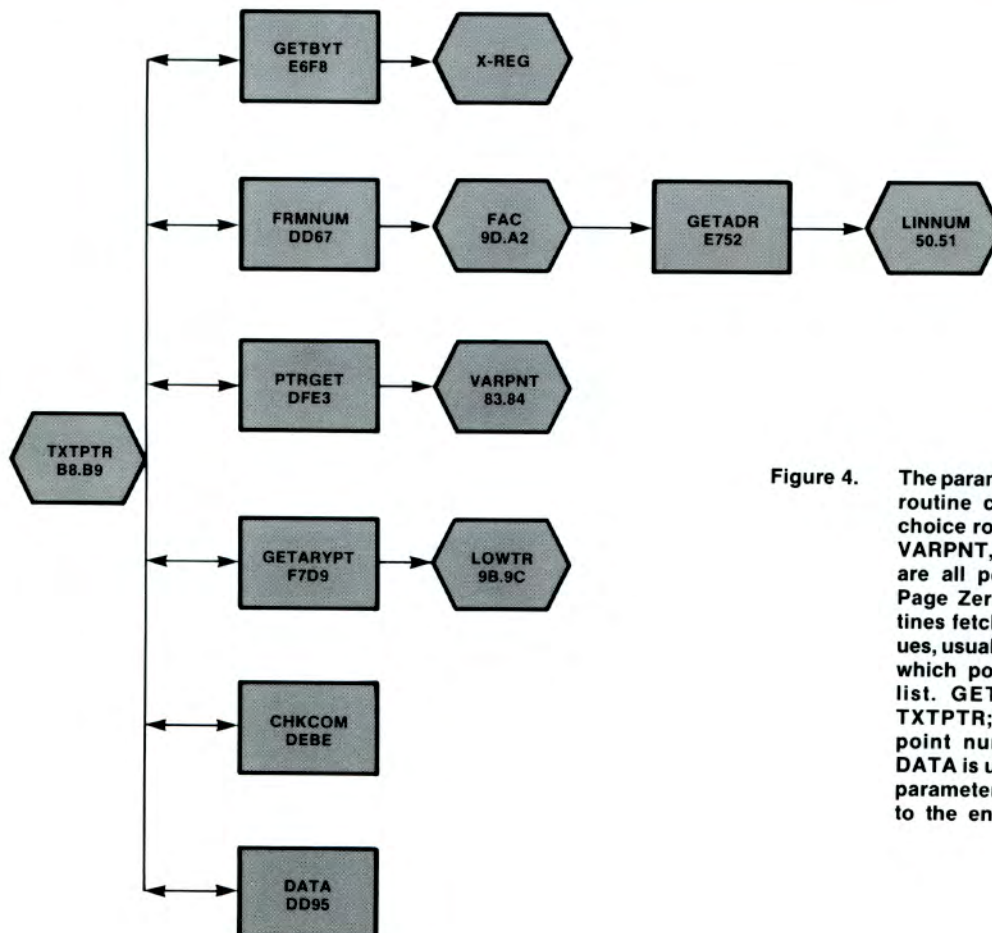
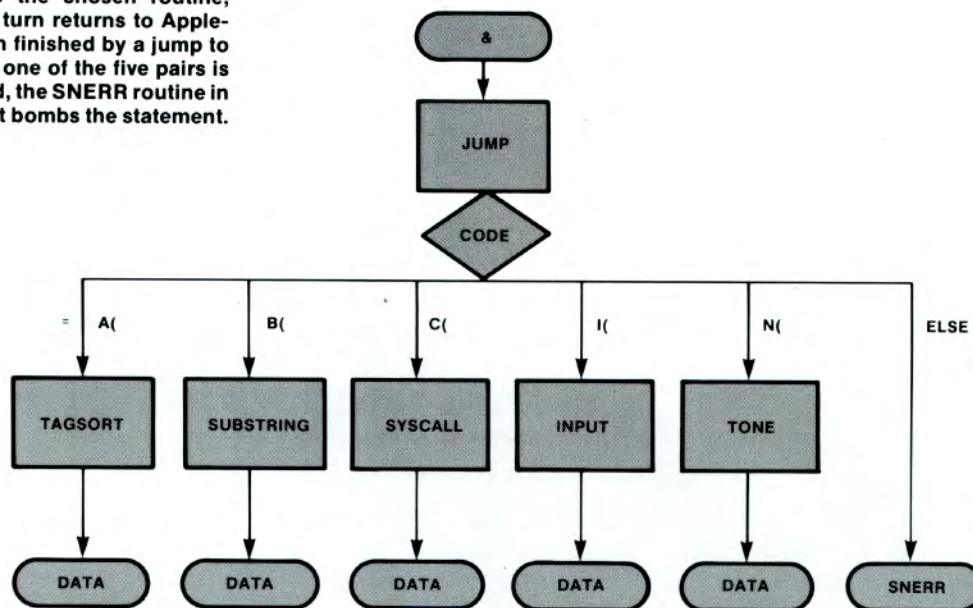


Figure 4. The parameters in an ampersand routine can be read by a few choice routines. TXTPTR, FAC, VARPNT, LOWTR, and LINNUM are all pointers and values of Page Zero. The Applesoft routines fetch the pointers and values, usually advancing TXTPTR, which points at the parameter list. GETADR does not use TXTPTR; it converts floating point numbers to addresses. DATA is used when finished the parameters to advance TXTPTR to the end of the statement.

continued on next page

Amp-L-Soft (Cont.)

```

10 REM *****
20 REM * BOOTER *
30 REM * BY PAUL IRWIN *
40 REM * COPYRIGHT (C) 1982 *
50 REM * BY MICRO-SPARC INC *
60 REM * LINCOLN, MA. 01773 *
70 REM *****
80 TEXT : HOME
90 PRINT " *** AMP-L-SOFT ***"
100 PRINT " BY PAUL IRWIN "
110 PRINT " PRINT " COPYRIGHT (C) 1982"
120 PRINT " BY MICRO-SPARC INC"
130 PRINT " LINCOLN, MA. 01773"
140 PRINT " ALL RIGHTS RESERVED"
150 PRINT
160 PRINT
170 PRINT : PRINT
180 : PRINT " LOADING AMP-L-SOFT...."
190 DEF FN AD(X) = PEEK (X) + 256 * PEEK (X + 1)
200 PRINT CHR$ (4); "LOADAMP-L-SOFT"
210 CALL FN AD( FN AD(999) - 79)
220 PRINT " ... AMP-L-SOFT INSTALLED."
230 PRINT : PRINT

```

```

10 REM *****
20 REM * AMP-L-SOFT DEMO *
30 REM * BY PAUL IRWIN *
40 REM * COPYRIGHT (C) 1982 *
50 REM * BY MICRO-SPARC INC *
60 REM * LINCOLN, MA. 01773 *
70 REM *****
80 REM DEMO TONE
90 HOME : VTAB 5: INVERSE : PRINT "TONE DEMO": NORMAL
100 INPUT "ENTER PITCH AND DURATION OF TONE ";P,D
110 & N(P,D)
120 PRINT
130 REM DEMO FREE FORMAT INPUT
140 GOSUB 10000: HOME : VTAB 5: INVERSE : PRINT "DATA INPUT
    DEMO": NORMAL
150 PRINT : PRINT "ENTER A STRING WITH ANY CHARACTERS": PRI
    NT
160 & I(IN$)
170 AS(1) = IN$
180 PRINT AS(1)
190 REM DEMO SUBSTRING SEARCH
200 PRINT
210 GOSUB 10000: HOME : VTAB 5: INVERSE : PRINT "SUBSTRING
    SEARCH DEMO": NORMAL
220 INPUT "ENTER SUBSTRING TO SEARCH ";AS
230 P$ = 1
240 BS = AS(1)
250 & (BS,AS,P$)
260 PRINT
270 PRINT "THE SUBSTRING '"AS;'" BEGINS IN LOC ";P$
280 GOSUB 10000: HOME : VTAB 5: PRINT : INVERSE : PRINT "HE
    X CONVERSION DEMO": NORMAL
290 REM DEMO HEX CONVERSION W/AMP SUBSTRING
300 INPUT "ENTER NUMBER 1-255 ";N
310 PRINT : PRINT "THE HEX EQUIVALENT IS: "; PRINT "$";
320 & C(N,0,0,64986,C$)
330 GOSUB 10000: HOME : VTAB 5: PRINT : INVERSE : PRINT "NU
    MBER OF SPACES DEMO": NORMAL
340 PRINT : INPUT "ENTER NUMBER OF SPACES (0-230) ";SP$
345 PRINT CHR$ (219);
350 & C(0,SP$,0,63818,C$)
351 PRINT CHR$ (221)
360 GOSUB 10000: HOME : VTAB 5: INPUT "ENTER THE # OF STRIN
    GS TO SORT ";N
370 DIM RS(N + 1),TS(N + 1)
380 BS = "ABCDEFGHIJKLMNPOQRSTUVWXYZ"
390 TEXT : HOME : VTAB 5: FLASH : PRINT "BUILDING STRINGS "
    : NORMAL
400 FOR F = 1 TO N: FOR G = 1 TO 5
410 RS(F) = RS(F) + MID$(BS, INT ( RND (1) * 24) + 1,1)
420 TS(F) = F
430 NEXT G,F
440 HOME : VTAB 5: PRINT "PRESS RETURN TO LIST RAW DATA": I
    NPUT "PRESS CTRL S TO PAUSE/RESUME ";XX$
450 FOR F = 1 TO N: PRINT RS(F),TS(F); NEXT F: PRINT : INPU
    T "PRESS RETURN FOR SORT ";XX$
460 PRINT "SORT BEGUN"
470 & A(RS,TS,1,N)
480 PRINT "SORT ENDED"
490 PRINT "PRESS RETURN TO LIST SORTED DATA": INPUT "PRESS
    CTRL S TO PAUSE/RESUME ";XX$
500 HOME : VTAB 5: FOR F = 1 TO N
510 IF RS(F) > RS(F + 1) THEN INVERSE
520 PRINT F; POKE 36,10: PRINT RS(F); POKE 36,20: PRINT T
    $(F); NEXT F
530 NORMAL : GOSUB 10000
540 HOME : VTAB 12: PRINT "END OF DEMO": END
10000 PRINT : PRINT : PRINT "PRESS ANY KEY TO CONTINUE =>";
    : GET AS: PRINT : RETURN

```

SOURCE FILE: AMP-L-SOFT UTILITIES

```

0000: 1 *****
0000: 2 *
0000: 3 * AMP-L-SOFT UTILITIES *
0000: 4 *
0000: 5 *****
0000: 6 *
0000: 7 * J U M P *
0000: 8 *
0000: 9 * ASSEMBLER: APPLE II V 1.0 *
0000: 10 * AUTHOR: PAUL IRWIN *
0000: 11 * WRITTEN: MAR, 1981 *
0000: 12 *
0000: 13 * INTERPRETS THE & COMMAND FROM *
0000: 14 * APPLESOFT. ALLOWS UP TO 26 *
0000: 15 * SEPERATE ROUTINES TO BE USED *
0000: 16 * FROM THE SAME PROGRAM. TO *
0000: 17 * USE, FIRST CALL 768 TO SETUP *
0000: 18 * THE & HOOK AND CLEAR THE *
0000: 19 * TABLE. THEN SET THE TABLE *
0000: 20 * ADDRESSES AS NEEDED. AFTER *
0000: 21 * THAT, AMPERSAND STATEMENTS *
0000: 22 * OF THE FORM *
0000: 23 * &X( PARAMETER LIST ) *
0000: 24 * WHERE X IS A LETTER, A TO Z, *
0000: 25 * WILL RUN THE ROUTINE WITH *
0000: 26 * TXTPTR SET TO THE FIRST OF *
0000: 27 * THE PARAMETER LIST. *
0000: 28 *
0000: 29 *****
0000: 30 *
0000: 31 *
0000: 32 * E Q U A T E S *
0000: 33 *
0000: 34 *
0000: 35 *
0000: 36 *
0000: 37 *
-----
0000: 38 ORG $000C
0000: 39 A1 EQU $3C
0000: 40 A2 EQU $3E
0000: 41 A4 EQU $42
0000: 42 LINNUM EQU $50
0000: 43 CHRGET EQU $00B1
0000: 44 CHRGET EQU $00B7
0000: 45 EXTLOC EQU $03F5 ;AMPERSAND VECTOR
0000: 46 CHKOPN EQU $DEBB ;EDIT '('
0000: 47 SNERR EQU $DEC9
0000: 48 ISLETC EQU $E07D ;EDIT 'A' TO 'Z'
0000: 49 SYSCALL EQU $0B7C ;SYSCALL ROUTINE
0000: 50 SUBSTR EQU $0BDD ;SUBSTRING SEARCH
0000: 51 TONES EQU $0972 ;TONES ROUTINE
0000: 52 INPUT EQU $09AB ;INPUT ANYTHING
0000: 53 TAGSRT EQU $09D8 ;TAGSORT RTNE
0000: 54 *
0000: 55 *
0000: 56 * INITIALIZE THE INTERPETER *
0000: 57 *
0000: 58 INITAL LDA #$4C ;JMP OPCODE
0000: 59 STA EXTLOC
0000: 60 LDA #>AMPER
0000: 61 STA EXTLOC+1
0000: 62 LDA #<AMPER
0000: 63 STA EXTLOC+2
0000: 64 RTS
0000: 65 *
0000: 66 * INTERPRET THE & ROUTINE NAME *
0000: 67 *
0000: 68 AMPER STA AMPERB ;TEMPORARILY
0000: 69 JSR ISLETC ;CHECK FOR LETTER
0000: 70 BCS AMPER1
0000: 71 JSR SNERR
0000: 72 AMPER1 JSR CHRGET
0000: 73 JSR CHKOPN ;CHECK FOR '('
0000: 74 *
0000: 75 * OKAY FOR THE SYNTAX. *
0000: 76 * LOOKUP THE ROUTINE # TO 25 *
0000: 77 * FOR 'A' TO 'Z' CODE *
0000: 78 *
0000: 79 LDA AMPERB ;REMEMBER LETTER
0000: 80 SEC
0000: 81 SBC #'A'
0000: 82 ASL A
0000: 83 TAX
0000: 84 LDA AMPERA,X
0000: 85 STA AMPER3+1
0000: 86 LDA AMPERA+1,X
0000: 87 STA AMPER3+2
0000: 88 JSR CHRGET ;LOOKING AT FIRST PARM
0000: 89 AMPER3 JMP $FFFF ;AWAY WE GO!
0000: 90 *
0000: 91 * JUMP TABLE FOR ALL '&' ROUTINES *
0000: 92 * AS CREATED BY INIT ROUTINE *
0000: 93 *
0000: 94 *
0000: 95 *
CODE ADDRESS POKES
0000: 96 AMPERA DW TAGSRT ;A 2119, LO 2120, HI
0000: 97 DW SUBSTR ;B 2121, LO 2122, HI
0000: 98 DW SYSCALL ;C 2123, LO 2124, HI
0000: 99 DW SNERR ;D 2125, LO 2126, HI
0000: 100 DW SNERR ;E 2127, LO 2128, HI
0000: 101 DW SNERR ;F 2129, LO 2130, HI
0000: 102 DW SNERR ;G 2131, LO 2132, HI
0000: 103 DW SNERR ;H 2133, LO 2134, HI
0000: 104 DW INPUT ;I 2135, LO 2136, HI
0000: 105 DW SNERR ;J 2137, LO 2138, HI
0000: 106 DW SNERR ;K 2139, LO 2140, HI
0000: 107 DW SNERR ;L 2141, LO 2142, HI
0000: 108 DW SNERR ;M 2143, LO 2144, HI
0000: 109 DW TONES ;N 2145, LO 2146, HI
0000: 110 DW SNERR ;O 2147, LO 2148, HI
0000: 111 DW SNERR ;P 2149, LO 2150, HI
0000: 112 DW SNERR ;Q 2151, LO 2152, HI
0000: 113 DW SNERR ;R 2153, LO 2154, HI
0000: 114 DW SNERR ;S 2155, LO 2156, HI
0000: 115 DW SNERR ;T 2157, LO 2158, HI
0000: 116 DW SNERR ;U 2159, LO 2160, HI
0000: 117 DW SNERR ;V 2161, LO 2162, HI

```



```

0073:C9 DE 118 DW SNERR ; W 2163, LO 2164, HI
0075:C9 DE 119 DW SNERR ; X 2165, LO 2166, HI
0077:C9 DE 120 DW SNERR ; Y 2167, LO 2168, HI
0079:C9 DE 121 DW SNERR ; Z 2169, LO 2170, HI
007B:00 122 AMPERB BRK
007C: 123 *****
007C: 124 *
007C: 125 * SYSCALL &
007C: 126 * BY PAUL IRWIN
007C: 127 *
007C: 128 * COPYRIGHT (C) 1982
007C: 129 * BY MICRO-SPARC INC
007C: 130 * LINCOLN, MA. 01773
007C: 131 * ALL RIGHTS RESERVED
007C: 132 *
007C: 133 * PASSES PARAMETERS TO ANY
007C: 134 * ROUTINE VIA 6502 REGISTER
007C: 135 *
007C: 136 * CALL SEQUENCE:
007C: 137 * &C(AR,XR,YR,R,CX)
007C: 138 *
007C: 139 * WHERE
007C: 140 * &C (PARSED BY AMPERJUMP
007C: 141 * AR = PARAM IN 6502 A-REG
007C: 142 * XR = PARAM IN 6502 X-REG
007C: 143 * YR = PARAM IN 6502 Y-REG
007C: 144 * R = ADDRESS OF ROUTINE
007C: 145 * CX = CARRY FLAG RETURNED
007C: 146 * IN INTEGER VARIABLE.
007C: 147 *
007C: 148 *****
007C: 149 *
007C: 150 *
007C: 151 *
007C: 152 *
007C: 153 * EQUATES
007C: 154 *
00B3: 155 VARPNT EQU *B3 ;ADDR OF VAR
D995: 156 DATA EQU *D995 ;MOVE TXTPTR TO END
DFE3: 157 PTRGET EQU *DFE3 ;ADDR TO VARPNT
DD67: 158 FRMNUM EQU *DD67 ;NUM FORMULA TO FAC
DEBE: 159 CHKCOM EQU *DEBE ;GOBBLE A COMMA
E6F8: 160 GETBYT EQU *E6F8 ;GET INTR TO X-REG
E752: 161 GETADR EQU *E752 ;FAC TO LINNUM
007C: 162 *
007C: 163 *
007C: 164 *
007C: 165 *
007C: 166 * ROUTINES
007C: 167 *
007C: 168 *
007C: 169 * GET THE CALLING PARAMS
007C: 170 *
007C:20 FB E6 171 JSR GETBYT ;GET A-REG
007F:BA 172 TXA
0080:48 173 PHA
0081:20 BE DE 174 JSR CHKCOM
0084:20 FB E6 175 JSR GETBYT ;GET X-REG
0087:BA 176 TXA
0088:48 177 PHA
0089:20 BE DE 178 JSR CHKCOM
008C:20 FB E6 179 JSR GETBYT ;GET Y-REG
008F:BA 180 TXA
0090:48 181 PHA
0091:20 BE DE 182 JSR CHKCOM
0094:20 67 DD 183 JSR FRMNUM ;GET RTNE ADDR.
0097:20 52 E7 184 JSR GETADR ;...INTO LINNUM
009A: 185 *
009A: 186 * SET CELL ADDRESS
009A: 187 *
009A:A5 50 188 LDA LINNUM
009C:BD AA 08 189 STA CALL+1
009F:A5 51 190 LDA LINNUM+1
00A1:BD AB 08 191 STA CALL+2
00A4: 192 *
00A4: 193 * SETUP REGISTERS FOR CALL
00A4: 194 *
00A4:68 195 PLA
00A5:A8 196 TAY
00A6:68 197 PLA
00A7:AA 198 TAX
00A8:68 199 PLA
00A9:20 FF FF 200 CALL JSR *FFFF ;CALL THE ROUTINE
00AC: 201 *
00AC: 202 * RETURN CARRY FLAG TO CALLER
00AC: 203 *
00AC:08 204 PHP
00AD:20 BE DE 205 JSR CHKCOM ;GRAB THE P-REG
00B0:20 E3 DF 206 JSR PTRGET ;FIND RETURN PARAM
00B3:A0 00 207 LDY #0
00B5:A9 00 208 LDA #0
00B7:91 83 209 STA (VARPNT),Y ;INTEGER HI=0
00B9:CB 210 INY
00BA:28 211 PLP
00BB:90 02 212 BCC #+4
00BD:A9 01 213 LDA #1 ; IF CARRY IS ON
00BF:91 83 214 STA (VARPNT),Y ;INTEGER LO=C
00C1:4C 95 D9 215 JMP DATA ;END THE STMT
00C4: 216 *
00C4: 217 *
00C4: 218 *****
00C4: 219 *
00C4: 220 * LITTLE BIT PICKERS
00C4: 221 *
00C4: 222 * THREE SYSCALL& ROUTINES TO
00C4: 223 * READ AND WRITE TO THE BITS
00C4: 224 * OF A GIVEN BYTE.
00C4: 225 *
00C4: 226 *****
00C4: 227 *
00C4: 228 * AZ = BYTE TO READ
00C4: 229 * XZ = BIT NUMBER: 7...0
00C4: 230 * &C(AZ,XZ,0,2244,PX)
00C4: 231 * VALUE OF BIT = PX
00C4: 232 *
00C4:4A 233 READ LSR A ;READS VALUE OF
00C5:CA 234 DEX ;A BIT WITHIN A BYTE
00C6:10 FC 235 BPL READ ;AS A 0 IF BIT=OFF
00CB:60 236 RTS ;OR A 1 IF BIT=ON.

```

```

00C9: 237 *
00C9: 238 * POKE 60, BYTE
00C9: 239 * XZ= BIT NUMBER: 7...0
00C9: 240 * &C(0,XZ,0,2249,PX)
00C9: 241 * BYTE WITH BIT ON = PEEK(60)
00C9: 242 *
00C9:38 243 SEC ;TURNS ON ONE BIT
00CA:2A 244 ONBIT ROL A ;AT THE BYTE
00CB:CA 245 DEX ;IN A1 (60, *3C)
00CC:10 FC 246 BPL ONBIT
00CE:05 3C 247 ORA A1
00D0:85 3C 248 STA A1
00D2:60 249 RTS
00D3: 250 *
00D3: 251 * POKE 60, BYTE
00D3: 252 * XZ= BIT NUMBER: 7...0
00D3: 253 * &C(255,XZ,0,2259,PX)
00D3: 254 * BYTE WITH BIT OFF = PEEK(60)
00D3: 255 *
00D3:18 256 CLC ;TURNS OF 1 BIT
00D4:2A 257 OFFBIT ROL A ;AT THE BYTE
00D5:CA 258 DEX ;IN A1 (60, *3C)
00D6:10 FC 259 BPL OFFBIT
00D8:25 3C 260 AND A1
00DA:85 3C 261 STA A1
00DC:60 262 RTS
00DD: 263 *
00DD: 264 * SEE CALL SEQUENCE OF SYSCALL&
00DD: 265 *
00DD: 266 *****
00DD: 267 *
00DD: 268 * SUBSTRING&
00DD: 269 * BY PAUL IRWIN
00DD: 270 *
00DD: 271 * COPYRIGHT (C) 1982
00DD: 272 * BY MICRO-SPARC INC
00DD: 273 * LINCOLN, MA. 01773
00DD: 274 * ALL RIGHTS RESERVED
00DD: 275 *
00DD: 276 *
00DD: 277 *
00DD: 278 * SEARCHES AN OBJECT STRING
00DD: 279 * FOR A MATCH TO A TARGET
00DD: 280 * STRING, BEGINNING AT A GIVEN
00DD: 281 * POSITION.
00DD: 282 *
00DD: 283 * CALL SEQUENCE
00DD: 284 * &B (OB%,TA%,PX)
00DD: 285 * WHERE
00DD: 286 * &B (IS PARSED BY AMPERJUMP
00DD: 287 * OB% = OBJECT STRING
00DD: 288 * TA% = TARGET SUBSTRING
00DD: 289 * PX = INTEGER, POSITION IN
00DD: 290 * OBJECT STRING
00DD: 291 *
00DD: 292 * NORMALLY, SET PX=1. OR ELSE
00DD: 293 * SET TO THE STARTING PLACE OF
00DD: 294 * THE DESIRED SEARCH.
00DD: 295 * THE VALUE RETURNED GIVES THE
00DD: 296 * LOCATION OF TA% IN OB% OR
00DD: 297 * ZERO IF NOT FOUND
00DD: 298 *
00DD: 299 *****
00DD: 300 *
00DD: 301 *
00DD: 302 * EQUATES
00DD: 303 *
0040: 304 A3 EQU *40 ;ADDR TARGET STRING
FCBA: 305 NTA1 EQU *FCBA ;INCR /CMP A2
00DD: 306 *
00DD: 307 *
00DD: 308 *
00DD: 309 *
00DD: 310 * ROUTINES
00DD: 311 *
00DD: 312 *
00DD: 313 * GET ADDRESSES OF PARAMETERS
00DD: 314 *
00DD:20 E3 DF 315 JSR PTRGET ;ADDR. OF OBJECT
00E0:A0 00 316 LDY #0
00E2:B1 83 317 LDA (VARPNT),Y
00E4:BD 6F 09 318 STA LOBJ
00E7:C8 319 INY
00EB:B1 83 320 LDA (VARPNT),Y
00EA:85 3C 321 STA A1
00EC:C8 322 INY
00ED:B1 83 323 LDA (VARPNT),Y
00EF:85 3D 324 STA A1+1
00F1:A5 3C 325 LDA A1
00F3:18 326 CLC
00F4:6D 6F 09 327 ADC LOBJ
00F7:85 3E 328 STA A2
00F9:A5 3D 329 LDA A1+1
00FB:69 00 330 ADC #0
00FD:85 3F 331 STA A2+1
00FF:20 BE DE 332 JSR CHKCOM
0082:20 E3 DF 333 JSR PTRGET ;ADDR OF TARGET
0085:A0 00 334 LDY #0
0087:B1 83 335 LDA (VARPNT),Y
0089:BD 70 09 336 STA LTAR
008C:C8 337 INY
008D:B1 83 338 LDA (VARPNT),Y
008F:85 40 339 STA A3
0091:C8 340 INY
0092:B1 83 341 LDA (VARPNT),Y
0094:85 41 342 STA A3+1
0096:A5 3E 343 LDA A2 ;SET A2 TO THE
0098:38 344 SEC ;LAST SUBSTRING
0099:ED 70 09 345 SBC LTAR ;IN OBJECT
009C:85 3E 346 STA A2 ;TO COMPARE
009E:A5 3F 347 LDA A2+1
00A0:E9 00 348 SBC #0
00A2:85 3F 349 STA A2+1
00A4:20 BE DE 350 JSR CHKCOM
00A7:20 E3 DF 351 JSR PTRGET ;ADDR OF POINTER
00A9:A0 01 352 LDY #1

```

continued on next page

Amp-L-Soft (Cont.)

```

092C:B1 03 353 LDA (VARPNT),Y ;INTEGER LD
092E:BD 71 09 354 STA POINT
0931:CE 71 09 355 DEC POINT
0934:AD 71 09 356 LDA POINT
0937:18 357 CLC ;SET A1 BEFORE THE
0938:65 3C 358 ADC A1 ;FIRST SUBSTRING
093A:85 3C 359 STA A1 ;IN OBJECT TO
093C:A9 00 360 LDA #0 ;TO COMPARE
093E:65 3D 361 ADC A1+1
0940:85 3D 362 STA A1+1
0942:C6 3C 363 DEC A1
0944:D0 02 364 BNE #+4
0946:C6 3D 365 DEC A1+1
0948: 366 #
094B:20 BA FC 367 NEXT JSR NXTA1
094B:B0 13 368 BCS MISS
094D:EE 71 09 369 INC POINT
0950:A0 00 370 LDY #0
0952:B1 3C 371 LOOP LDA (A1),Y
0954:D1 40 372 CMP (A3),Y
0956:D0 F0 373 BNE NEXT
0958:C8 374 INY
0959:CC 70 09 375 CPY LTAR
095C:D0 F4 376 BNE LOOP
095E:F0 05 377 BEQ HIT
0960: 378 #
0960: 379 # RETURN POINT TO CALLER
0960: 380 # BUT AS A ZERO IF MISS
0960: 381 #
0960:A9 00 382 MISS LDA #0
0962:BD 71 09 383 STA POINT
0965:A0 01 384 HIT LDY #1
0967:AD 71 09 385 LDA POINT
096A:91 83 386 STA (VARPNT),Y
096C:4C 95 D9 387 JMP DATA ;EXIT STATEMENT
096F: 388 #
096F: 389 #
096F: 390 # DATA STORAGE
096F: 391 #
096F: 392 #
096F: 393 LOBJ DS 1 ;LENGTH OF OBJECT
0970: 394 LTAR DS 1 ;LENGTH OF TARGET
0971: 395 POINT DS 1 ;POINTER PARAM
0972: 396 *****
0972: 397 #
0972: 398 # TONE#
0972: 399 # BY PAUL IRWIN
0972: 400 #
0972: 401 # COPYRIGHT (C) 1982
0972: 402 # BY MICRO-SPARC INC
0972: 403 # LINCOLN, MA. 01773
0972: 404 # ALL RIGHTS RESERVED
0972: 405 #
0972: 406 #
0972: 407 #
0972: 408 # GENERATES A TONE ON THE
0972: 409 # APPLE'S BUILT-IN SPEAKER.
0972: 410 #
0972: 411 # CALL SEQUENCE:
0972: 412 # &N (P,D)
0972: 413 # WHERE
0972: 414 # &N( IS PARSED BY AMPERJUMP
0972: 415 # P = PERIOD 0...255
0972: 416 # D = DURATION 1...255
0972: 417 #
0972: 418 *****
0972: 419 #
0972: 420 #
0972: 421 # EQUATES
0972: 422 WAIT EQU %FCAB
0972: 423 #
0972: 424 #
0972: 425 #
0972: 426 #
0972: 427 #
0972: 428 # ROUTINE
0972: 429 #
0972: 430 # GET PERIOD AND DURATION PARAMS
0972: 431 #
0972:20 F8 E6 432 JSR GETBYT
0975:BE A6 09 433 STX PERIOD
0978:20 BE DE 434 JSR CHKCOM
097B:20 F8 E6 435 JSR GETBYT ;DURATION
097E:BE A7 09 436 STX DURATN
0981: 437 #
0981: 438 # A ZERO PERIOD MEANS REST NOTE
0981: 439 #
0981:AD A6 09 440 LDA PERIOD
0984:D0 09 441 BNE TONE
0986:AD A7 09 442 LDA DURATN
0989:20 A8 FC 443 JSR WAIT ;DELAY FOR A REST NOTE
098C:4C 95 D9 444 JMP DATA ;THEN FINISHED
098F: 445 #
098F: 446 # A PERIOD > 0 MEANS A TONE NOTE
098F: 447 #
098F:AD 30 C0 448 TONE LDA #C030 ;TOGGLE SPEAKER
0992:88 449 TONE1 DEY
0993:D0 05 450 BNE TONE2
0995:CE A7 09 451 DEC DURATN
0998:F0 09 452 BEQ TONE3
099A:CA 453 TONE2 DEX
099B:D0 F5 454 BNE TONE1
099D:AE A6 09 455 LDX PERIOD
09A0:4C 8F 09 456 JMP TONE
09A3:4C 95 D9 457 TONE3 JMP DATA ;FINISHED
09A6: 458 #
09A6: 459 #
09A6: 460 # DATA STORAGE
09A6: 461 #
09A6: 462 PERIOD DS 1
09A7: 463 DURATN DS 1
09A8: 464 *****
09A8: 465 #
09A8: 466 # INPUT#
09A8: 467 # BY PAUL IRWIN

```

```

09AB: 468 #
09AB: 469 # COPYRIGHT (C) 1982
09AB: 470 # BY MICRO-SPARC INC
09AB: 471 # ALL RIGHTS RESERVED
09AB: 472 #
09AB: 473 #
09AB: 474 #
09AB: 475 #
09AB: 476 # REPLACES THE STRING INPUT
09AB: 477 # IN APPLESOFT. ALLOWS THE
09AB: 478 # USE OF QUOTES, COMMAS, AND
09AB: 479 # COLONS IN THE INPUT STRING
09AB: 480 # WITHOUT TRUNCATION. ESCAPE
09AB: 481 # AND CONTROL SEQUENCES ARE
09AB: 482 # RETAINED. USES KSM INPUT
09AB: 483 # HOOK; I.E. WORKS WITH DISK.
09AB: 484 #
09AB: 485 # CALL SEQUENCE:
09AB: 486 # &I (IN#)
09AB: 487 # WHERE
09AB: 488 # &I( IS PARSED BY AMPERJUMP
09AB: 489 # IN# = STRING VARIABLE TO
09AB: 490 # INPUT.
09AB: 491 #
09AB: 492 *****
09AB: 493 #
09AB: 494 #
09AB: 495 # EQUATES
09AB: 496 #
09AB: 497 DSCTMP EQU #9D ;NEW DESCRIPTOR
09AB: 498 BUF EQU #0200 ;INPUT BUFFER
09AB: 499 INLIN EQU #DS2C ;INPUT TO BUF
09AB: 500 STRINI EQU #E3D5 ;GET SPACE & DESCR
09AB: 501 MOVSTR EQU #E5E2 ;Y,X, TO (FRESPC)
09AB: 502 #
09AB: 503 #
09AB: 504 #
09AB: 505 #
09AB: 506 # ROUTINES
09AB: 507 #
09AB: 508 #
09AB: 509 # GET VARIABLE AND THE STRING
09AB: 510 #
09AB:20 E3 DF 511 JSR PTRGET ;GET STRING VAR
09AB:20 2C D5 512 JSR INLIN ;GET INPUT TO BUF
09AB:A2 FF 513 LDX #FFF
09AB:E8 514 MAIN1 INX ;FIND LENGTH
09AB:BD 00 02 515 LDA BUF,X ;OF STRING IN BUF
09AB:C9 00 516 CMP #0
09AB:D0 FB 517 BNE MAIN1
09AB:8A 518 TXA
09AB: 519 #
09AB: 520 # WITH LENGTH OF INPUT STRING IN
09AB: 521 # THE A-REG, PREPARE SPACE AND A
09AB: 522 # NEW DESCRIPTOR IN VARIABLE.
09AB: 523 #
09AB:20 D5 E3 524 JSR STRINI ;CREATE SPACE AND DESCR
09AB:A0 00 525 LDY #0
09AB:A5 9D 526 LDA DSCTMP
09AB:91 83 527 STA (VARPNT),Y ;COPY DESCRIPTOR
09AB:C8 528 INY
09AB:A5 9E 529 LDA DSCTMP+1
09AB:91 83 530 STA (VARPNT),Y
09AB:C8 531 INY
09AB:A5 9F 532 LDA DSCTMP+2
09AB:91 83 533 STA (VARPNT),Y
09AB: 534 #
09AB: 535 # MOVE THE STRING FROM INPUT BUFFER
09AB: 536 # TO THE NEW STRING SPACE
09AB: 537 #
09AB:A5 9D 538 LDA DSCTMP ;LENGTH
09AB:A2 00 539 LDX #0 ;ADDR LOW
09AB:A0 02 540 LDY #0 ;ADDR HIGH
09AB:20 E2 E5 541 JSR MOVSTR ;BUF TO (FRESPC)
09AB: 542 #
09AB: 543 #
09AB:4C 95 D9 544 JMP DATA ;FINISH STATEMENT
09AB: 545 #
09AB: 546 *****
09AB: 547 #
09AB: 548 # T A G S O R T &
09AB: 549 #
09AB: 550 # ASSEMBLER: APPLE II VER 1.0
09AB: 551 # AUTHOR: PAUL IRWIN
09AB: 552 # WRITTEN: MARCH, 1981
09AB: 553 #
09AB: 554 # THE QUICKSORT ALGORITHM USED
09AB: 555 # ON VECTORS OF SORT KEYS AND
09AB: 556 # TAGS. KEYS AND TAGS MUST
09AB: 557 # BE INITIALISED BEFORE THE
09AB: 558 # CALL; THEY ARE RETURNED IN
09AB: 559 # ASCENDING SEQUENCE. TO CALL
09AB: 560 # THE SORT USE THE AMPERSAND
09AB: 561 # FUNCTION FROM APPLESOFT:
09AB: 562 #
09AB: 563 # &A(S#,T#,F#,L)
09AB: 564 # WHERE
09AB: 565 # &A( IS PARSED BY AMPERSAND
09AB: 566 # JUMP UTILITY: AMPERJUMP
09AB: 567 # S# = NAME OF SORT VECTOR
09AB: 568 # SINGLE DIMENSION
09AB: 569 # T# = NAME OF TAGS VECTOR
09AB: 570 # SINGLE DIMENSION, INT
09AB: 571 # F# = 1, THE FIRST ELEMENT
09AB: 572 # L = THE LAST ELEMENT
09AB: 573 #
09AB: 574 *****
09AB: 575 #
09AB: 576 # EQUATES
09AB: 577 #
09AB: 578 # APPLESOFT USAGE
09AB: 579 #
09AB: 580 LOWTR EQU #009B
09AB: 581 CHKCLS EQU #DEB8
09AB: 582 GETARYPT EQU #F7D9
09AB: 583 #
09AB: 584 # QUICKSORT LOCAL USAGE
09AB: 585 #
09AB: 586 INDY EQU #00 ;USED BY POINT

```



```

0002: 587 DESCR EQU #02 ;USED BY POINT
0004: 588 TAGS EQU #04 ;USED BY POINT
0006: 589 IREC1 EQU #06 ;INDEX LEFTMOST
0008: 590 DREC1 EQU #02 ;POINTER TO DESCRIPTOR
0010: 591 IREC1 EQU #04 ;POINTER TO TAG
0012: 592 IREC2 EQU #06 ;INDEX RIGHTMOST
0014: 593 DREC2 EQU #08 ;POINTER TO DESC.
0016: 594 IREC2 EQU #0A ;POINTER TO TAG
0018: 595 IREC3 EQU #0C ;INDEX CURSOR
0020: 596 DREC3 EQU #0E ;POINTER TO DESC.
0022: 597 IREC3 EQU #10 ;POINTER TO TAG
0024: 598 IRECL EQU #12 ;INDEX LEFT CURSOR
0026: 599 DRECL EQU #14 ;POINTER TO DESC.
0028: 600 ITEMP EQU #16 ;POINTER TO TAG
0030: 601 ITEMP EQU #18 ;INDEX TEMP STRING
0032: 602 ITEMP EQU #1A ;POINTER TO DESC.
0034: 603 ITEMP EQU #1C ;POINTER TO TAG
0036: 604 STRNG1 EQU #1E ;COMPARE STRING 1
0038: 605 STRNG2 EQU #20 ;COMPARE STRING 2
0040: 606 ISTACK EQU #22 ;STACK INDEX

ROUTINES

0042: 607 *
0044: 608 *
0046: 609 *
0048: 610 *
0050: 611 *
0052: 612 *
0054: 613 *****
0056: 614 * MAIN LINE *
0058: 615 *****
0060: 616 * ENTER HERE FROM AMPERJUMP.
0062: 617 * PICK UP THE PARAMETERS
0064: 618 * STARTING AFTER &A( WITH
0066: 619 * THE STRING VECTOR.
0068: 620 *
0070: 621 JSR GETARYPT
0072: 622 LDA LOWTR ;ADDRESS OF SORT
0074: 623 CLC ;STRING ARRAY
0076: 624 ADC #7
0078: 625 STA SRTARY
0080: 626 LDA LOWTR+1
0082: 627 ADC #0
0084: 628 STA SRTARY+1
0086: 629 JSR CHRGT ;CHECK FOR A COMMA
0088: 630 JSR CHKCOM ;BETWEEN PARMS
0090: 631 JSR GETARYPT ;ADDRESS OF TAGS
0092: 632 LDA LOWTR ;INTEGER ARRAY
0094: 633 CLC
0096: 634 ADC #7
0098: 635 STA TAGARY
0100: 636 LDA LOWTR+1
0102: 637 ADC #0
0104: 638 STA TAGARY+1
0106: 639 JSR CHRGT ;CHECK FOR A COMMA
0108: 640 JSR CHKCOM ;BETWEEN PARMS
0110: 641 JSR FRMMUM
0112: 642 JSR GETADR ;FIRST ELEMENT NUMBER
0114: 643 JSR CHKCOM ;WITH COMMA
0116: 644 LDA LINNUM
0118: 645 STA INDEX1
0120: 646 LDA LINNUM+1
0122: 647 STA INDEX1+1
0124: 648 JSR CHRGT
0126: 649 JSR FRMMUM
0128: 650 JSR GETADR ;LAST ELEMENT NUMBER
0130: 651 JSR CHKCLS ;WITH ')' CLOSE
0132: 652 LDA LINNUM
0134: 653 STA INDEX2
0136: 654 LDA LINNUM+1
0138: 655 STA INDEX2+1
0140: 656 JSR DATA ;OK, ADVANCE TXTPTR
0142: 657 *
0144: 658 * PARAMETERS READ OK. NOW
0146: 659 * THE SORT IS CALLED AND
0148: 660 * ITS PAGE ZERO NEEDS MANAGED.
0150: 661 *
0152: 662 JSR SAVEZ ;SAVE PAGE ZERO
0154: 663 JSR TSORT ;DO THE SORT
0156: 664 JSR RESTZ ;RESTORE PAGE ZERO
0158: 665 RTS ;... TO APPLESOFT PROGRAM
0160: 666 *
0162: 667 *****
0164: 668 * SAVE SOME OF PAGE ZERO *
0166: 669 *****
0168: 670 SAVEZ LDX #03F
0170: 671 SAVEZ1 LDA IREC1,X
0172: 672 STA SAVEA,X
0174: 673 DEX
0176: 674 BPL SAVEZ1
0178: 675 RTS
0180: 676 *
0182: 677 *****
0184: 678 * RESTORE SOME OF PAGE ZERO *
0186: 679 *****
0188: 680 RESTZ LDX #03F
0190: 681 RESTZ1 LDA SAVEA,X
0192: 682 STA IREC1,X
0194: 683 DEX
0196: 684 BPL RESTZ1
0198: 685 RTS
0200: 686 *
0202: 687 *****
0204: 688 * TAG SORT *
0206: 689 *****
0208: 690 *
0210: 691 * THE TAG SORT IS CALLED ONLY
0212: 692 * ONCE HERE; FROM THE MAIN LINE.
0214: 693 *
0216: 694 TSORT LDA INDEX1 ; SET INDEXES
0218: 695 STA IREC1 ; IREC1 AND REC2
0220: 696 LDA INDEX1+1 ; TO INDEX1
0222: 697 STA IREC1+1 ; AND INDEX2
0224: 698 LDA INDEX2 ; RESPECTIVELY.
0226: 699 STA IREC2
0228: 700 LDA INDEX2+1
0230: 701 STA IREC2+1
0232: 702 LDA #>STACKA ; RESET STACK INDEX
0234: 703 STA ISTACK ; TO TOP (STACKA)
0236: 704 LDA #<STACKA
0238: 705 STA ISTACK+1

```

```

0A70: 706 *
0A72: 707 * THE TAG SORT IS CALLED
0A74: 708 * RECURSIVELY HERE; FROM WITHIN.
0A76: 709 *
0A78: 710 SORT LDA IREC1 ;DO WHILE
0A80: 711 SEC ;IREC1 < IREC2
0A82: 712 SBC IREC2
0A84: 713 LDA IREC1+1
0A86: 714 SBC IREC2+1
0A88: 715 BCS SORTX
0A90: 716 JSR PART ;FINDS IREC3
0A92: 717 * FIRST, SORT LEFT PARTITION
0A94: 718 JSR STACK ;SAVE REC1,REC2,REC3
0A96: 719 LDA IREC3 ;SET REC2 <-- REC3 - 1
0A98: 720 SEC
0A9A: 721 SBC #1
0A9C: 722 STA IREC2
0A9E: 723 LDA IREC3+1
0AA0: 724 SBC #0
0AA2: 725 STA IREC2+1
0AA4: 726 JSR SORT ;REC1 TO (REC3-1)
0AA6: 727 JSR UNSTACK
0AA8: 728 * NEXT, SORT RIGHT PARTITION
0AA9: 729 JSR STACK
0AA9: 730 LDA IREC3 ;SET REC1 <-- REC3 + 1
0AA9: 731 CLC
0AA9: 732 ADC #1
0AA9: 733 STA IREC1
0AA9: 734 LDA IREC3+1
0AA9: 735 ADC #0
0AA9: 736 STA IREC1+1
0AA9: 737 JSR SORT ;(REC3+1) TO REC2
0AA9: 738 JSR UNSTACK
0AA9: 739 SORTX RTS ;END DO WHILE
0AAB: 740 *
0AAB: 741 *****
0AAB: 742 * STACK RECORD INDICES *
0AAB: 743 *****
0AAB: 744 STACK LDX #0
0AAD: 745 LDA IREC1
0AAF: 746 STA (ISTACK,X)
0AB1: 747 INC ISTACK
0AB3: 748 LDA IREC1+1
0AB5: 749 STA (ISTACK,X)
0AB7: 750 INC ISTACK
0AB9: 751 LDA IREC2
0ABB: 752 STA (ISTACK,X)
0ABD: 753 INC ISTACK
0ABF: 754 LDA IREC2+1
0AC1: 755 STA (ISTACK,X)
0AC3: 756 INC ISTACK
0AC5: 757 LDA IREC3
0AC7: 758 STA (ISTACK,X)
0AC9: 759 INC ISTACK
0ACB: 760 LDA IREC3+1
0ACD: 761 STA (ISTACK,X)
0ACE: 762 INC ISTACK
0AD1: 763 RTS
0AD2: 764 *
0AD2: 765 *****
0AD2: 766 * UNSTACK RECORD INDICES *
0AD2: 767 *****
0AD2: 768 UNSTACK LDX #0
0AD4: 769 DEC ISTACK
0AD6: 770 LDA (ISTACK,X)
0AD8: 771 STA IREC3+1
0ADA: 772 DEC ISTACK
0ADC: 773 LDA (ISTACK,X)
0ADE: 774 STA IREC3
0AE0: 775 DEC ISTACK
0AE2: 776 LDA (ISTACK,X)
0AE4: 777 STA IREC2+1
0AE6: 778 DEC ISTACK
0AE8: 779 LDA (ISTACK,X)
0AEA: 780 STA IREC2
0AEC: 781 DEC ISTACK
0AEE: 782 LDA (ISTACK,X)
0AF0: 783 STA IREC1+1
0AF2: 784 DEC ISTACK
0AF4: 785 LDA (ISTACK,X)
0AF6: 786 STA IREC1
0AF8: 787 RTS
0AF9: 788 *****
0AF9: 789 * PARTITION R1,R2 AT R3 *
0AF9: 790 *****
0AF9: 791 *
0AF9: 792 * SETTING UP THE PARAMETERS
0AF9: 793 *
0AF9: 794 PART LDA IREC1 ;SET REC3 TO THE
0AFB: 795 CLC ;MEDIAN OF REC1 AND REC2
0AFC: 796 ADC IREC2 ;I.E. ADD IREC1 AND
0AFE: 797 STA IREC3 ;IREC2 THEN DIVIDE
0B00: 798 LDA IREC1+1 ;THE SUM BY TWO.
0B02: 799 ADC IREC2+1
0B04: 800 STA IREC3+1
0B06: 801 ROR IREC3+1
0B08: 802 ROR IREC3
0B0A: 803 LDX #IREC1
0B0C: 804 JSR POINT2
0B0E: 805 LDX #IREC3
0B10: 806 JSR POINT2
0B12: 807 JSR SWAPA ;SWAP REC1 WITH REC3
0B14: 808 LDA IREC1 ;SET RECL <-- REC1 + 1
0B16: 809 CLC
0B18: 810 ADC #1
0B1A: 811 STA IRECL
0B1C: 812 LDA IREC1+1
0B1E: 813 ADC #0
0B20: 814 STA IREC1+1
0B22: 815 LDA IREC2
0B24: 816 STA IREC3
0B26: 817 LDA IREC2+1
0B28: 818 STA IREC3+1
0B2A: 819 *
0B2C: 820 * PARTITION DECISION
0B2C: 821 *

```

continued on next page

Amp-L-Soft (Cont.)

```

0B2C:A2 60 822 DECIDE LDJ #IREC1 ;SET THE POINTERS
0B2E:20 57 0C 823 JSR POINT2 ;FOR REC1, RECL, AND
0B31:A2 72 824 LDJ #IRECL ;REC3.
0B33:20 57 0C 825 JSR POINT2
0B36:A2 6C 826 LDJ #IREC3
0B38:20 57 0C 827 JSR POINT2
0B3B:A9 00 828 LDA #0
0B3D:8D 09 0D 829 STA CASE
0B40:A5 72 830 LDA IRECL ;IF IRECL >= IREC3
0B42:18 831 CLC ;THEN CASE = CASE + 1
0B43:E5 6C 832 SBC IREC3
0B45:A5 73 833 LDA IRECL+1
0B47:E5 6D 834 SBC IREC3+1
0B49:90 03 835 BCC DECIDE1
0B4B:EE 09 0D 836 INC CASE
0B4E:A0 00 837 DECIDE1 LDY #0 ;LENGTH <--
0B50:B1 62 838 LDA (DREC1),Y ;MINIMUM LENGTH
0B52:D1 6E 839 CMP (DREC3),Y ;OF (REC1,REC3)
0B54:90 02 840 BCC DECIDE2
0B56:B1 6E 841 LDA (DREC3),Y
0B58:8D 0B 0D 842 DECIDE2 STA LENGTH
0B5B:C8 843 INY
0B5C:B1 62 844 LDA (DREC1),Y ;SET STRNG1 AS REC1
0B5E:85 7E 845 STA STRNG1
0B60:B1 6E 846 LDA (DREC3),Y
0B62:85 80 847 STA STRNG2
0B64:C8 848 INY
0B65:B1 62 849 LDA (DREC1),Y ;SET STRNG2 AS REC3
0B67:85 7F 850 STA STRNG1+1
0B69:B1 6E 851 LDA (DREC3),Y
0B6B:85 81 852 STA STRNG2+1
0B6D:A0 00 853 LDY #0
0B6F:B1 7E 854 DECIDE3 LDA (STRNG1),Y ;IF REC1 >= REC3
0B71:D1 80 855 CMP (STRNG2),Y ;THEN CASE <-- CASE + 2
0B73:90 16 856 BCC DECIDE5
0B75:D0 0E 857 BNE DECIDE4
0B77:C8 858 INY
0B78:CC 0B 0D 859 CPY LENGTH
0B7B:D0 F2 860 BNE DECIDE3
0B7D:A0 00 861 LDY #0
0B7F:B1 62 862 LDA (DREC1),Y
0B81:D1 6E 863 CMP (DREC3),Y
0B83:90 06 864 BCC DECIDE5
0B85:EE 09 0D 865 DECIDE4 INC CASE
0B88:EE 09 0D 866 INC CASE
0B8B:A0 00 867 DECIDE5 LDY #0 ;LENGTH <--
0B8D:B1 62 868 LDA (DREC1),Y ;MINIMUM LENGTH
0B8F:D1 74 869 CMP (DREC1),Y ;OF (REC1,RECL)
0B91:90 02 870 BCC DECIDE6
0B93:B1 74 871 LDA (DREC1),Y
0B95:8D 0B 0D 872 DECIDE6 STA LENGTH
0B98:C8 873 INY
0B99:B1 62 874 LDA (DREC1),Y ;SET STRNG1 AS REC1
0B9B:85 7E 875 STA STRNG1
0B9D:B1 74 876 LDA (DREC1),Y
0B9F:85 80 877 STA STRNG2
0BA1:C8 878 INY
0BA2:B1 62 879 LDA (DREC1),Y ;SET STRNG2 AS RECL
0BA4:85 7F 880 STA STRNG1+1
0BA6:B1 74 881 LDA (DREC1),Y
0BA8:85 81 882 STA STRNG2+1
0BAA:A0 00 883 LDY #0
0BAC:B1 80 884 DECIDE7 LDA (STRNG2),Y ;IF RECL >= REC1
0BAE:D1 7E 885 CMP (STRNG1),Y ;THEN CASE <-- CASE + 4
0BB0:90 1C 886 BCC DECIDE9
0BB2:D0 0E 887 BNE DECIDE8
0BB4:C8 888 INY
0BB5:CC 0B 0D 889 CPY LENGTH
0BB8:D0 F2 890 BNE DECIDE7
0BBA:A0 00 891 LDY #0
0BBC:B1 74 892 LDA (DREC1),Y
0BBE:D1 62 893 CMP (DREC1),Y
0BC0:90 0C 894 BCC DECIDE9
0BC2:EE 09 0D 895 DECIDE8 INC CASE
0BC5:EE 09 0D 896 INC CASE
0BC8:EE 09 0D 897 INC CASE
0BCB:EE 09 0D 898 INC CASE
0BCE:AD 09 0D 899 DECIDE9 LDA CASE
0BD1:C9 07 900 CMP #7
0BD3:D0 07 901 BNE DECIDE10
0BD5:A9 03 902 LDA #3
0BD7:8D 09 0D 903 STA CASE
0BDA:D0 09 904 BNE DECIDE11
0BDC:C9 05 905 DECIDE10 CMP #5
0BDE:D0 05 906 BNE DECIDE11
0BE0:A9 01 907 LDA #1
0BE2:8D 09 0D 908 STA CASE
0BE5:EA 909 DECIDE11 NOP
0BE6: 910 #
0BE6: 911 # PARTITION ACTION
0BE6: 912 #
0BE6:C9 00 913 ACTION0 CMP #0
0BE8:D0 10 914 BNE ACTION1
0BEA:A5 6C 915 LDA IREC3
0BEC:38 916 SEC
0BED:E9 01 917 SBC #1 ;IREC3 <-- IREC3 - 1
0BEF:85 6C 918 STA IREC3
0BF1:A5 6D 919 LDA IREC3+1
0BF3:E9 00 920 SBC #0
0BF5:85 6D 921 STA IREC3+1
0BF7:4C 2C 0B 922 JMP DECIDE
0BFA:C9 01 923 ACTION1 CMP #1
0BFC:D0 10 924 BNE ACTION2
0BFE:A5 6C 925 LDA IREC3
0C00:38 926 SEC
0C01:E9 01 927 SBC #1 ;IREC3 <-- IREC3 - 1
0C03:85 6C 928 STA IREC3
0C05:A5 6D 929 LDA IREC3+1
0C07:E9 00 930 SBC #0
0C09:85 6D 931 STA IREC3+1
0C0B:4C 2C 0B 932 JMP DECIDE
0C0E:C9 02 933 ACTION2 CMP #2
0C10:D0 10 934 BNE ACTION3
0C12:A5 72 935 LDA IRECL
0C14:18 936 CLC
0C15:69 01 937 ADC #1 ;IRECL <-- IRECL + 1

```

```

0C17:85 72 938 STA IRECL
0C19:A5 73 939 LDA IRECL+1
0C1B:69 00 940 ADC #0
0C1D:85 73 941 STA IRECL+1
0C1F:4C 2C 0B 942 JMP DECIDE
0C22:C9 03 943 ACTION3 CMP #3
0C24:D0 04 944 BNE ACTION4
0C26:20 8F 0C 945 JSR SWAPB ;REC3 WITH REC1
0C29:60 946 RTS ;EXIT OF PARTITION
0C2A:C9 04 947 ACTION4 CMP #4
0C2C:D0 10 948 BNE ACTION5
0C2E:A5 6C 949 LDA IREC3
0C30:38 950 SEC
0C31:E9 01 951 SBC #1
0C33:85 6C 952 STA IREC3
0C35:A5 6D 953 LDA IREC3+1
0C37:E9 00 954 SBC #0
0C39:85 6D 955 STA IREC3+1
0C3B:4C 2C 0B 956 JMP DECIDE
0C3E:EA 957 ACTION5 NOP ;NOT DEFINED
0C3F:C9 06 958 ACTION6 CMP #6
0C41:D0 13 959 BNE ACTION7
0C43:20 C6 0C 960 JSR SWAPB ;RECL WITH REC3
0C46:A5 6C 961 LDA IREC3
0C48:38 962 SEC
0C49:E9 01 963 SBC #1 ;IREC3 <-- IREC3 - 1
0C4B:85 6C 964 STA IREC3
0C4D:A5 6D 965 LDA IREC3+1
0C4F:E9 00 966 SBC #0
0C51:85 6D 967 STA IREC3+1
0C53:4C 2C 0B 968 JMP DECIDE
0C56:00 969 ACTION7 BRK ;THAT'S ALL OF THEM!
0C57: 970 #
0C57: 971 *****
0C57: 972 # SET POINTERS FROM INDICES #
0C57: 973 *****
0C57:B5 00 974 POINT2 LDA INDX,X ;CALCULATES
0C59:18 975 CLC ;DESCRIPTOR <--
0C5A:2A 976 ROLA ;3*INDX + SRTARY
0C5B:95 02 977 STA DESCR,X ;AND
0C5D:95 04 978 STA TAGS,X ;TAGS <--
0C5F:B5 01 979 LDA INDX+1,X ;2*INDX + TAGARY
0C61:2A 980 ROLA
0C62:95 03 981 STA DESCR+1,X
0C64:95 05 982 STA TAGS+1,X
0C66:85 00 983 LDA INDX,X
0C68:75 02 984 ADC DESCR,X
0C6A:95 02 985 STA DESCR,X
0C6C:85 01 986 LDA INDX+1,X
0C6E:75 03 987 ADC DESCR+1,X
0C70:95 03 988 STA DESCR+1,X
0C72:AD FD 0C 989 LDA SRTARY
0C75:75 02 990 ADC DESCR,X
0C77:95 02 991 STA DESCR,X
0C79:AD FE 0C 992 LDA SRTARY+1
0C7C:75 03 993 ADC DESCR+1,X
0C7E:95 03 994 STA DESCR+1,X
0C80:AD 00 0D 995 LDA TAGARY
0C83:75 04 996 ADC TAGS,X
0C85:95 04 997 STA TAGS,X
0C87:AD 01 0D 998 LDA TAGARY+1
0C8A:75 05 999 ADC TAGS+1,X
0C8C:95 05 1000 STA TAGS+1,X
0C8E:60 1001 RTS
0C8F: 1002 #
0C8F: 1003 *****
0C8F: 1004 # SWAP REC1 AND REC3 #
0C8F: 1005 *****
0C8F:A0 00 1006 SWAPB LDY #0
0C91:B1 62 1007 LDA (DREC1),Y
0C93:AA 1008 TAX
0C94:B1 6E 1009 LDA (DREC3),Y ; SWAP FIRST BYTE
0C96:91 62 1010 STA (DREC1),Y ; OF DESCRIPTORS:
0C98:8A 1011 TXA ;THE STRING LENGTH
0C99:91 6E 1012 STA (DREC3),Y
0C9B:B1 64 1013 LDA (TREC1),Y
0C9D:AA 1014 TAX
0C9E:B1 70 1015 LDA (TREC3),Y ; SWAP FIRST BYTE
0CA0:91 64 1016 STA (TREC1),Y ; OF TAG DESCR:
0CA2:8A 1017 TXA ; THE INTEGER (HI BYTE)
0CA3:91 70 1018 STA (TREC3),Y
0CA5:C8 1019 INY
0CA6:B1 62 1020 LDA (DREC1),Y
0CA8:AA 1021 TAX
0CA9:B1 6E 1022 LDA (DREC3),Y ; SWAP SECOND BYTE
0CAB:91 62 1023 STA (DREC1),Y ; OF DESCRIPTORS:
0CAD:8A 1024 TXA ; STRING ADDRESS (HI BYTE)
0CAE:91 6E 1025 STA (DREC3),Y
0CB0:B1 64 1026 LDA (TREC1),Y
0CB2:AA 1027 TAX
0CB3:B1 70 1028 LDA (TREC3),Y ; SWAP SECOND BYTE
0CB5:91 64 1029 STA (TREC1),Y ; OF TAG DESCR:
0CB7:8A 1030 TXA ; THE INTEGER (LO BYTE)
0CB8:91 70 1031 STA (TREC3),Y
0CBA:C8 1032 INY
0CBB:B1 62 1033 LDA (DREC1),Y
0CBD:AA 1034 TAX
0CBE:B1 6E 1035 LDA (DREC3),Y ; SWAP THIRD BYTE
0CC0:91 62 1036 STA (DREC1),Y ; OF DESCRIPTORS:
0CC2:8A 1037 TXA ; THE STRING LENGTH
0CC3:91 6E 1038 STA (DREC3),Y
0CC5:60 1039 RTS
0CC6: 1040 #
0CC6: 1041 *****
0CC6: 1042 # SWAP RECL AND REC3 #
0CC6: 1043 *****
0CC6:A0 00 1044 SWAPB LDY #0
0CC8:B1 74 1045 LDA (DREC1),Y
0CCA:AA 1046 TAX
0CCB:B1 6E 1047 LDA (DREC3),Y ; SWAP FIRST BYTE
0CCD:91 74 1048 STA (DREC1),Y ; OF DESCRIPTORS:
0CCF:8A 1049 TXA ; THE STRING LENGTH
0CD0:F9 16 1050 STA (DREC3),Y
0CD2:B1 7E 1051 LDA (TREC1),Y
0CD4:AA 1052 TAX
0CD5:B1 70 1053 LDA (TREC3),Y ; SWAP FIRST BYTE
0CD7:91 7E 1054 STA (TREC1),Y ; OF TAG DESCR:
0CD9:8A 1055 TXA ; THE INTEGER (HI BYTE)

```



```

00CDA:91 70 1056 STA (TREC3),Y
00CDC:CB 1057 INY
00CDD:B1 74 1058 LDA (DRECL),Y
00CDF:AA 1059 TAX
00CE0:B1 6E 1060 LDA (DREC3),Y ; SWAP SECOND BYTE
00CE2:91 74 1061 STA (DRECL),Y ; OF DESCRIPTORS:
00CE4:8A 1062 TXA ; STRING ADDRESS (HI BYTE)
00CE5:91 6E 1063 STA (DREC3),Y
00CE7:B1 76 1064 LDA (TREC3),Y
00CE9:AA 1065 TAX
00CEA:B1 70 1066 LDA (TREC3),Y ; SWAP SECOND BYTE
00CEC:91 76 1067 STA (TREC3),Y ; OF TAG DESCR:
00CEE:8A 1068 TXA ; THE INTEGER (LO BYTE)
00CF1:91 70 1069 STA (TREC3),Y
00CF1:CB 1070 INY
00CF2:B1 74 1071 LDA (DRECL),Y
00CF4:AA 1072 TAX
00CF5:B1 6E 1073 LDA (DREC3),Y ; SWAP THIRD BYTE
00CF7:91 74 1074 STA (DRECL),Y ; OF DESCRIPTORS:
00CF9:8A 1075 TXA ; STRING ADDRESS (LO BYTE)
00CFA:91 6E 1076 STA (DREC3),Y
00CFC:60 1077 RTS
00CFD: 1078 *
00CFD: 1079 *
00CFD: 1080 * DATA STORAGE
00CFD: 1081 *
00CFD: 1082 *
00CFD: 1083 *
00CFD: 1084 * WORKING STORAGE FOR FARMS
00CFD: 1085 *
00CFD: 1086 SRTARY DS 3 ; SORT ARRAY ADDRESS
00D00: 1087 TAGARY DS 3 ; TAGS ARRAY ADDRESS
00D03: 1088 INDEX1 DS 3 ; FIRST ELEMENT OF SORT
00D06: 1089 INDEX2 DS 3 ; LAST ELEMENT OF SORT
00D09: 1090 CASE DS 2
00D0B: 1091 LENGTH DS 2
00D0D: 1092 *
00D0D: 1093 SAVEA DS $40 ; PAGE ZERO SAVE AREA
00D4D: 1094 STACKA DS $100 ; FARMS STACK AREA
00E4D:00 1095 BRK

```

*** SUCCESSFUL ASSEMBLY: NO ERRORS

SOURCE FILE: LOADER

```

0000: 1 *****
0000: 2 * LOADER *
0000: 3 * BY PAUL IRWIN *
0000: 4 *
0000: 5 * COPYRIGHT (C) 1982 *
0000: 6 * BY MICRO-SPARC INC *
0000: 7 * LINCOLN, MA. 01773 *
0000: 8 * ALL RIGHTS RESERVED *
0000: 9 * APPLE DOS TOOL KIT ASSEMBLER *
0000: 10 *****
0000: 11 *
0000: 12 * ROUTINE TO LOAD AND CONNECT *
0000: 13 * THE AMPERSAND ROUTINES WITH *
0000: 14 * AN EXISTING PROGRAM. *
0000: 15 *
0000: 16 * FROM APPLESOFT, CALL BEFORE *
0000: 17 * ANY VARIABLES OR ARRAYS ARE *
0000: 18 * REFERENCED. USE COMMANDS: *
0000: 19 *
0000: 20 * BLOAD AMPER MASTER *
0000: 21 * CALL 18256 *
0000: 22 *
0000: 23 * FINAL POSITION OF ROUTINES *
0000: 24 * IS IN $B0C.EFF AREA, WITH *
0000: 25 * USER PROGRAM RELOCATED TO *
0000: 26 * START AT $F01. *
0000: 27 *
0000: 28 * THE LOADER RELOCATES THE *
0000: 29 * CODE FROM $480C DOWN TO A *
0000: 30 * STARTING ADDRESS OF $B0C. *
0000: 31 *
0000: 32 *****
0000: 33 * EQUATES *
003C: 34 A1 EQU $3C ; MOVE BEGIN
003E: 35 A2 EQU $3E ; MOVE END
0042: 36 A4 EQU $42 ; MOVE DESTIN
0050: 37 LINNUM EQU $50 ; INTEGER
005E: 38 INDEX EQU $5E ; INTEGER
0067: 39 TXTTAB EQU $67 ; START PROG
0069: 40 VARTAB EQU $69 ; START VARS
0079: 41 NXTPTR EQU $79 ; NEXT STMT
0094: 42 HIGHDS EQU $94 ; USED BY BLTU
0096: 43 HIGHTR EQU $96 ; USED BY BLTU
00AF: 44 PRGEND EQU $AF ; PROG END
00B8: 45 TXTPTR EQU $B8 ; TEXT PTR
00BC: 46 INIT EQU $B0C ; & RESIDENCE
00F0: 47 NUPROG EQU $F00 ; DESTIN PROG
00FC: 48 ILOAD EQU $4B0C ; RELOC START
00FF: 49 ELOAD EQU $4DFF ; RELOC END
03C3: 50 BLT2 EQU $D3C3 ; PART OF BLTU
D66C: 51 CLEARC EQU $D66C ; CLEAR VAR/STACK
FE2C: 52 MOVE EQU $FE2C ; MONITOR MOVE

```

----- NEXT OBJECT FILE NAME IS LOADER.OBJ0

```

4750: 53 ORG $4750
4750: 54 *
4750: 55 *
4750: 56 * ROUTINES
4750: 57 *
4750: 58 *
4750: 59 *
4750: 60 * MOVE PROGRAM FROM PRESENT LOCATION
4750: 61 * TO ADDRESS IN NUPROG.
4750: 62 *
4750:A9 0F 63 LDA #<NUPROG ; KONZEN'S SEQ
4752:85 51 64 STA LINNUM+1 ; USED TO MOVE PROG
4754:A9 00 65 LDA #>NUPROG ; TEXT UPWARD IN MEM
4756:85 50 66 STA LINNUM ; BY CALLING BLT2
4758:A5 67 67 LDA TXTTAB

```

```

475A:85 96 68 STA HIGHTR
475C:A5 80 69 LDA PRGEND+1 ; REF APPLE ORCHARD
475E:85 97 70 STA HIGHTR+1 ; ISSUE #1
4760:C6 96 71 DEC HIGHTR
4762:30 72 SEC
4763:A5 50 73 LDA LINNUM
4765:85 94 74 STA HIGHDS
4767:E5 67 75 SBC TXTTAB
4769:85 50 76 STA LINNUM
476B:A5 51 77 LDA LINNUM+1
476D:85 95 78 STA HIGHDS+1
476F:E5 68 79 SBC TXTTAB+1
4771:85 51 80 STA LINNUM+1
4773:A5 AF 81 LDA PRGEND
4775:E5 67 82 SBC TXTTAB
4777:A0 83 TAY
4778:A5 B0 84 LDA PRGEND+1
477A:E5 68 85 SBC TXTTAB+1
477C:AA 86 TAX
477D:10 87 CLC
477E:65 95 88 ADC HIGHDS+1
4780:85 95 89 STA HIGHDS+1
4782:C0 90 INY
4783:D0 01 91 BNE #+3
4785:E0 92 INX
4786:E0 93 INX
4787:C0 94 INY
4788:20 C3 D3 95 JSR BLT2 ; DOES THE MOVE
478B: 96 *
478B: 97 * ADJUST APPLSOFT POINTERS
478B: 98 *
478B:A2 69 99 LDX #VARTAB
478D:20 EE 47 100 JSR ADD
4790:A2 AF 101 LDX #PRGEND
4792:20 EE 47 102 JSR ADD
4795:A2 67 103 LDX #TXTTAB
4797:20 EE 47 104 JSR ADD
479A:A2 B8 105 LDX #TXTPTR
479C:A5 B9 106 LDA TXTPTR+1
479E:C9 02 107 CMP #2
47A0:F0 03 108 BEQ #+5
47A2:20 EE 47 109 JSR ADD
47A5:A6 79 110 LDX NXTPTR
47A7:20 EE 47 111 JSR ADD
47AA: 112 *
47AA: 113 * ADJUST NEXT LINE ADDRESSES
47AA: 114 * IN PROGRAM TEXT
47AA: 115 *
47AA:A5 67 116 LDA TXTTAB
47AC:85 5E 117 STA INDEX
47AE:A5 68 118 LDA TXTTAB+1
47B0:85 5F 119 STA INDEX+1
47B2:A0 00 120 LOAD1 LDY #0
47B4:30 121 SEC
47B5:B1 5E 122 LDA (INDEX),Y
47B7:65 50 123 ADC LINNUM
47B9:91 5E 124 STA (INDEX),Y
47BB:AA 125 TAX
47BC:C0 126 INY
47BD:B1 5E 127 LDA (INDEX),Y
47BF:F0 0A 128 BEQ LOAD2
47C1:65 51 129 ADC LINNUM+1
47C3:91 5E 130 STA (INDEX),Y
47C5:86 5E 131 STX INDEX ; POINT TO NEXT
47C7:85 5F 132 STA INDEX+1
47C9:90 E7 133 BCC LOAD1 ; LINE ALWAYS
47CB: 134 *
47CB: 135 *
47CB: 136 * PROGRAM TEXT OK - NOW RELOCATE
47CB: 137 * THE AMPERSAND ROUTINES
47CB: 138 *
47CB:A9 0C 139 LOAD2 LDA #>INIT
47CD:85 42 140 STA A4
47CF:A9 0B 141 LDA #<INIT
47D1:85 43 142 STA A4+1
47D3:A9 0C 143 LDA #>ILOAD
47D5:85 3C 144 STA A1
47D7:A9 4B 145 LDA #<ILOAD
47D9:85 3D 146 STA A1+1
47DB:A9 FF 147 LDA #>ELOAD
47DD:85 3E 148 STA A2
47DF:A9 4D 149 LDA #<ELOAD
47E1:85 3F 150 STA A2+1
47E3:A0 00 151 LDY #0
47E5:20 2C FE 152 JSR MOVE ; TO INIT
47E8: 153 *
47E8: 154 * INITIALIZE THE & HOOK
47E8: 155 *
47E8:20 0C 0B 156 JSR INIT ; IN AMPERJUMP
47E8:4C 6C D6 157 JMP CLEARC ; ASOFT CLEANS UP
47EE: 158 *
47EE: 159 * ADD LINNUM +1 TO PAGE ZERO
47EE: 160 * POINTER GIVEN BY X-REG.
47EE: 161 *
47EE:3B 162 ADD SEC
47EF:85 00 163 LDA #00,X
47F1:65 50 164 ADC LINNUM
47F3:95 00 165 STA #00,X
47F5:85 01 166 LDA #01,X
47F7:65 51 167 ADC LINNUM+1
47F9:95 01 168 STA #01,X
47FB:60 169 RTS

```

*** SUCCESSFUL ASSEMBLY: NO ERRORS

Apple Electronic Message Center

Electronic Message Center (E.M.C.)

by Rudy A. Guy
1114 W. 22nd Street
Erie, PA 16502

How many times have you found yourself standing in front of a terminal in an airport or a bus station watching a continuous flow of information on departure and arrival schedules? How about the cable television information channel? Have you found yourself watching endless scheduling notes and public service announcements? Now comes the big question: have you ever priced one of these "Message Center" systems? If you have, you know that the price is way out of the ball park for most individuals and institutions. But, since you have had the foresight (I assume) to own an Apple computer, a limited Electronic Message Center is only a few hours of key pounding away.

Electronic Message Center (E.M.C.) is written in Applesoft BASIC and requires at least 32K and one disk drive. Even though it is not as elaborate as some of the higher priced "systems", E.M.C. will get the job done at a fraction of the cost.

E.M.C. was developed at the request of a local high school teacher. It was his idea to have important notices and announcements displayed throughout the day on several different terminals. You can also use E.M.C. in your home or office for passing along messages and announcements to your family or associates. It's flexible, eye-catching, and fun.

My initial attempt was rather crude, as it did not allow for entry of commas, colons, or other forbidden characters within a message. Further, there was no way to select an individual group of messages — they were all added to one file. The final shortcoming was the lack of a formatting routine for displaying the messages. Happily, each one of these limitations has been overcome in the final E.M.C. and you'll find the solutions to be of value in your own programming efforts.

GETTING STARTED

When E.M.C. is run, the user will be presented with the program's main menu. From here it is possible to:

1. LOAD & DISPLAY A FILE
2. ADD TO A FILE
3. ADD A NEW FILE
4. CHANGE A FILE
5. DELETE A FILE
6. REVIEW NAME-FILE
7. EXIT PROGRAM

Since we are dealing with the initial run of E.M.C., it would be wise to select **option 3, ADD A NEW FILE**. I know that there will be those of you who will try one of the other options first just to see if the program will bomb. Well, go ahead and try. All that you will get is a **NO TITLES IN FILE** message and the program will return to the main menu. So, let's get back to option 3 (lines 3000 - 3170) and see how a new file is added.

The first item that the user will be prompted for is a **new file name of not more than ten characters in length**. Once this name is entered, the program checks for a duplicate file so that an older file will not be overwritten. If no duplicate is found, you will be prompted for your **message of up to 224 characters**. So that you won't have to guess if your message is too long, 224 underline characters will be displayed. Should an entry go beyond this length, an error message will be given and you will have to reenter the message. E.M.C. contains a text formatting routine for displaying the messages, so **you do not have to be concerned about words being split between lines** when messages are displayed. This formatting routine will be described later in the article.

INPUT ANYTHING

At this point in the program there is a subroutine that may be of interest, a BASIC "Input Anything" routine (lines 8000 - 8070). Rather than add on a machine language routine to allow input of forbidden characters, such as the comma and the colon, I opted to use an already existent feature of the Apple, **the keyboard buffer**.

Anytime that data is entered into the Apple via the keyboard, it is placed in the **keyboard buffer** which is located from decimal **memory location 512 through 767 (\$200-2FF)**. To get the input from the program into this buffer, it is necessary to call the **GETLN** subroutine in the Monitor which is located at **\$FD6A (-662)**. As soon as a **carriage return** is entered, the **GETLN** subroutine will terminate and program execution returns to BASIC line 8020. At this point each memory location is **PEEKed**, through the use of a loop, for the character value that it contains. If the value of the memory location does not equal **141**, a carriage return, its character equivalent is added to **ST\$**. If a carriage return is encountered, the loop is exited and program execution returns the add module at line 3090 where **ME\$(MC)** is made equivalent to **ST\$**.

The final bit of data that will be requested is a **date**, in the form, **MM/DD/YY**. (This is an optional item and may be omitted by entering a carriage return.) When all the data has been entered, you will be prompted to verify the data you have supplied. E.M.C. is designed to handle a **maximum of 100 messages**, but if this number is too small, it may be increased by changing the **DIM** statements in line 120.

ADDING TO AN EXISTING FILE

If it becomes necessary to add more messages to an existing file, **option 2** should be selected from the main menu. You will be prompted for the file to be accessed, the new message and the date. When all data has been entered, the new message will be added to the end of the message file. Since **E.M.C. uses sequential files**, this addition was made through the use of the DOS **"APPEND"** command. **If you are still using DOS 3.2 or 3.2.1, there is an error in the APPEND command which could cause some problems**. To alleviate this problem in **DOS 3.2 systems**, the following changes should be made to E.M.C.:

(ADD)

```
105 POKE 778,168: POKE 779,0:
    POKE 780,32: POKE 781,237:
    POKE 782,253
```

(CHANGE)

```
2100 PRINT CHR$(4)"APPEND "A$:
    PRINT CHR$(4)"WRITE "A$:
    PRINT ME$(MC): PRINT MD$(MC):
    CALL 778: PRINT:
    PRINT CHR$(4)"CLOSE"
```

This fix should take care of any problems. (Note: For more information on this fix, please refer to the International Apple Core's APNOTE of March 15, 1980.)

CHANGE A FILE

In order to **modify an existing file**, select **option 4** from the program menu. You will be prompted for the file that you wish to change. When this selection has been made, a modified "Input Anything" routine (lines 9000-9120) will read the file from the disk. Once all the messages have been read into memory, each will be displayed to the screen. You may elect to modify the message, leave the message unchanged, or purge the message from the file. When all changes have been made to the messages, the file will be rewritten, in corrected form, to the disk.

DELETE A FILE & REVIEW FILE NAMES

Option 5 permits the **deletion of any file** from the disk. You are prompted for the file name that is to be deleted. If the name entered is an existing file name, you will have to confirm your decision to delete the file by entering **"DEL"**. If an incorrect response is given at this point, you will be returned to the program menu.

Option 6, as implied by the menu, will display all message files currently stored in the **TITLES** text file. This option will display **38 titles, 2 columns of 19 names**, at a time to the screen.

DISPLAYING THE MESSAGES

When **option 1** is selected, and you select the file to be displayed, one of the strongest routines in the program is put to work — the **print formatter**. This routine, when used correctly, will eliminate the notorious split words which so often creep into text portions of programs. **Two variables must be set** before the formatting routine is accessed; they are **LM (left margin)** and **RM (right margin)**. **RM** must be set equal to the width of the screen; in this program the value is **32**. The following is a line by line description of how the routine works:

- 10010 Check to see if **A\$** is less than width of screen; if it is, print **A\$** and return.
- 10020 Set up descending loop, maximum length to 1.
- 10030 Set **C\$** equal to the **J** element of **A\$** and check for punctuation or a space. If **C\$** is a space or a punctuation mark then jump out of loop to line 10050.

Electronic Message Center (E.M.C.) (Cont.)

- 10040** Decrement counter by **NEXT** command.
- 10050** If **C\$** is a space, eliminate it and set **A1\$ = LEFT\$(A\$, J-1)** and **A\$ =** to remainder of original **A\$**. Go to print routine.
- 10060** If **C\$** is a punctuation mark and it is in the maximum position (**RM-LM=J**) then go check for the next character (10130). If the next character is a space then set **A1\$=LEFT\$(A\$,J)** and **A\$ =** to remainder of original **A\$** minus one character. Print **A1\$** and go to 10100.
- 10070** No space or punctuation encountered. The line must be broken at maximum length.
- 10080** Defaults here if no punctuation or space is encountered and there is no

more string to check. Go to final print routine at 10110.

- 10090** Print **A1\$** when no space or punctuation is found.
- 10100** Check to see if there is more of **A\$** to be formatted. If yes then go to start of new loop at line 10020.
- 10110** Formatting complete. Print **A\$** at **HTAB LM**.
- 10120** Jump out of subroutine; the job is done.
- 10130** Part of the routine to check for the type of character following a punctuation mark. If the remaining portion of **A\$** is shorter than or equal to the screen width, no check is needed. Go back to print routine in 10060.
- 10140** If next character is not a space then set **A1\$ = LEFT\$(A\$,J)** and **A\$ =** to the remainder of the original **A\$**. Print **A1\$** but keep cursor on same

line to avoid auto line feed at end of line. Jump out of subroutine and see if there is more to be formatted.

- 10150** Next character is a space, so return to 10060.

The complete file selected by the user is read into memory, so the disk will only be accessed at the beginning of the routine. The messages selected to be displayed will loop endlessly if so desired. **To terminate the display mode, a Control-C will get you back to the program menu.**

As with any program there is always a way of making it better adapted to its environment. One possible change, for those with real-time clocks, would be to eliminate the date and replace it with the current time of day. As I indicated earlier, E.M.C. is not as elaborate as the high priced message center, but the price can't be beaten.



```

10 REM *****
20 REM      EMC
30 REM      BY RUDY A. GUY
40 REM      COPYRIGHT (C) 1982
50 REM      BY MICRO-SPARC INC
60 REM      LINCOLN, MA. 01773
70 REM *****
100 POKE 768,104: POKE 769,168: POKE 770,104: POKE 771,166:
    POKE 772,223: POKE 773,154: POKE 774,72: POKE 775,152: POKE
    776,72: POKE 777,96
110 ONERR GOTO 32000
120 DIM T$(100),M$(100),D$(100)
130 TEXT : HOME : INVERSE : PRINT "++++++": FOR I = 1 TO 2
    2: PRINT "+ " : NEXT : PRINT "++++++": VTAB 1
140 VTAB 8: HTAB 2: PRINT "++++": VTAB 10: HTAB 3: PRINT "
    EMC": VTAB 12: HTAB 3: PRINT "BY": VTAB 14: HTAB 2: PRINT "
    R.GUY": VTAB 16: HTAB 2: PRINT "++++"
150 VTAB 1: FOR I = 8 TO 40: HTAB 1: PRINT "+": NEXT
160 FOR I = 2 TO 23: VTAB 1: HTAB 40: PRINT "+": NEXT
170 VTAB 24: FOR I = 8 TO 39: HTAB 1: PRINT "+": NEXT
180 POKE 2039,43
190 NORMAL : POKE 32,7: POKE 33,32: POKE 34,1: POKE 35,23:
    HOME
200 VTAB 4: A$ = "<MAIN MENU>": GOSUB 11000: VTAB 22: A$ =
    "(C) 1982 MICRO-SPARC, INC.": GOSUB 11000: VTAB 5
210 RESTORE : FOR I = 1 TO 7: READ A$(I): PRINT : A$ = A$(I)
    : GOSUB 11000: NEXT
220 PRINT : PRINT " SELECT ONE OF THE ABOVE:": GET A$: PRI
    NT A$: IF A$ < ">" OR A$ > "7" THEN HOME : GOTO 200
230 A = VAL (A$): ON A GOSUB 1000,2000,3000,4000,5000,6000,
    7000
240 TEXT : HOME : CLEAR : GOTO 110
1000 REM ::DISPLAY FILE:::
1010 HOME : VTAB 12: PRINT " FILE NAME ": FOR I = 1 TO 10
    : PRINT CHR$(95);: NEXT : PRINT : VTAB 12: HTAB 13: INPUT
    "": A$: IF A$ = "" THEN RETURN
1020 IF LEN (A$) > 10 THEN 1000
1030 IF FL = 1 THEN FL = 0: GOTO 1050
1040 GOSUB 15000
1050 FOR I = 1 TO CZ - 1: IF A$ = T$(I) THEN 1080
1060 NEXT : HOME : VTAB 12: PRINT " THAT FILE DOES NOT EXIS
    T.": PRINT : PRINT " TRY AGAIN ": INPUT YN$: IF LEFT$(YN$,
    1) = "Y" THEN FL = 1: GOTO 1010
1070 RETURN
1080 GOSUB 9000
1090 POKE 35,21
1100 HOME : VTAB 22: INVERSE : PRINT "-----"
    "": SPEED= 140: NORMAL
1110 POKE 232,3
1120 FOR I = 1 TO MC - 1: A$ = M$(I)
1130 INVERSE : VTAB 23: PRINT " ITEM # "I": HTAB 16: PRINT
    "DATE ": IF M$(I) = "" THEN PRINT " ": GOTO 1160
1140 IF LEN (M$(I)) < 8 THEN M$(I) = " " + M$(I): GOTO
    1140
1150 HTAB 31 - LEN (M$(I)): PRINT M$(I):
1160 NORMAL
1170 LM = 2: RM = 32
1180 FOR J = 0 TO 8: VTAB PEEK (232) + J: HTAB 1: CALL -
    868: FOR L = 1 TO 100: NEXT : NEXT : VTAB PEEK (232)
1190 GOSUB 10000
1200 IF PEEK (232) = 3 THEN POKE 232,12: GOTO 1220
1210 IF PEEK (232) = 12 THEN POKE 232,3
1220 NEXT
1230 FOR I = 1 TO 3000: NEXT
1240 FOR I = 21 TO 2 STEP - 1: VTAB 1: HTAB 1: CALL - 868
    : FOR J = 1 TO 100: NEXT : NEXT
1250 GOTO 1100
2000 REM ::ADD TO FILE:::
2010 HOME : VTAB 12: PRINT " FILE NAME ": FOR I = 1 TO 10
    : PRINT CHR$(95);: NEXT : PRINT : VTAB 12: HTAB 13: INPUT
    "": A$: IF A$ = "" THEN RETURN
2020 IF LEN (A$) > 10 THEN 2000
2030 IF FL = 1 THEN FL = 0: GOTO 2050
2040 GOSUB 15000

```

```

2050 FOR I = 1 TO CZ - 1: IF A$ = T$(I) THEN 2080
2060 NEXT : HOME : VTAB 12: PRINT " THAT FILE DOES NOT EXIS
    T.": PRINT : PRINT " TRY AGAIN ": INPUT YN$: IF LEFT$(YN$,
    1) = "Y" THEN FL = 1: GOTO 2000
2070 RETURN
2080 AD = 1: HOME : GOSUB 3090
2090 PRINT : PRINT " IS THIS CORRECT ": INPUT YN$: IF LEF
    T$(YN$,1) = "N" THEN 2080
2100 PRINT CHR$(4)"APPEND "A$: PRINT CHR$(4)"WRITE "A$:
    PRINT MES(MC): PRINT MDS(MC): PRINT CHR$(4)"CLOSE"
2110 PRINT : PRINT " ANOTHER ": INPUT YN$: IF LEFT$(YN$,
    1) = "N" THEN AD = 0: RETURN
2120 HOME : GOSUB 3090: GOTO 2090
3000 REM ::ADD NEW FILE:::
3010 GOSUB 15000: REM READ NAME FILE
3020 HOME
3030 VTAB 3: PRINT " FILE NAME ": FOR I = 1 TO 10: PRINT
    CHR$(95);: NEXT : PRINT : VTAB 3: HTAB 13: INPUT "": T$(CZ
    + 1): IF LEN (T$(CZ + 1)) > 10 THEN HOME : GOTO 3030
3040 IF T$(CZ + 1) = "" THEN 3170
3050 MC = 1
3060 IF CZ = 0 THEN 3090
3070 FOR I = 1 TO CZ: IF T$(CZ + 1) = T$(I) THEN HOME :
    VTAB 12: A$ = "THAT TITLE ALREADY EXISTS.": GOSUB 11000: FOR
    I = 1 TO 3000: NEXT : GOTO 3020
3080 NEXT
3090 INVERSE : VTAB 5: HTAB 2: PRINT "ENTER MESSAGE ": NOR
    MAL : PRINT : FOR I = 1 TO 224: PRINT CHR$(95);: NEXT : PR
    INT : VTAB 7: HTAB 1: GOSUB 8000: MES(MC) = ST$
3100 VTAB 15: PRINT " DATE (MM/DD/YY) ": FOR I = 1 TO 8:
    PRINT CHR$(95);: NEXT : PRINT : VTAB 15: HTAB 19: INPUT "":
    MDS(MC)
3120 IF AD = 1 THEN RETURN
3130 PRINT : INPUT " IS THIS CORRECT ?": YN$: IF LEFT$(YN$,
    1) = "N" THEN HOME : GOTO 3090
3140 PRINT : PRINT " ANOTHER ?": INPUT "": YN$: IF LEFT$(
    YN$,1) = "Y" THEN MC = MC + 1: HOME : GOTO 3090
3150 PRINT CHR$(4)"OPEN "T$(CZ + 1): PRINT CHR$(4)"WRI
    TE "T$(CZ + 1): FOR I = 1 TO MC: PRINT MDS(I): PRINT MDS(I)
    : NEXT : PRINT CHR$(4)"CLOSE"
3160 PRINT CHR$(4)"OPEN TITLES: PRINT CHR$(4)"POSITION
    TITLES,R"CZ - 1: PRINT CHR$(4)"WRITE TITLES": PRINT T$(C
    Z + 1): PRINT "****": PRINT CHR$(4)"CLOSE"
3170 RETURN
4000 REM ::CHANGE A FILE:::
4010 HOME : VTAB 12: PRINT " FILE NAME ": FOR I = 1 TO 10
    : PRINT CHR$(95);: NEXT : PRINT : VTAB 12: HTAB 13: INPUT
    "": A$: IF A$ = "" THEN RETURN
4020 IF LEN (A$) > 10 THEN 4000
4030 IF FL = 1 THEN FL = 0: GOTO 4060
4040 GOSUB 15000
4050 F$ = A$
4060 FOR I = 1 TO CZ - 1: IF A$ = T$(I) THEN 4090
4070 NEXT : HOME : VTAB 12: PRINT " THAT FILE DOES NOT EXIS
    T.": PRINT : PRINT " TRY AGAIN ": INPUT YN$: IF LEFT$(YN$,
    1) = "Y" THEN FL = 1: GOTO 4000
4080 RETURN
4090 GOSUB 9000
4100 FOR I = 1 TO MC - 1: A$ = M$(I)
4110 HOME : VTAB 3: PRINT " ITEM # "I: PRINT : LM = 2: RM = 3
    2: GOSUB 10000: PRINT : PRINT " DATE ": MDS(I)
4120 VTAB 17: PRINT " CHANGE ": PRINT " (D)ATE (M)ESSAGE (
    P)URGE": PRINT " (N)EXT": VTAB PEEK (37) - 2: HTAB 10: GET
    YN$: PRINT YN$
4130 IF YN$ = "N" THEN 4180
4140 CH = 1
4150 IF YN$ = "D" THEN 4260
4160 IF YN$ = "P" THEN 4330
4170 IF YN$ = "M" THEN 4290
4180 IF MC = 1 THEN 4210
4190 IF I > = MC - 1 THEN 4210

```

continued on page 191

THE SHAPE

The Shape

by Bill Fortenberry
2906 36 Ave.
Meridian, MS 39301

One of the things that separates the Apple II from other micro computers is its powerful graphics, and at the core of these capabilities is the shape table. The shape table provides the user with the ability to define a HIRES graphic image, identify it by an index number, and then rotate, scale, and draw the shape in any of the Apple's HIRES colors. Creating a shape table is comparable to doing time for armed robbery. It's a long, tedious process where one coding or typing error can put you right back at the beginning.

THE SHAPE to the rescue! THE SHAPE is an Applesoft program that allows you to create, view, edit, and save shape tables. The shapes built by the program are in the standard hex format and can be used in any Applesoft, Integer or assembly language program.

HOW TO BUILD A SHAPE TABLE

The complete instructions for building a shape table are found in the Applesoft reference manual (Programmer's Aid #1 manual for Integer), but to get everyone going, I will give a summary of them here. The first step is to decide on a shape and draw it on graph paper. Now pick a starting point and begin drawing arrows that point from one dot to the next, and only turn on 90 degree angles (these arrows are usually called **the vectors** of the shape). The arrows must then be coded as a 3 digit binary number (which must then be coded as an 8 bit byte). One of the problems with shapes is the way you cram these 3 bit numbers down to 8 bits (THE SHAPE does this for you).

After each byte is set, you must build the **shape table index**. Like the index in a book, the shape table index tells where each shape is located. When you finish the index, you are ready to call the monitor, enter the table, exit the monitor, test the shape, see that it's wrong, look at your arrows again, look at your numbers again, check each byte again, find one mistake, enter the monitor, type the table, exit the monitor, test the table, see that it's wrong, look at your arrows again, look at your numbers again, check the bytes again, sell your Apple, buy a Northstar. If doing all of the above does not appeal to you, read on.

LO-RES TO HI-RES

Late one night, after I had been trying for about 4 hours to set up a shape table for a new game, I decided to write a program to do the actual coding and typing for me. Since I wasn't going to code the vectors myself, I needed a way to get the shape some place where the computer could "see" it. I chose the LO-RES screen mainly because of the availability of the **SCRN** function. This enabled me to "see" each dot on the screen.

Next I had to get the shape on the screen. Out of this need came a simple horizontal and vertical line drawing routine. (I always draw a shape on graph paper so the lines are easy to see.) Now, how would I code the shape? Since Apple shapes are picky about moving up two

consecutive (no plot) spaces in a row, and since I had the problem of trying to put 9 bits of information in an 8 bit word, I decided to **scan the shape right to left, and top to bottom. This totally eliminates any vectors with a value of zero** (the signal for the end of the shape table), and when poking the vectors in memory I only put two vectors in a byte.

My goal with THE SHAPE was to write a program that would give the user a completed shape table (index + definitions) while requiring minimum input.

INSTRUCTIONS FOR THE SHAPE

OPTION ONE — BUILD A SHAPE TABLE

When you run the program, a menu will appear with 2 options. Let's take option one and build a table (I'll talk about option two later.) First, the text page will disappear and the blank graphics page will be displayed. At the bottom of the screen, you will be asked if you want the **turtle** or **line mode**. Here is how the **line mode** works.

After you pick **line mode**, the screen will display:

HLIN FROM PLOT

This means that you are in the **horizontal line mode** with the **plot on**. Type a **5** and press return. The Apple replies with **TO**. Now type a **9** and press return. Next the screen displays **ON**. Type **7** and press return. This will draw a **horizontal line from 5,7 to 9,7** (it's the big red line toward the top of the screen).

Now type a **V** and hit return. The screen should say:

VLIN FROM PLOT

You are now in the **vertical line mode** (these modes act just like **HLIN** and **VLIN** in BASIC). Try the same sequence of numbers as above. Did you get a vertical line?

Type an **E** and return. The screen will display:

HLIN FROM ERASE

Any lines you draw now will be black (or erased). Notice that you returned to the **HLIN** mode.

Now try an **F** and press return. Where did that white square come from? This is the **free cursor mode**. Type some **I's**, **J's**, **K's** and **M's** and the cursor should move around the screen. Press the **P** key. Now when you move the cursor it should leave a red dot. Now put the cursor in the middle of one of your lines, hit the **E** key, and move the cursor. It should have left a black dot. When you press **P** or **E** in the **free mode**, it will **plot** or **erase** the point the cursor is on. By the way, the **coordinates of the cursor** are displayed at the bottom of the screen. The free mode is designed for touching up shapes and finding coordinates for the line statements. To exit from the free mode, press **Q**. In any mode, you may press the question mark key to get a list of available commands.

When you are in the HLIN or VLIN mode, pressing Q will cause the program to start coding your shape.

The screen will now display:

LOCATING SHAPE EDGES

The program is now finding the top, bottom and sides of the shape. After a while, it will display:

CODING PLOT VECTORS

The program is now looking at each dot on the screen and coding it as a number. Then the screen will show:

POKING SHAPE DEFINITION

The numbers that represent your shape are being combined and poked into memory.

When the screen shows:

THIS DEFINITION STARTS ON THE WHITE DOT. PRESS RETURN TO PLOT SHAPE

You should find the white dot and mark its position on your graph paper. This is where the Apple will begin to draw your shape (the coordinates in the **DRAW** statement).

SHAPE DISPLAY

We are now ready to see the shape. Press Return and enter the **scale** and **rotation** just as you would in a BASIC program. Choose the color selection with the **first letter** of the desired color and press Return. You should now see your shape in glorious HI-RES graphics. When you press Return, the program will ask if you want to draw the shape again. If you press **Y**, you can enter new coordinates, etc. and redraw it. If you answer **N**, the computer will ask if you want to edit the shape. If you didn't like the way it looked, answer **Y** and you will go back to the edit mode. If you are satisfied with the shape type an **N**.

Now the Apple will ask if you want to keep this shape as a **base** for the next shape. If, for example, you are building a series of shapes that will show a man walking. Since only the legs will move, why should you have to redraw the head and torso for each figure? With THE SHAPE you don't have to. Just answer **Y** to the question and you will be put right back into the Enter mode. (**Except** that the last shape is now stored safely away) and you are looking at the next shape in line. Now just change the position of the legs, and proceed to the next shape in the sequence.

THE TURTLE MODE

If you select the **Turtle mode** instead of the Line mode, the screen will display:

MOVE TO START AND PRESS 'P'

Use the **'I', 'J', 'K' and 'M'** keys to move the cursor to where you want to start your shape, and then press **'P'**. The screen will display:

PLOT ON X=xx Y=yy

This tells you that the **plotting** is turned ON and gives you the values for the **x** and **y** coordinates. Just keep pressing the cursor keys to draw your shape. To turn the plot OFF, just press **'E'**. If you make a mistake while drawing in the turtle mode, the **ESC** key will cause the cursor to **back up** along your shape and erase the errors.

LINE VS. TURTLE

In the **line mode**, the computer starts to code your shape when you push the 'Q' key. In the **turtle mode**, the computer follows every plot as you make it. Because of this, the turtle mode will code a shape faster (if you draw it with a minimum of moves).

With the **turtle mode**, you don't get to use one shape as a base for another. Also, if you choose to edit a shape, you must start over with a blank screen.

The beginner will feel safer with the line mode, while an experienced programmer will want to use the turtle mode to draw more efficient shapes. No matter which mode you use, the computer will generate an index for you. After you have finished your table, the computer will give you the commands to save your shape Table.

OPTION TWO — THE POKE CALCULATOR

When you build a Shape Table with this program, the Shape Table is stored in HI-RES page one. When you load the table back into memory to use with your own program, you must put it in another memory location.

Remember that when you use a Shape Table with Applesoft, you must poke the address of the table into locations 232 and 233. The POKE calculator will make this easier for you. When you select Option Two from the main menu and enter the address where you want to put the table, the program will show you what POKES to use in your program.

MEMORY USAGE BY THE SHAPE

In order for this program to work properly you must have a 48K machine. The program resides from LOMEM up to HI-RES page one. Page one of HI-RES is reserved for the shape table as it is being built. Page Two is used to show the shape, and all memory above page

Two is set aside for variable storage. This program does **not** contain an **ONERR GOTO** section (to conserve space), but input is checked and most bad input is ignored. You should become familiar with the program before building a large shape Table.

VARIABLES USED IN THE PROGRAM

Most of the variables serve double duty to conserve memory, but here are some of the primary variables:

AS	— input string
SN	— number of shapes in the table
SS	— Shape Table starting location
G	— array to hold shape vectors
SD	— array that holds the starting location of each definition
SC	— loop counter (counts up to # of shapes in table)
QQ	— throw away variable
B & C	— input in line mode
X & Y	— screen coordinates
ZC	— old screen color
CZ	— current screen color
X,Y & Z	— misc. pointers

THE PROGRAM LOGIC

The program does not have many comments because of memory limits. The table below should help you decipher the program section and their logic.

80-180	These lines display the title page
200-270	Display the menu and accept the user's input
290-390	Find out how many shapes are in the table
410-460	The main program loop
480-710	Draw a horizontal line or branch to Vlin or Free Cursor
730-960	Same as above for vertical lines

980-1420	Free cursor + turning ON or OFF individual points
1190-1360	Make an imaginary box around the figure
1380-1630	Scan shape and fill array G with shape vectors
1650-1830	Poke the shape vectors into memory
1850-1980	Let the user view the shape and set up the index
2000-2180	Let the user edit and use the shape again
2200	Foot of the main loop
2220-2310	Print details about the table and BSAVE command
2320-2400	Plot and erase subroutines for the line mode
2420	General purpose text centering subroutine
2440-2520	Calculate POKE values for the user
2540-2610	A short hex to decimal converter
2630-2970	The code for the Turtle mode

The Shape is by no means complete. Many things were intentionally omitted because they had already been done or would have required program chaining. The shape coding algorithm could also be improved, although it works fine in the present form. Hopefully, THE SHAPE will give other programmers a basis for developing better shape table utilities.



```

10 REM *****
20 REM * THE SHAPE *
30 REM * BILL FORTENBERRY *
40 REM * COPYRIGHT (C) 1982 *
50 REM * BY MICRO-SPARC INC *
60 REM * LINCOLN, MA. 01773 *
70 REM *****
80 LOMEM: 24576
90 TEXT : HOME : PRINT "*****"
*****
100 FOR I = 2 TO 12: VTAB I: PRINT "": TAB( 40); "": NEXT
110 VTAB I
120 PRINT "*****"
130 INVERSE :AS = " " : VTAB 3: GOSUB 2420: VTAB 5
: GOSUB 2420
140 VTAB 4: INVERSE :AS = " THE SHAPE ": GOSUB 2420: NORMAL
150 VTAB 7:AS = "APPLE II SHAPE TABLE UTILITY": GOSUB 2420
160 VTAB 9:AS = "BY: BILL FORTENBERRY": GOSUB 2420
170 AS = "(C) 1982 BY MICRO-SPARC, INC.": VTAB 11: GOSUB 242
0
180 POKE 34,13
190 REM
MENU
200 VTAB 15: PRINT "WOULD YOU LIKE TO": PRINT : PRINT "
1. BUILD A SHAPE TABLE": PRINT " 2. CALCULATE POKE
VALUES": PRINT " 3. INSTRUCTIONS & DEMO"
210 PRINT : INPUT "TYPE THE NUMBER (1-3) OF YOUR CHOICE
=>":AS
220 IF AS = "" THEN 200
230 A = VAL (AS)
240 IF A < 1 OR A > 3 THEN 200
250 HOME
260 IF A = 3 THEN PRINT CHR$( 4);"RUN SHAPE INSTRUCTIONS"
270 IF A = 2 THEN 2430
280 REM
GATHER DATA
290 VTAB 17: INPUT "HOW MANY SHAPES IN THIS TABLE ":SN$
300 IF SN$ = "" THEN 290
310 SN = VAL (SN$)
320 IF SN < = 0 THEN 290
330 SS = 10300
340 REM POKE SHAPE LOCATION

```

```

350 POKE 232,SS - INT (SS / 256) * 256
360 POKE 233, INT (SS / 256)
370 DIM G(1445),SD(SN + 2)
380 SD(1) = SS + 2 + SN * 2
390 GR : POKE 34,20
400 REM
LOOP FOR # OF SHAPES
410 FOR SC = 1 TO SN
420 QQ = FRE (0)
430 GR : HOME : POKE 34,20: HOME
440 HOME : PRINT "SHAPE #":SC: PRINT "TURTLE OR L)LINE MODE
(T/L) ": INPUT "":AS: IF AS = "T" THEN 2620
450 COLOR= 1:F = 0
460 HOME : PRINT TAB( 33)"PLOT": POKE 33,30
470 REM
HLIN MODE
480 HOME : PRINT "HLIN FROM ": INPUT "":AS
490 IF AS = "" THEN 480
500 IF AS = "V" THEN HOME : GOTO 730
510 IF AS = "F" THEN 980
520 IF AS = "P" THEN 2330
530 IF AS = "E" THEN 2370
540 IF AS = "Q" THEN 1190
550 IF AS = "?" THEN 2980
560 PRINT "TO ": INPUT "":B$
570 IF B$ = "" THEN 560
580 IF B$ = "?" THEN 2980
590 B = VAL (B$)
600 IF B > 38 THEN B = 38
610 IF B < 1 THEN B = 1
620 PRINT "ON ": INPUT "":C$
630 IF C$ = "?" THEN 2980
640 IF C$ = "" THEN 620
650 C = VAL (C$)
660 IF C > 38 THEN C = 38
670 IF C < 1 THEN C = 1
680 A = VAL (AS)
690 IF A < 1 THEN A = 1
700 IF A > 38 THEN A = 38
710 HLIN A,B AT C: PRINT : HOME : GOTO 480
720 REM
VLIN MODE

```

continued on next page

The Shape (Cont.)

```

730 INPUT "VLIN FROM ";A$
740 IF A$ = "" THEN 730
750 IF A$ = "H" THEN HOME : GOTO 480
760 IF A$ = "F" THEN 980
770 IF A$ = "P" THEN 2330
780 IF A$ = "E" THEN 2370
790 IF A$ = "Q" THEN 1190
800 IF A$ = "?" THEN 2980
810 INPUT "TO ";B$
820 IF B$ = "?" THEN 2980
830 IF B$ = "" THEN 810
840 B = VAL (B$)
850 IF B > 38 THEN B = 38
860 IF B < 1 THEN B = 1
870 INPUT "AT ";C$
880 IF C$ = "?" THEN 2980
890 IF C$ = "" THEN 870
900 C = VAL (C$)
910 IF C > 38 THEN C = 38
920 IF C < 1 THEN C = 1
930 A = VAL (A$)
940 IF A < 1 THEN A = 1
950 IF A > 38 THEN A = 38
960 VLIN A,B AT C: PRINT : HOME : GOTO 730
970 REM

FREE CURSOR MODE
980 X = 20:Y = 20
990 ZC = SCRN( X,Y): COLOR= 15: PLOT X,Y
1000 POKE 33,40
1010 HOME : PRINT "X=";X;" Y=";Y
1020 GET A$
1030 IF A$ = "Q" THEN COLOR= ZC: PLOT X,Y: GOTO 2330
1040 IF A$ = "P" THEN ZC = 1: GOTO 1020
1050 IF A$ = "E" THEN ZC = 0: GOTO 1020
1060 IF A$ = "?" THEN 3040
1070 XX = X:YY = Y
1080 IF A$ = "I" AND Y > 1 THEN YY = Y - 1: GOTO 1130
1090 IF A$ = "M" AND Y < 38 THEN YY = Y + 1: GOTO 1130
1100 IF A$ = "J" AND X > 1 THEN XX = X - 1: GOTO 1130
1110 IF A$ = "K" AND X < 38 THEN XX = X + 1: GOTO 1130
1120 GOTO 1020
1130 CZ = SCRN( XX,YY)
1140 COLOR= 15: PLOT XX,YY: COLOR= ZC: PLOT X,Y:X = XX:Y = YY
1150 ZC = CZ
1160 HOME : PRINT "X=";X;" Y=";Y
1170 GOTO 1020
1180 REM

BOX IN SHAPE
1190 L = 0:Z = 0
1200 POKE 33,40: HOME
1210 PRINT "LOCATING SHAPE EDGES"
1220 FOR D = 1 TO 39: IF SCRN( L + 1,D) = 1 THEN Z = 1
1230 NEXT
1240 IF Z = 0 THEN L = L + 1: GOTO 1220
1250 J = 39:Z = 0
1260 FOR D = 1 TO 39: IF SCRN( J - 1,D) = 1 THEN Z = 1
1270 NEXT
1280 IF Z = 0 THEN J = J - 1: GOTO 1260
1290 Z = 0:K = 39
1300 FOR D = 1 TO 39: IF SCRN( D,K - 1) = 1 THEN Z = 1
1310 NEXT
1320 IF Z = 0 THEN K = K - 1: GOTO 1300
1330 Z = 0:I = 0
1340 FOR D = 1 TO 39: IF SCRN( D,I + 1) = 1 THEN Z = 1
1350 NEXT
1360 IF Z = 0 THEN I = I + 1: GOTO 1340
1370 REM

CODE SHAPE
1380 Z = 1:D = 1:X = L:Y = I + 1
1390 PRINT "CODING PLOT VECTORS"
1400 X = X + 1: SX = X: SY = Y: IF SCRN( X,Y) = 0 THEN 1400
1410 BX = X: BY = Y
1420 GOTO 1440
1430 X = X + D
1440 IF D = 1 AND X = J - 1 AND SCRN( X,Y) = 1 THEN G(Z) = 6:Y = Y + 1:D = - D:Z = Z + 1: GOTO 1490
1450 IF D = 1 AND X = J - 1 AND SCRN( X,Y) = 0 THEN G(Z) = 2:Y = Y + 1:D = - D:Z = Z + 1: GOTO 1490
1460 IF D = - 1 AND X = L + 1 AND SCRN( X,Y) = 0 THEN G(Z) = 2:Y = Y + 1:D = - D:Z = Z + 1: GOTO 1490
1470 IF D = - 1 AND X = L + 1 AND SCRN( X,Y) = 1 THEN G(Z) = 6:Y = Y + 1:D = - D:Z = Z + 1: GOTO 1490
1480 GOTO 1580
1490 IF Y > = K THEN 1630
1500 F = 0
1510 FOR B = L TO J
1520 IF SCRN( B,Y) = 1 THEN F = 1
1530 NEXT B
1540 IF F = 0 THEN G(Z) = 2:Y = Y + 1:Z = Z + 1: GOTO 1490
1550 F = 0
1560 IF Y > = K THEN 1630
1570 GOTO 1440
1580 IF SCRN( X,Y) = 1 AND D = 1 THEN G(Z) = 5:Z = Z + 1

```

```

1590 IF SCRN( X,Y) = 1 AND D = - 1 THEN G(Z) = 7:Z = Z + 1
1600 IF SCRN( X,Y) = 0 AND D = 1 THEN G(Z) = 1:Z = Z + 1
1610 IF SCRN( X,Y) = 0 AND D = - 1 THEN G(Z) = 3:Z = Z + 1
1620 GOTO 1430
1630 G(Z) = 10
1640 REM

POKE SHAPE
1650 A = SD(SC)
1660 T = A
1670 Z = 0
1680 PRINT "POKING SHAPE DEFINITION"
1690 Z = Z + 1
1700 IF G(Z) = 10 THEN 1790
1710 K = G(Z):Z = Z + 1
1720 IF G(Z) = 10 THEN 1780
1730 K = K + G(Z) * 8:Z = Z + 1
1740 IF G(Z) = 10 THEN 1780
1750 IF G(Z) < 4 AND G(Z) > 0 THEN K = K + G(Z) * 64:Z = Z + 1
1760 Z = Z - 1
1770 POKE A,K:A = A + 1: GOTO 1690
1780 POKE A,K:A = A + 1
1790 POKE A,0
1800 PRINT "": REM CTRL-G
1810 COLOR= 15: PLOT SX,SY
1820 HOME : PRINT "THIS DEFINITION STARTS ON THE WHITE DOT."
1830 VTAB 24: INPUT "PRESS RETURN TO PLOT SHAPE ";A$
1840 REM

DRAW SHAPE
1850 HOME
1860 INPUT "SCALE = ";G$:G = VAL (G$): IF G < 1 THEN 1860
1870 SCALE= G: INPUT "ROT = ";G$:G = VAL (G$): ROT= G
1880 HOME : PRINT "G)REEN P)URPLE O)RANGE B)LUE W)HITE"
1890 : PRINT : INPUT "ENTER FIRST LETTER OF COLOR ";A$
1890 IF A$ < > "G" AND A$ < > "W" AND A$ < > "B" AND A$ < > "P" AND A$ < > "O" THEN 1880
1900 REM POKE THE DIRECTORY
1910 POKE SS,SN: POKE SS + 1,0
1920 X = SS + 1
1930 FOR I = 1 TO SC
1940 B = SD(I) - SS
1950 X = X + 1:TA = INT (B / 256)
1960 POKE X,B - TA * 256
1970 X = X + 1: POKE X,TA
1980 NEXT I
1990 REM

EDIT THE SHAPE
2000 IF A$ = "W" THEN HCOLOR= 3
2010 IF A$ = "G" THEN HCOLOR= 1
2020 IF A$ = "P" THEN HCOLOR= 2
2030 IF A$ = "O" THEN HCOLOR= 5
2040 IF A$ = "B" THEN HCOLOR= 6
2050 HGR2 : DRAW SC AT 140,40
2060 INPUT A$: POKE - 16298,0: POKE - 16300,0: POKE - 16301,0
2070 SD(SC + 1) = A + 1
2080 IF TM = 0 THEN COLOR= 1: PLOT BX,BY
2090 HOME
2100 INPUT "DRAW THIS SHAPE AGAIN (Y/N) ";A$: IF A$ = "" THEN 2090
2110 IF A$ = "Y" THEN 1850
2120 HOME
2130 INPUT "EDIT THIS SHAPE (Y/N) ";A$: IF A$ = "" THEN 2130
2140 IF A$ = "Y" AND TM = 1 THEN 2620
2150 IF A$ = "Y" THEN 450
2160 IF TM = 1 THEN 2200
2170 HOME : INPUT "USE THIS SHAPE AS A BASE FOR ANOTHER (Y/N)";A$
2180 IF A$ = "Y" AND SC < SN THEN SC = SC + 1: GOTO 450
2190 REM

NEXT SHAPE
2200 NEXT SC
2210 REM

TABLE DATA
2220 TEXT
2230 HOME : PRINT "SHAPE TABLE DATA"
2240 PRINT : PRINT : PRINT "SHAPE TABLE STARTS AT ";SS
2250 PRINT : PRINT "AND ENDS AT ";SD(SC)
2260 PRINT : PRINT "TABLE LENGTH IS ";SD(SC) - SS + 1
2270 REM SHAPE LOCATION
2280 PRINT : PRINT "SAVE SHAPE WITH THIS COMMAND -"
2290 PRINT
2300 PRINT "BSAVE "; INVERSE : PRINT "FILE NAME";: NORMAL
2310 : PRINT "A ";SS;"L ";SD(SC) - SS + 1
2320 END
2330 REM

PLOT & ERASE SUBS
2330 POKE 33,40: HOME :
2340 PRINT TAB( 33)"PLOT": POKE 33,30
2350 COLOR= 1
2360 HOME : GOTO 480

```

continued on page 192

DISK MAP

DISK MAP

by James O. Church
20 Placid St.
Trumbull, CT 06611

Sometimes it's nice to do a little extra with your Apple even though it doesn't serve a very useful purpose. It is useful to know how many free sectors there are left on your diskette, but there's a little frosting on the cake if you can have a picture of the tracks and see which ones are being used and which ones are free. I believe anything which increases one's understanding of the Apple is justification enough to do something like DISK MAP.

TRACK BIT MAPS

Boot up DOS with a freshly INITIALIZED disk, enter the monitor with **CALL - 151** and type **B3F3.B47F<RTN>**. You will be looking at a memory dump of what are known as **Track Bit Maps**. Each track on the disk you booted is represented by 4 bytes. **Track 0** is represented by the first 4 bytes you see, and what you see is **00 00 00 00**.

The first two bytes are used in these Track Bit Maps, and the second two bytes are unused. Whenever a bit is "on" (i.e., equal to 1) in a Track Bit Map it means there is an unused sector on the track. In the dump (list 1) accompanying this article you will observe that **Track 0, 1 and 2** are all zero bytes, meaning that there are **no empty sectors** on these three tracks. What's there? DOS.

Track 3, however, has **FF FF 00 00** in the Track Bit Map for that track. That means that the bits look like this in the first two bytes. **11111111 11111111**. (We can disregard the second two bytes. They are always zero and not used.) The meaning of this is that all 16 sectors on track 3 are free.

If you find **track 17**, you will find it says **00 00 00 00**, meaning it is used. That is because the disk directory is usually there.

COUNTING SECTORS

All we must do to count the sectors which are used, then, is count the number of bits in the 35 Track Bit Maps which are "off" (i.e., equal to 0), and we can ignore the second two bytes of each Track Bit Map.

My program not only counts the number of bits that are "off" in all the Track Bit Maps, but also puts the number of these bits found into a table for each track.

HOW IT WORKS

How does one count the bits which are "off" in a byte? The easiest way is to put the byte into the accumulator and shift it left with the **ASL** instruction. This puts the first bit into the **CARRY BIT** which can then be tested with the **BCS** test. If the **CARRY BIT** is set, it means that the sector is free and we don't count it. Obviously, its up to me to decide whether to count "on" bits or "off" bits. I arbitrarily decided to count off bits (or used sectors). I place the **result** of the count in the table for each track, and the total result in **location \$08 and \$09**. For each byte we must count all 8 bits.

THE GRAPHICS

The graphics part is relatively simple. I first put a dark color on the screen in 35 columns, representing the 35 tracks in 16 locations in each column. Then I have the program consult the table I made for each track and enter a light color in the column beginning at a point which shows graphically how many sectors are USED. Actually, the map may not be an accurate indication of **which sectors** on a track are free. It only provides a display of **how many sectors** of a track are free and how many are used. Generally speaking, if one sees a lot of light color, he knows that the disk is relatively empty.

I use the program as a HELLO program a lot and find it helpful when I wonder if there is room on the disk for saving something I am about to create. Instead of having to find the disk with the **FREE SPACE** program on it, I can **RUNHELLO**. For this reason, I turned the machine language into **DATA** statements (list 2) so it would always be there.

Alternatively, you may prefer to **BLOAD** Disk Map when you wish to use it. If you enter it at **\$4000** (list 3) and it is in machine language, you can probably **BLOAD** it and **CALL** it while you have an Applesoft program in memory. This would allow you to determine disk space without bombing your Apple program (unless your program uses the memory block beginning at location hex \$4000).



LIST 1

```
*B3F3.B47F
B3F3- 00 00 00 00 00 00 FF
B3F8- 00 00 00 00 00 00 FF
B400- FF 00 00 FF FF 00 00
B408- FF 00 00 FF FF 00 00
B410- FF 00 00 3F 00 00 FF
B418- FF 00 00 1F FF 00 3F
B420- FF 00 00 1F FF 00 7F
B428- FF 00 00 7F FF 00 0F
B430- FF 00 00 7F FF 00 00
B438- 00 00 00 FF FF 00 FF
B440- FF 00 00 FF FF 00 0F
B448- FF 00 00 FF FF 00 0F
B450- FF 00 00 FF FF 00 FF
B458- FF 00 00 FF FF 00 FF
B460- FF 00 00 3F FF 00 07
B468- FF 00 00 FF FF 00 FF
B470- FF 00 00 FF FF 00 FF
B478- FF 00 00 3F 00 00 00
```

LIST 2

```
1 REM *****
2 REM * << DISK MAPPER >> *
3 REM * BY JAMES O. CHURCH *
4 REM * COPYRIGHT (C) 1982 *
5 REM * BY MICRO-SPARC INC *
6 REM * LINCOLN, MA. 01773 *
7 REM * ALL RIGHTS RESERVED *
8 REM *****
9 HOME
10 VTAB 10: PRINT "PLEASE WAIT ONE
    MOMENT FOR DISK MAP"
90 PRINT CHR$(4); "VERIFY HELLO"
100 FOR I = 0 TO 299: READ A: POKE
    16384 + I, A: NEXT
110 CALL 16384
120 A = PEEK (9) * 256 + PEEK (8)
130 PRINT : PRINT "SECTORS FREE "; 5
    60 - A; " SECTORS USED "; A
140 PRINT "LIGHT COLOR AREA IS FREE
    ";
150 GET A$
160 TEXT : HOME
1000 DATA 169,243,133,6,169
1001 DATA 179,133,7,162,8
1002 DATA 160,0,140,214,64
1003 DATA 140,215,64,177,6
1004 DATA 10,176,3,238,214
1005 DATA 64,202,208,247,162
1006 DATA 8,200,177,6,10
1007 DATA 176,3,238,214,64
1008 DATA 202,208,247,174,215
1009 DATA 64,173,214,64,157
1010 DATA 216,64,238,215,64
```

```
1011 DATA 173,215,64,201,35
1012 DATA 240,23,24,165,6
1013 DATA 105,4,133,6,165
1014 DATA 7,105,0,133,7
1015 DATA 162,8,160,0,140
1016 DATA 214,64,76,18,64
1017 DATA 173,80,192,173,86
1018 DATA 192,173,83,192,32
1019 DATA 54,248,169,2,141
1020 DATA 213,64,169,0,141
1021 DATA 212,64,169,2,32
1022 DATA 100,248,173,212,64
1023 DATA 172,213,64,32,0
1024 DATA 248,24,173,212,64
1025 DATA 105,1,141,212,64
1026 DATA 201,32,144,234,169
1027 DATA 0,141,212,64,24
1028 DATA 173,213,64,105,1
1029 DATA 141,213,64,201,37
1030 DATA 144,216,32,251,64
1031 DATA 169,20,133,37,32
1032 DATA 34,252,32,66,252
1033 DATA 169,2,133,36,169
1034 DATA 176,32,237,253,169
1035 DATA 19,133,36,169,196
1036 DATA 32,237,253,32,142
1037 DATA 153,169,0,133,8
1038 DATA 133,9,162,34,189
1039 DATA 216,64,24,101,8
1040 DATA 133,8,165,9,103
1041 DATA 0,133,9,202,16
1042 DATA 239,96,32,37,16
1043 DATA 35,16,16,16,16
1044 DATA 16,16,10,0,0
1045 DATA 0,7,16,2,16
1046 DATA 16,16,16,16,2
1047 DATA 0,0,0,0,0
1048 DATA 2,0,12,8,3
1049 DATA 3,11,0,16,16
1050 DATA 16,169,15,32,100
1051 DATA 248,162,0,160,2
1052 DATA 140,213,64,189,216
1053 DATA 64,201,16,240,20
1054 DATA 10,141,212,64,172
1055 DATA 213,64,32,0,248
1056 DATA 238,212,64,173,212
1057 DATA 64,201,32,144,243
1058 DATA 238,213,64,232,224
1059 DATA 35,144,221,96,0
```

LIST 3

```
Copyright (c) 1982
by Micro-SPARC Inc.
Lincoln, MA. 01773
All Rights Reserved

:ASM
1000 *
1010
1020 * .OR $4000
1030 * SHOW 35 TRACKS AND HOW MUCH OF
1040 * EACH TRACK IS USED ON THE DISK
1050 *
1060 * TRACK BIT MAPS $B3F3 TO $B47E
1070 *
0006- 1080 PTR .EQ $06
0008- 1090 COUNT .EQ $08
0024- 1100 CH .EQ $24
0025- 1110 CV .EQ $25
B3F3- 1120 BIT.MAPS .EQ $B3F3
C050- 1130 GRAPHICS .EQ $C050
C056- 1140 LORES .EQ $C056
C053- 1150 MIX .EQ $C053
F800- 1160 MON.PLOT .EQ $F800
F836- 1170 MON.CLRTOP .EQ $F836
F864- 1180 MON.SETCOL .EQ $F864
FC22- 1190 MON.VTAB .EQ $FC22
FC42- 1200 MON.CLEARE .EQ $FC42
FD8E- 1210 MON.CROUT .EQ $FD8E
FD8D- 1220 MON.COUT .EQ $FD8D
1230 *
1240 * SET POINTER TO BIT MAPS
1250 *
1260 GO
4000- A9 F3 1270 LDA #BIT.MAPS
4002- 85 06 1280 STA PTR
4004- A9 B3 1290 LDA #BIT.MAPS
4006- 85 07 1300 STA PTR+1
4008- A9 00 1310 LDA #0
400A- 8D 02 41 1320 STA TRACK.NUMBER
1330 *
1340 * COUNT.AND.MAKE TABLE
1350 * FIRST BYTE OF TRACK BIT MAP
1360 *
4016- 0A 1370 ASL COUNT 0 BITS
4017- 80 03 1440 BCS SET1 BR IF NOT 0
4019- EE 01 41 1450 INC TOTAL COUNT IT
401C- CA 1460 SET1 DEX BR 8 TIMES
401D- D0 F7 1470 BNE ONE
1480 *
401F- A2 08 1490 LDX #08 SECOND BYTE
4021- C8 1500 INY
4022- B1 06 1510 LDA (PTR),Y
1520 *
1530 * COUNT NUMBER OF 0 BITS IN
1540 * SECOND BYTE OF TRACK BIT MAP
1550 *
4024- 0A 1560 TWO ASL COUNT 0 BITS
4025- 80 03 1570 BCS SET2 BR IF NOT 0
4027- EE 01 41 1580 INC TOTAL COUNT IT
402A- CA 1590 SET2 DEX
402B- D0 F7 1600 BNE TWO 8 TIMES
```

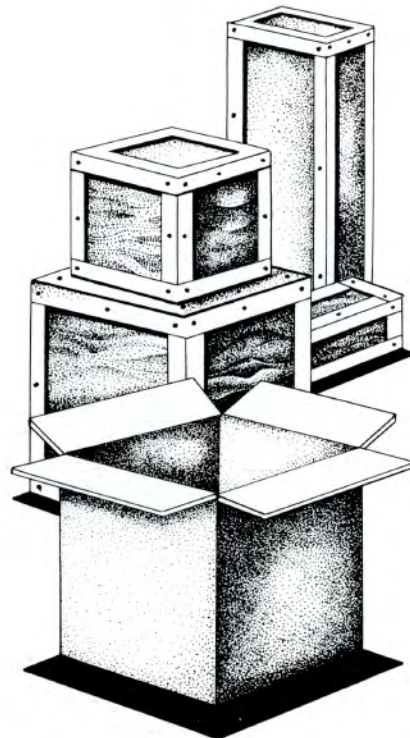
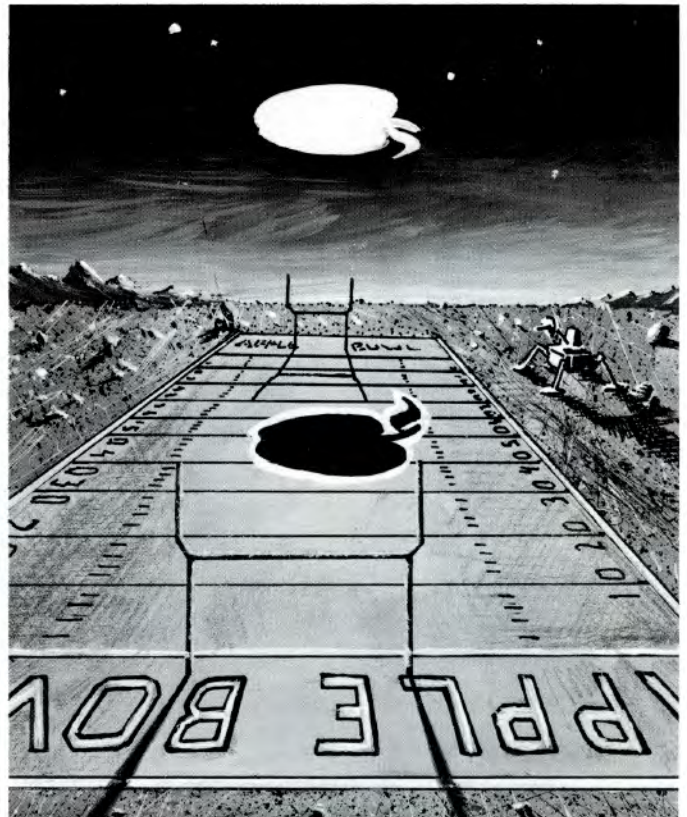
continued on next page

Disk Map (Cont.)

```

1610 *-----
1620 * STORE IN TABLE FOR THIS TRACK
1630 *-----
402D- AE 02 41 1640 LDX TRACK,NUMBER
4030- AD 01 41 1650 LDA TOTAL NO. OF 0 BITS
4033- 9D 03 41 1660 STA TABLE,X PUT IN TABLE
4036- EE 02 41 1670 INC TRACK,NUMBER
4039- AD 02 41 1680 LDA TRACK,NUMBER
403C- C9 23 1690 CMP #23 STOP AT 34
403E- F0 10 1700 REG DO,GRAPHICS
1710 *-----
1720 * ADVANCE TO NEXT TRACK BIT MAP
1730 *-----
4040- 18 1740 CLC
4041- A5 06 1750 LDA PTR ADD 4 TO PTR
4043- 69 04 1760 ADC #04
4045- 85 06 1770 STA PTR
4047- A5 07 1780 LDA PTR+1
4049- 69 00 1790 ADC #00
404B- 85 07 1800 STA PTR+1
404D- 4C 00 1810 JMP COUNT,AND,MAKE, TABLE
1820 *-----
1830 DO,GRAPHICS
4050- AD 50 C0 1840 LDA GRAPHICS
4053- AD 56 C0 1850 LDA LORES
4056- AD 53 C0 1860 LDA MIX
4059- 20 36 FB 1870 JSR MON,CLRTOP
405C- A9 02 1880 LDA #02 INIT HORIZONTAL
405E- 8D 00 41 1890 STA HORIZONTAL
4061- A9 00 41 1900 LDA #00 INIT VERTICAL
4063- 8D FF 40 1910 STA VERTICAL
4066- A9 02 1920 LDA #02 DARK COLOR
406B- 20 64 FB 1930 JSR MON,SETCOL
1940 *-----
1950 * DRAW VERTICAL LINE COLOR #02
1960 * TOP TO BOTTOM, WHOLE SCREEN
1970 *-----
1980 DRAW,DARK
406B- AD FF 40 1990 LDA VERTICAL
406E- AC 00 41 2000 LDY HORIZONTAL
4071- 20 00 FB 2010 JSR MON,PLOT
4074- 18 2020 CLC
4075- AD FF 40 2030 LDA VERTICAL
4078- 69 01 2040 ADC #01
407A- 8D FF 40 2050 STA VERTICAL
407D- C9 20 2060 CMP #20
407F- 90 EA 2070 BCC DRAW,DARK
2080 *-----
2090 * ADD 1 TO HORIZONTAL THRU 36
2100 * BECAUSE WE STARTED AT 2
2110 * GO BACK AND DO ANOTHER
2120 *-----
4081- A9 00 2130 LDA #00
4083- 8D FF 40 2140 STA VERTICAL
4086- 18 2150 CLC
4087- AD 00 41 2160 LDA HORIZONTAL
408A- 69 01 2170 ADC #01
408C- 8D 00 41 2180 STA HORIZONTAL
408F- C9 25 2190 CMP #25
4091- 90 D8 2200 BCC DRAW,DARK
2210 *-----
4093- 20 CF 40 2220 JSR DRAW,LIGHT
4096- A9 14 2230 LDA #14
4098- 85 25 2240 STA CV
409A- 20 22 FC 2250 JSR MON,VTAR
409D- 20 42 FC 2260 JSR MON,CLREOP
40A0- A9 02 2270 LDA #02
40A2- 85 24 2280 STA CH
40A4- A9 80 2290 LDA #80 LETTER O
40A6- 20 ED FD 2300 JSR MON,COUT
40A9- A9 13 2310 LDA #13
40AB- 85 24 2320 STA CH
40AD- A9 C4 2330 LDA #D+80 D-DIRECTORY
40AF- 20 ED FD 2340 JSR MON,COUT
40B2- 20 BE FD 2350 JSR MON,CROUT
40B5- A9 00 2360 LDA #00
40B7- 85 08 2370 STA COUNT
40B9- 85 09 2380 STA COUNT+1
2390 *-----
2400 * ADD UP ALL THE USED SECTORS
2410 * PUT TOTAL WHERE IT CAN BE USED
2420 * WHEN NEEDED
2430 *-----
40BB- A2 22 2440 LDX #22
40BD- 8D 03 41 2450 LOOP1 LDA TABLE,X
40C0- 18 2460 CLC
40C1- 65 08 2470 ADC COUNT
40C3- 85 08 2480 STA COUNT
40C5- A5 09 2490 LDA COUNT+1
40C7- 69 00 2500 ADC #00
40C9- 85 09 2510 STA COUNT+1
40CB- CA 2520 DEX
40CC- 10 EF 2530 BPL LOOP1
40CE- 60 2540 RTS
2550 *-----
2560 * PUT IN COLOR #0F
2570 *-----
2580 DRAW,LIGHT
40CF- A9 0F 2590 LDA #0F
40D1- 20 64 FB 2600 JSR MON,SETCOL
40D4- A2 00 2610 LDX #00
40D6- A0 02 2620 LDY #02
40D8- 8C 00 41 2630 STY HORIZONTAL
40DB- 8D 03 41 2640 BR9 LDA TABLE,X
40DE- C9 10 2650 CMP #10 16 BITS ALL 0
40E0- F0 14 2660 BEQ BR2 NO FREE SPACE
40E2- 0A 2670 ASL DOUBLE COUNT
40E3- 8D FF 40 2680 STA VERTICAL START POSITION
40E6- AC 00 41 2690 LDY HORIZONTAL
40E9- 20 00 FB 2700 BR10 JSR MON,PLOT
40EC- EE FF 40 2710 INC VERTICAL
40EE- AD FF 40 2720 LDA VERTICAL
40F2- C9 20 2730 CMP #20 DRAW TO BOTTOM
40F4- 90 F3 2740 BCC BR10
40F6- EE 00 41 2750 INC HORIZONTAL NEXT COLUMN
40F9- EB 2760 INX
40FA- E0 23 2770 CPX #23 DON'T DO 35
40FC- 90 D0 2780 BCC BR9
40FE- 60 2790 RTS
2800 *-----
2810 VERTICAL
40FF- 2820 .BS 1
2830 HORIZONTAL
4100- 2840 .BS 1
4101- 2850 TOTAL .BS 1
2860 TRACK,NUMBER
4102- 2870 .BS 1
4103- 2880 TABLE .BS 23
0126- 2890 Z.LENGTH .EQ *-GO

```



APPLE BOWL FOOTBALL

Apple Bowl Football

by Preston R. Black, MD
Richard I Morris Jr.
The Durham Software Group
16 Durham St.
Boston, MA 02115

Any true football fan is by nature a grandstand or Monday morning quarterback. How many times have you said to yourself after the hometown favorites had blown another one, "If I was the quarterback I would never have called a pass play in that situation. Everyone in the stadium was looking for it. I would have tried a screen pass or a draw!"? Well, here's your chance to put your words into action. Apple Bowl Football is an interactive strategy game which gives you the opportunity to test your play-calling ability against the Apple II. You have your choice of **three passing plays** and **four running plays**. A **full kicking game** including quick kicks is also available. There are **safeties, long runs, and long passing bombs**. Your quarterback can be sacked and you can fumble on any play. Don't get caught offsidies or clipping. If you are caught, you will have penalty yards marked off against you.

PLAYING APPLE BOWL FOOTBALL

Apple Bowl Football requires at least 32K of memory when DOS is used concurrently. If DOS is not used and if the REM statements and the instructions are removed, the game can probably be squeezed into 16K.

Operating the program is fairly easy. Most things are self-evident and almost everything else is explained within the program. At the start of your first game you are asked to initialize the random number generator (as explained above) and to set the video output speed. The Scoreboard leaves only a small text area at the bottom of the screen. Consequently, the **play-by-play narrative** can scroll off the screen before it is comprehended. By **slowing down the display**, time is allowed for the player to read the play-by-play narrative. For new players of Apple Bowl Football, a speed of **2 or 3** (equivalent to **SPEED=105** and **SPEED=155** respectively) is recommended. After the player becomes more familiar with the game, he may wish to use a speed of **4 or 5** (**SPEED=205** and **SPEED=255** respectively).

THE PLAYS

A listing of the plays available in Apple Bowl Football is given with the instructions and can be relisted at any time during the game by entering "PLAYS" when the Apple II prompts the player with "NEW PLAY:". A listing of the plays is also given in **Table 1** to facilitate the flow of the game.

TABLE 1

The current plays available in Football:

LPASS — LONG PASSING PLAY
SPASS — SHORT PASSING PLAY
SCRN — SCREEN PASS
MRUN — RUN UP THE MIDDLE
OTRUN — OFF TACKLE RUN
DRAW — DRAW PLAY
SWEEP — A SWEEP
PUNT — A PUNT
QKKIK — A QUICK KICK
FLDGL — A FIELD GOAL ATTEMPT
STAT — LIST OF CURRENT STATISTICS
TMOUT — TAKES A TIME-OUT

PLAYS — GIVES A LIST OF AVAILABLE PLAYS

The original version of this program was written by us in FORTRAN IV on a PDP-10 time-sharing system. When we purchased our Apple II, the program was translated into Applesoft. Many changes have been made in the program since the original version. Most of these have been to incorporate the special features of Applesoft Basic and the Apple II. The final product of eight years of various versions is Football. Over the years we have had many hours of enjoyment from this game. We hope that the readers of NIBBLE will enjoy this program as much as we have.

THE PASSING GAME

Every attempt has been made to make the Apple Bowl Football game realistic. As in an actual football game, the player is more likely to complete a screen pass than a short pass down field. And the completion rate for long passes is lower than for short passes. But if the defense reads your screen pass, you may lose yardage. Any of the pass plays can break for a long gainer. Pass too frequently, and the computer will recognize this tendency and make adjustment to its defense. Interceptions and quarterback sacks then become more likely.

THE RUNNING GAME

The running game is also realistic. Sweeps and draws are more wide open and can gain more yardage than running plays up the middle. But they are more likely to lose yardage than conservative plays such as runs off tackle or runs up the middle. It is also more difficult to gain yardage in an obvious running situation such as third or fourth down and short yardage, or whenever you are inside your opponent's ten yard line. Again, all of the running plays can be broken for long gainers.

THE KICKING GAME

The player has a complete kicking game. The plays include punts and as a surprise, you can also quick kick. Because they are unexpected, quick kicks are usually longer than punts and the returns are shorter. But quick kicks on a fourth down fool no one and are treated as punts.

The player also has a very accurate, strong-legged placekicker. Field goals can be attempted from anywhere on the field. But should you miss an attempt from outside the twenty yard line, your opponent will start from the original line of scrimmage. If the field goal is short, it will be returned by your opponent.

TIMING

Are the last two minutes of each half your favorite? Do not despair. Even though there are no commercials, a two minute warning (with a timeout) is issued each half. You have three time outs per half. Using them wisely in your two minute drill will allow the player to move into scoring position before the end of the half, or score the winning touchdown in the waning moments of the game. If the score is tied after four quarters of play, a sudden victory overtime period is automatically invoked for one more try.

Defensive plays are always called by the Apple II. Prior versions of the program have had a player-called defense feature. But this

feature slowed down the flow of the game so much, it was finally removed. The Apple II is impartial on defense selection. Regardless of who is on offense **the Apple II will try to anticipate plays based on patterns and field position.**

STATISTICS

Do you want to know how well your defensive and offensive teams are performing? How many plays have you run? What is your passing, running, and total yardage? What is your average gain per carry? What is your average yardage per pass? How well are your punter and kicker performing? How many turnovers have there been? How many penalty yards have been assessed against you? What is your opponent's total yardage? How many times has your defensive unit sacked your opponent's quarterback? Apple Bowl Football keeps full statistics (which are always current). These statistics can be displayed **at any time** during the game to answer all of the questions posed above. The statistics are automatically displayed at the end of each half.

THE SCOREBOARD

Perhaps the most innovative feature of Apple Bowl Football is the Scoreboard which is always displayed on the top half of the TV screen. Formatted like a stadium scoreboard, the display includes the time left in the quarter, the down and ball placement, the score, the current quarter, and possession data. The scoreboard is updated after every play and has all the information needed for decision making. The lower half of the screen displays the play-by-play narrative.

PROGRAMMING LOGIC

Mathematically, football is a game of probabilities. Each play has a statistical probability of success and another probability determines the **magnitude** of that success. They both depend upon a multitude of factors. Reviewing professional football annual statistics reveals many interesting facts. While football is usually thought of as primarily a passing game, **passing accounts for less than 40% of all football plays**. And the **average pass gains only 8 to 10 yards**. Furthermore, only 45 to 55% of passes are completed. Moreover, the **average running play gains only 2.5 to 3.5 yards**.

Statistics of this nature are used in Apple Bowl Football. Parameters are set based on these statistics, and random numbers are generated to decide the outcome of a particular play. In addition, special parameters are incorporated for those situations considered "special" in real football. Thus it is more difficult to complete a pass on third down and long yardage, or to gain yardage on the ground within your opponent's ten yard line. Yardage gained is also determined by parameters designed for each play and the generation of random numbers. For example, the parameters of running plays are designed so that the **average sweep will gain more than the average run up the middle**. Yet **you are less likely to lose yardage on the more conservative play**. Likewise a long pass will gain more yardage than a screen pass. But the long pass is more likely to be incomplete or intercepted. The length of punts and punt returns are based on similar algorithms.

continued on next page

Apple Bowl Football (Cont.)

Whether a penalty, a fumble, or bonus yardage occurs is determined on each play. Again, parameters are set and random numbers are generated to decide the outcome when the **FUMBLE**, **PENALTY**, or **BONUS** routines are called in the program. The success or failure of field goals is also determined **statistically**. The longer the field goal attempt, the more likely that it will be unsuccessful. But short field goals are not sure things either. They too can be missed. Even points after touchdowns will occasionally be missed.

Overall parameters have been set to simulate those observed in real football games. Every effort has been made to make Apple Bowl Football as realistic as possible.

RANDOM NUMBERS

Most of the coding in Apple Bowl Football is straightforward. Since the program makes extensive use of random numbers, we will briefly discuss the Applesoft random number generator. The implementation of the Scoreboard feature and the Scoreboard Updater will also be analyzed.

The pseudo-random number generator resident in Applesoft Basic is very interesting. Unlike the pseudo-random number generator found in Integer Basic, Applesoft's generates random numbers **between 0 and 1**. In Integer Basic, the command RND(X) will return a random integer between 1 and X. In Applesoft the argument is used to set the function mode. A **positive** argument (greater than 0) generates a new random number each time the function is called. An argument of **zero** returns the most recently generated random number. A **negative** argument sets a **sequence** of random numbers to be used. This sequence will be repeated each time the same negative argument is used. The random number generated is always positive and between 0 and 1.

Moreover, pseudo-random number generators like the one in Applesoft usually generate the same table for a given seed number. Since the seed cannot be set randomly — the first number is predictable — a way to randomize the sequence has to be found. For those readers with a clock card, the time can be used to randomize the seed. Those of us without clocks must find another way. Otherwise, whenever Apple Bowl Football is played, the first game would always be exactly the same as the first game played on a previous occasion. Therefore, the player is allowed to set the random number generator at

the beginning of the first game he plays. Yet the player has the option to **replay** games if he so desires by entering the same number to set the function at the start of the game. Remember the game must be repeated exactly the same, with the same plays, to get the same result.

In Applesoft the expression **X*RND(1)+1** will give a random number between 1 and X. We are only interested in the integer values for the random numbers in the game. Since we use random numbers so frequently, we defined a function, using the **DEF FN** of Applesoft, to give a random integer from 1 to X, when X is the argument of the function defined. This is implemented in **line 160**.

HOW THE SCOREBOARD WORKS

Implementation of the Scoreboard takes advantage of the text screen control that Applesoft affords the programmer. The video display of the Apple II in the TEXT mode has the capacity for 960 characters in a twenty-four line by forty column matrix. By **assigning a column and line value to a character**, the programmer can place a character anywhere on the screen. The **VTAB** and **HTAB** commands provide a method to do this.

The **VTAB** command moves the cursor to the **line** designated by its argument. **VTAB uses absolute moves**, relative only to the top and bottom of the screen; it ignores the text window. If VTAB moves the cursor below the text window, normal scrolling is active on all subsequent printing.

HTAB moves the cursor to the **position** designated by its argument on the current screen line. HTAB's moves are **relative to the left margin of the current screen line**. HTAB ignores the text screen, operating within the current line. HTAB can move the cursor to the left or to the right.

TAB is similar to HTAB with these important differences: **First**, TAB must be used in a PRINT statement. **Second**, TAB moves the cursor to the right of the cursor's current position. If the value of the argument for TAB is less than the current cursor position, the cursor will not move. **Third**, TAB has an interesting idiosyncrasy. Within a single PRINT statement the TAB does not affect characters previously printed on the current line. However, if the TAB is in another PRINT statement that references a line on the screen that has been previously printed on, the TAB becomes similar to printing spaces and all the text previously written on that line will be erased.

TEXT WINDOWS

Applesoft Basic also allows you to partition the text screen into "**windows**" of any dimen-

sions. A familiar example of such a text window is the four lines at the bottom of the low resolution graphics display for text. There are **four special POKE-commands to set the size of the "window" for the screen**. Within this window text will be shown and scrolled.

POKE 32,L will set the left margin of the text screen to the value specified by **L**. **L** can be in the range from **0 to 39** where **0** is the left margin. Note that the width of the window is not changed by this command; this means that the right margin is moved by the same amount as the left margin.

POKE 33,W sets the **width** (i.e. the number of characters per line) of the text screen to the value of **W**, which can be in the range from **1 through 40**. **W** cannot be zero; POKE 33,0 bombs Applesoft.

POKE 34,T sets the **top margin** of the text screen to the value specified by **T**. **T** can be in the range from **0 to 23**, where **0** is the top line of the screen.

POKE 35,B sets the **bottom margin** of the text screen to the value specified by **B**. **B** can be in the range from **0 to 23**, where **23** is the bottom line on the screen. Never set the bottom of the window (B) above the top of the window (T).

The programmer can use these commands to control output to the screen and customize output to suit his needs.

The SCOREBOARD routine takes advantage of some of these Applesoft features. After the screen has been cleared by the HOME command, the VTAB and HTAB commands are used to construct the scoreboard. Advantage is taken of HTAB's ability to use a variable as an argument to properly place the scores for each quarter (line 340). Once the scoreboard has been constructed, POKE 34,16 (line 420) sets the text window so that the scoreboard is not overwritten.

The SCOREBOARD UPDATER routine relies upon the fact that VTAB does not recognize the text window to make changes in the scoreboard. We can change those parts of the scoreboard that need changing without affecting other parts of the scoreboard. The VTAB 24 in line 600 returns us to the text screen, again preventing the scoreboard from being overwritten. The remainder of the scoreboard implementation and updating are easily understood from reviewing the program listing.

The interested reader can explore the remainder of the design of Apple Bowl Football by studying the program listing. The various routines are clearly delineated by REM statements and are easily found.



```

10 REM *****
20 REM * APPLE FOOTBALL 3.0 *
30 REM *
40 REM * BY PRESTON BLACK *
50 REM * AND RICHARD MORRIS *
60 REM *
70 REM * COPYRIGHT (C) 1982 *
80 REM * BY MICRO-SPARC INC *
90 REM * LINCOLN, MA. 01773 *
100 REM *****
110 DEF FN Z(X) = INT (X * 100 + .5) / INT (100.5)
120 DIM DS(9),SUB(2)
130 W1$ = "GAINS":W2$ = "LOSES"
140 DIM FIRST(2),ZPLAY(2),RUSH(2),PASS(2),PUNT(2),YPUNT(2),
    FLD(2),MADE(2),XPUNT(2),FUM(2),RECO(2),NENT(2),NPEN(2),PPEN(
    2),LOSS(2),COMP(2),YRUSH(2),RPASS(2),PLAYS(15),TEAMS(2),SOR(
    4),TMOUT(2),DFENS(10),YARD(2),PDN(2)
150 DIM KZ(4,3),YSAK(2)
160 DEF FN A(X) = INT (X * RND (1)) + 1: DEF FN B(X) =
    (X / 2) - (INT (X / 2))
170 FOR I = 1 TO 12: READ PLYS(I): NEXT I: DATA LPASS,SPA
    SS,SCRN,MRUN,DRAW,SWEEP,OTRUN,PUNT,QKIK,FLDGL,TMOUT,STAT
180 FOR I = 1 TO 3: FOR J = 1 TO 4: READ KZ(J,I): NEXT J: N
    EXT I: DATA 4,2,2,3,4,3,5,4,4,10,8,6

```

```

190 FOR I = 1 TO 2:RUSH(I) = 0:FUM(I) = 0:RECO(I) = 0:NENT(
    I) = 0:PASS(I) = 0:COMP(I) = 0:LOSS(I) = 0:YRUSH(I) = 0:RPAS
    S(I) = 0:FIRST(I) = 0:YPUNT(I) = 0:XPUNT(I) = 0:FLD(I) = 0:M
    ADE(I) = 0:NPEN(I) = 0:PPEN(I) = 0: NEXT
200 FOR I = 1 TO 2:YSAK(I) = 0: NEXT
210 GOTO 4770
220 REM
230 REM ***SCOREBOARD***
240 REM
250 SPEED= 255: TEXT : HOME : HTAB 15: PRINT "SCOREBOARD"
260 NMN = INT (ITIME / 60):NSEC = INT (ITIME - (NMN * 60
    ))
270 VTAB 3: HTAB 17
280 IF ITIME > 0 THEN 300
290 PRINT " 0:00": GOTO 330
300 IF NMN < 10 THEN PRINT " ";
310 PRINT NMN;" ":" "; IF NSEC < 10 THEN PRINT "0 ";
320 PRINT NSEC
330 VTAB 6: FOR I = 1 TO 4:DS = 8 + 4 * I: IF I = QTR THEN
    FLASH
340 HTAB DS: PRINT I;: NORMAL : NEXT I: PRINT TAB( 28);"TOT
    AL"
350 FLASH : VTAB 8: IF AN = 1 THEN PRINT "*****":
360 NORMAL : PRINT TAB( 2);TEAMS(1);: FOR I = 1 TO 4:DS =
    8 + 4 * I: PRINT TAB( DS);SOR(I,1);: NEXT I: PRINT TAB( 29)
    :L1
370 FLASH : IF AN = 2 THEN PRINT "*****";

```


Apple Bowl Football (Cont.)

```

380 NORMAL : PRINT TAB( 2);TEAMS(2);: FOR I = 1 TO 4:DS =
      8 + 4 * I: PRINT TAB( DS);SCR(1,2);: NEXT : PRINT TAB( 29)
      L2
390 VTAB 11: PRINT "BALL ON THE ";TEAMS(YARD(1));" ";YARD(2)
      J;" YARD LINE"
400 VTAB 13: PRINT "DOWN=";DOWN; TAB( 20);YSTGO;" YARDS TO
      GO"
410 VTAB 15: PRINT "-----"
      -"
420 POKE 34,16: VTAB 24: SPEED= ISPD: RETURN
430 REM
440 REM ***SCOREBOARD UPDATER***
450 REM
460 NMN = INT (ITIME / 60):NSEC = INT (ITIME - (NMN * 60
      ))
470 SPEED= 255: IF ITIME > 0 THEN 490
480 VTAB 3: HTAB 18: PRINT "0:00": GOTO 600
490 VTAB 3: HTAB 17: IF NMN < 10 THEN PRINT " ";
500 PRINT NMN;"": IF NSEC < 10 THEN PRINT "0";
510 PRINT NSEC: VTAB 24: GOTO 600
520 SPEED= 255: VTAB 6:DS = 8 + 4 * (QTR - 1): PRINT TAB(
      DS);(QTR - 1);DS = 8 + 4 * QTR: FLASH : PRINT TAB( DS);QTR
      : NORMAL : VTAB 24: GOTO 600
530 SPEED= 255: ON AN GOTO 540,550
540 VTAB 8: FLASH : PRINT "": NORMAL : VTAB 9: PRINT " ":
      GOTO 600
550 VTAB 8: PRINT " ": VTAB 9: FLASH : PRINT "": NORMAL :
      GOTO 600
560 DS = 8 + 4 * QTR: VTAB 8: HTAB DS: PRINT SCR(QTR,1);: HT
      AB 29: PRINT L1: VTAB 9: HTAB DS: PRINT SCR(QTR,2);: HTAB 29
      : PRINT L2: GOTO 600
570 SPEED= 255: VTAB 11: HTAB 13: PRINT TAB( 27): VTAB 11:
      HTAB 13: PRINT TEAMS(YARD(1));" ";YARD(2);" YARD LINE "
580 VTAB 13: HTAB 6: PRINT DOWN;: HTAB 20: IF YSTGO < 10 TH
      EN PRINT " ";
590 PRINT TAB( 15): VTAB 13: HTAB 20: PRINT YSTGO;" YARDS
      TO GO "
600 SPEED= ISPD: VTAB 24: RETURN
610 REM
620 REM ***BONUS***
630 REM
640 XNAN = RND (2):DS(1) = 10: FOR I = 2 TO 9:DS(I) = DS(I
      - 1) + 5: NEXT
650 IF XNAN * 20 > = 1 THEN 750
660 NB = 1: IF XNAN * 30 < = 1 THEN NB = 2
670 IF XNAN * 40 < = 1 THEN NB = 3
680 IF XNAN * 50 < = 1 THEN NB = 4
690 IF XNAN * 60 < = 1 THEN NB = 5
700 IF XNAN * 70 < = 1 THEN NB = 6
710 IF XNAN * 80 < = 1 THEN NB = 7
720 IF XNAN * 90 < = 1 THEN NB = 8
730 IF XNAN * 100 < = 1 THEN NB = 9
740 BNUS = DS(NB):XNAN = 0: GOTO 760
750 BNUS = 0:XNAN = 0
760 RETURN
770 REM
780 REM ***FUMBLE***
790 REM
800 CHANCE = FN A(100): IF CHANCE < = 1 THEN 820
810 FUMBL = 0:RECOV = 0: GOTO 890
820 FUMBL = 1:FUM(AN) = FUM(AN) + 1: PRINT : PRINT TEAMS(AN)
      : " FUMBLES!!!"
830 RECOV = FN A(2): IF RECOV = AN THEN 870
840 SET = AN:AN = OFF:OFF = SET:NPLY = NPLY + 1:ITIME = ITIM
      E - 5:DOWN = 1:YSTGO = 10
850 IF YARD(1) < > AN AND YARD(2) < 10 THEN YSTGO = YARD(2
      )
860 GOTO 880
870 FUMBL = 0:RECOV(AN) = RECOV(AN) + 1:RECOV = 0
880 PRINT : PRINT TEAMS(AN);" RECOVERS!!!": PRINT : PRINT "
      BALL ON THE ";TEAMS(YARD(1));" ";YARD(2);" YARD LINE"
890 RETURN
900 REM
910 REM ***PENALTY***
920 REM
930 KFIRST = 0:CHANCE = FN A(100): IF CHANCE < = 5 THEN 95
      0
940 PENALT = 0: RETURN
950 YFIX = FN A(100):YDS = 5: IF YFIX > 66 THEN YDS = 15
960 XTM = FN A(2)
970 IF XTM = OFF THEN 1070
980 IF YARD(1) = OFF THEN 1030
990 IF YARD(2) > 20 THEN 1020
1000 XDS = INT (YARD(2) / 2): IF YDS = 5 AND XDS < 5 THEN Y
      DS = XDS
1010 IF YDS = 15 THEN YDS = XDS
1020 YARD(2) = YARD(2) - YDS: GOTO 1160
1030 IF YARD(2) > 35 THEN 1050
1040 YARD(2) = YARD(2) + YDS: GOTO 1160
1050 IF YARD(2) < = 45 AND YDS = 5 THEN 1040
1060 YARD(1) = AN:YARD(2) = 50 - (YARD(2) + YDS - 50): GOTO
      1160
1070 IF YARD(1) = AN THEN 1120
1080 IF YARD(2) > 20 THEN 1110
1090 XDS = INT (YARD(2) / 2): IF YDS = 5 AND XDS < 5 THEN Y
      DS = XDS
1100 IF YDS = 15 THEN YDS = XDS
1110 YARD(2) = YARD(2) - YDS: GOTO 1160
1120 IF YARD(2) > 35 THEN 1140
1130 YARD(2) = YARD(2) + YDS: GOTO 1160
1140 IF YDS = 5 AND YARD(2) < = 45 THEN 1040
1150 YARD(1) = OFF:YARD(2) = 50 - (YARD(2) + YDS - 50)
1160 NPEN(XTM) = NPEN(XTM) + 1:PPEN(XTM) = PPEN(XTM) + YDS:
      IF KICK = 1 THEN 1230
1170 IF XTM = AN THEN 1220
1180 YSTGO = YSTGO - YDS:
1190 IF YSTGO > 0 THEN 1230
1200 DOWN = 1:YSTGO = 10: IF YARD(1) = OFF AND YARD(2) < 10
      THEN YSTGO = YARD(2)
1210 KFIRST = 1:FIRST(AN) = FIRST(AN) + 1: GOTO 1230
1220 YSTGO = YSTGO + YDS
1230 IF YDS < > 5 AND YDS < > 15 THEN 1250
1240 PRINT : PRINT "<<PENALTY ON ";TEAMS(XTM);">> " ;YDS;"
      YARDS": PRINT "BALL IS PLACED ON THE ";TEAMS(YARD(1));" ";YA
      RD(2): GOTO 1260

```

```

1250 PRINT : PRINT "<<PENALTY ON ";TEAMS(XTM);">>": PRINT "
      HALF THE DISTANCE TO THE GOAL LINE!": PRINT "BALL IS PLACE O
      N THE ";TEAMS(YARD(1));" ";YARD(2)
1260 PRINT : IF KFIRST = 1 THEN PRINT "FIRST DOWN FOR ";TE
      AMS(AN)
1270 IF ZN = 0 THEN 1290
1280 DOWN = 1:YSTGO = 10:ZN = 0
1290 GOSUB 570: POP : GOTO 1580
1300 FOR I = 1 TO 9: PRINT " ";
1310 FOR J = 1 TO 10:X = PEEK ( - 16336): NEXT J
1320 NEXT
1330 RETURN
1340 IF OVT = 1 THEN 1400
1350 QTR = 1:TMOUT(1) = 3:TMOUT(2) = 3:INCOMP = 0:WARN = 0:T
      IMSET = 0:ITIME = 900:OVT = 0
1360 SAFETY = 0: FOR I = 1 TO 4:SCR(1,1) = 0:SCR(1,2) = 0: N
      EXT
1370 REM
1380 REM ***THE KICK OFF***
1390 REM
1400 GOSUB 250: VTAB 24
1410 YARD(1) = OFF:YARD(2) = 40:THIS = 99:ITIME = ITIME - 5:
      HUDDL = 0:KICK = 1
1420 GOSUB 530: GOSUB 570: VTAB 24
1430 PRINT : PRINT TEAMS(OFF);" KICKS OFF AND"
1440 YDL = 50 - (35 + FN A(18)): IF YDL < = 0 THEN 1540
1450 IZAL = 20
1460 IF YDL < = 10 THEN IZAL = IZAL + 5
1470 IF YDL < = 5 THEN IZAL = IZAL + 5
1480 GOSUB 640:RTURN = 5 + FN A(IZAL) + (BNUS * 2):ITIME =
      ITIME - 3 - (RTURN / 10): IF RTURN > = 100 - YDL THEN 4020
1490 IF RTURN + YDL > 50 THEN 1510
1500 YARD(1) = AN:YARD(2) = RTURN + YDL:DOWN = 1:YSTGO = 10:
      GOTO 1520
1510 YARD(1) = OFF:YARD(2) = 50 - (RTURN + YDL - 50):DOWN =
      1:YSTGO = 10: IF YARD(2) < 10 THEN YSTGO = YARD(2)
1520 PRINT : PRINT TEAMS(AN);" RECEIVES THE KICK": PRINT "O
      N THEIR ";YDL;" YARD LINE": PRINT "AND RUNS IT BACK ";RTURN;
      " YARDS": PRINT "TO THE ";TEAMS(YARD(1));" ";YARD(2)
1530 GOSUB 800: GOSUB 930: GOSUB 530: GOSUB 560: GOSUB 570:
      GOTO 1580
1540 YARD(1) = AN:YARD(2) = 20:DOWN = 1:YSTGO = 10: PRINT :
      PRINT "THE KICK GOES INTO THE END ZONE": PRINT : PRINT "AND
      TEAMS(AN);" WILL START": PRINT "FROM THEIR OWN 20 YARD LINE
      ": GOSUB 530: GOSUB 570
1550 REM
1560 REM ***PLAY ANALYSER***
1570 REM
1580 SPEED= ISPD: POKE 34,16: VTAB 24:KICK = 0: IF DOWN = 5
      THEN 3690
1590 GOSUB 460: VTAB 24
1600 ZN = 0
1610 IF AN = 2 THEN 1660
1620 IF DOWN < > 4 THEN 1660
1630 IF ITIME < 60 AND QTR = 4 AND L1 < L2 THEN 1660
1640 IF YARD(1) = OFF AND YARD(2) < = 40 THEN 3740
1650 GOTO 3380
1660 NPLY = NPLY + 1: IF FN B(QTR) = 0 AND ITIME < = 120 A
      ND WARN = 0 THEN 1680
1670 GOTO 1690
1680 PRINT : PRINT ">>> TWO MINUTE WARNING <<<"
1690 IF FN B(QTR) = 0 AND ITIME < = 120 THEN WARN = 1
1700 NET = (SCR(1,1) + SCR(2,1) + SCR(3,1) + SCR(4,1)) - (SC
      R(1,2) + SCR(2,2) + SCR(3,2) + SCR(4,2))
1710 IF FN B(QTR) = 0 AND ITIME < = 120 AND TIMSET = 0 AN
      D NET < = 0 AND AN = 1 AND HUDDL > 0 THEN 2000
1720 INCOMP = 0:TIMSET = 0: POKE 34,16: IF ITIME < = 0 THEN
      4180
1730 IF AN = 2 THEN 1910
1740 REM
1750 REM ***COMPUTER ON OFFENSE***
1760 REM
1770 TYPE = FN A(100): IF TYPE < = 25 THEN THIS = FN A(3)
1780 IF TYPE > 25 THEN THIS = FN A(4) + 3
1790 IF DOWN = 3 AND YSTGO < = 3 THEN THIS = FN A(4) + 3
1800 IF DOWN = 3 AND YSTGO > 4 THEN THIS = FN A(2) + 1
1810 IF THIS = 3 THEN THIS = FN A(2) + 1
1820 IF YARD(1) = OFF AND YARD(2) < 15 AND THIS = 1 THEN TH
      IS = 2
1830 IF L1 < L2 AND ITIME < = 120 AND FN B(QTR) = 0 THEN
      THIS = 2
1840 IF QTR = 2 AND ITIME < 15 AND YARD(1) = OFF THEN 3740
1850 IF QTR = 4 AND L1 + 3 = > L2 AND ITIME < 25 AND YARD(
      1) = OFF AND YARD(2) < = 40 THEN 3740
1860 IF FN B(QTR) = 0 AND ITIME < 120 AND YARD(1) = OFF AN
      D YARD(2) < 5 THEN THIS = 4
1870 PRINT : PRINT TEAMS(1);" CHOOSES ";PLAY$(THIS): GOTO 2
      060
1880 REM
1890 REM ***PLAYER ON OFFENSE***
1900 REM
1910 PRINT : INPUT "NEW PLAY: ";HPL$: IF HPL$ < > "PLAYS"
      THEN 1930
1920 TEXT : HOME : SPEED= 255: GOTO 4780
1930 FOR I = 1 TO 15: IF MID$(HPL$,1,2) = MID$(PLAY$(I)
      ,1,2) THEN 1960
1940 NEXT
1950 PRINT : PRINT "ILLEGAL PLAY, TRY AGAIN!": GOTO 1580
1960 THIS = 1: IF THIS = 10 THEN 3740
1970 IF THIS = 8 OR THIS = 9 THEN 3390
1980 IF THIS < 11 THEN 2060
1990 IF THIS > 11 THEN 2040
2000 TIMSET = 1:TMOUT(AN) = TMOUT(AN) - 1: IF TMOUT(AN) >
      = 0 THEN 2030
2010 PRINT : IF AN = 2 THEN PRINT "NO MORE TIME-OUTS LEFT"
2020 GOTO 1580
2030 HUDDL = 0: PRINT : PRINT "TIME-OUT, ";TMOUT(AN);" LEFT
      IN THE HALF": GOTO 1580
2040 THIS = 12:TIMSET = 1
2050 TEXT : HOME : SPEED= 255: GOTO 4380
2060 ITIME = ITIME - HUDDL
2070 IF THIS > 3 THEN DFS = FN A(4) + 3
2080 IF THIS < 4 THEN DFS = FN A(3)
2090 IF THIS < 4 THEN 2460
2100 SUB(AN) = 0

```

continued on next page

Apple Bowl Football (Cont.)

```

2110 SUCC = 1 + ( FN A(2) - 1): IF THIS = DFS THEN SUCC = -
2120 REM
2130 REM ***RUN PLAY***
2140 REM
2150 RUSH(AN) = RUSH(AN) + 1:ITIME = (ITIME - 8) - FN A(10)
:HUDDL = 20: GOSUB 800: IF FUMBL = 1 THEN 1580
2160 IF DOWN > 2 AND SUCC = 2 THEN SUCC = - 1
2170 GOSUB 930: GOSUB 640
2180 IF YARD(2) < 10 AND YARD(1) = OFF THEN SUCC = 0:BNUS =
0
2190 IF DOWN > = 3 THEN BNUS = 0
2200 KB = KZ(THIS - 3, FN A(3))
2210 GAIN = FN A(KB) + SUCC + BNUS:YRUSH(AN) = YRUSH(AN) +
GAIN: IF YARD(1) = AN THEN 2330
2220 YSTGO = YSTGO - GAIN: IF YARD(2) = GAIN > 50 THEN 2240
2230 YARD(2) = YARD(2) - GAIN: GOTO 2250
2240 YARD(1) = AN:YARD(2) = (50 - (YARD(2) - GAIN)) + 50
2250 DOWN = DOWN + 1: IF YARD(2) > 0 THEN 2270
2260 YRUSH(AN) = YRUSH(AN) + YARD(2): GOTO 4030
2270 W$ = W$: IF GAIN < 0 THEN W$ = W$
2280 IF GAIN < 0 THEN GAIN = ABS(GAIN)
2290 PRINT : PRINT "THE PLAY "W$;" "YARD(2) YARDS": PRINT
"TO THE "YARD(1):" "YARD(2)
2300 IF YSTGO < = 0 THEN 3660
2310 GOSUB 570
2320 GOTO 1580
2330 DOWN = DOWN + 1:YSTGO = YSTGO - GAIN: IF YARD(2) + GAIN
< = 50 THEN 2370
2340 YARD(1) = OFF:YARD(2) = 50 - (GAIN + YARD(2) - 50): IF
YARD(2) > 0 THEN 2360
2350 YRUSH(AN) = YRUSH(AN) + YARD(2): GOTO 4030
2360 GOTO 2380
2370 YARD(2) = YARD(2) + GAIN: IF YARD(2) < = 0 THEN 4030
2380 W$ = W$: IF GAIN < 0 THEN W$ = W$
2390 IF GAIN < 0 THEN GAIN = ABS(GAIN)
2400 PRINT : PRINT "THE PLAY "W$;" "YARD(2) YARDS": PRINT
"TO THE "YARD(1):" "YARD(2): IF YSTGO < = 0 THEN 3
660
2410 GOSUB 570
2420 GOTO 1580
2430 REM
2440 REM ***PASS PLAY***
2450 REM
2460 PASS = FN A(100) - SUB(AN):SUB(AN) = SUB(AN) + 1
2470 PZ = 0
2480 IF DOWN > 2 AND THIS = 1 THEN PZ = PZ + 2
2490 IF DOWN = 4 THEN PZ = PZ + 5
2500 PASS(AN) = PASS(AN) + 1: IF PASS > 5 THEN 2640
2510 GOSUB 930
2520 REM
2530 REM ***PASSER THROWN FOR A LOSS***
2540 REM
2550 ITIME = ITIME - 8 - FN A(9):HUDDL = 20:LOSS = - 3 -
FN A(8):RPASS(AN) = RPASS(AN) + LOSS:DOWN = DOWN + 1:YSTGO =
YSTGO - LOSS
2560 VSAK(OFF) = VSAK(OFF) + 1
2570 IF YARD(1) = OFF THEN 2590
2580 YARD(2) = YARD(2) + LOSS: GOTO 2620
2590 IF YARD(2) = LOSS > 50 THEN 2610
2600 YARD(2) = YARD(2) - LOSS: GOTO 2620
2610 YARD(1) = AN:YARD(2) = 50 - (YARD(2) - LOSS - 50)
2620 IF YARD(2) < = 0 THEN 4140
2630 ALOS = ABS(LOSS): PRINT : PRINT TEAMS(AN):" PASSER IS
THROWN FOR A " : PRINT ALOS:" YARD LOSS TO THE "YARD(1):"
YARD(2): GOSUB 640: GOSUB 570: GOTO 1580
2640 IF THIS = 1 THEN TSS = 18 + FN A(25)
2650 IF THIS = 2 THEN TSS = FN A(7) + 3
2660 IF THIS = 3 THEN TSS = - 3
2670 IF PASS > 9 + PZ THEN 3000
2680 REM
2690 REM ***INTERCEPTED***
2700 REM
2710 ZN = 1
2720 ITIME = ITIME - 7 - FN A(10):NENT(OFF) = NENT(OFF) + 1
:HUDDL = 20:SET = AN:AN = OFF:OFF = SET: GOSUB 640:RNBK = 5
+ FN A(30) + BNUS:TXIS = THIS:THIS = 0
2730 IF YARD(1) = OFF THEN 2790
2740 IF TXIS = 3 AND YARD(2) > = 47 THEN 2780
2750 YARD(2) = YARD(2) - TSS: IF YARD(2) > 0 THEN 2820
2760 PRINT : PRINT TEAMS(OFF):" PASS INTERCEPTED BY": PRINT
TEAMS(AN):" IN THE END ZONE!": PRINT "AND THEY WILL START F
ROM THEIR OWN 20."
2770 YARD(1) = AN:YARD(2) = 20:DOWN = 1:YSTGO = 10: GOSUB 53
0: GOSUB 570: GOTO 1580
2780 YARD(2) = 50 - (YARD(2) - TSS - 50):YARD(1) = OFF: GOTO
2820
2790 IF YARD(2) + TSS > 50 THEN 2810
2800 YARD(2) = YARD(2) + TSS: GOTO 2820
2810 YARD(1) = AN:YARD(2) = 50 - (YARD(2) + TSS - 50)
2820 PRINT : PRINT TEAMS(OFF):" PASS INTERCEPTED BY": PRINT
TEAMS(AN):" AT THE "YARD(1):" "YARD(2)
2830 IF YARD(1) = AN THEN 2850
2840 YARD(2) = YARD(2) - RNBK: GOTO 2880
2850 IF YARD(2) + RNBK > 50 THEN 2870
2860 YARD(2) = YARD(2) + RNBK: GOTO 2880
2870 YARD(1) = OFF:YARD(2) = 50 - (YARD(2) + RNBK - 50)
2880 IF YARD(2) > 0 THEN PRINT "AND RUN BACK "YARD(2) YARD
S": PRINT "TO THE "YARD(1):" "YARD(2)
2890 GOTO 2930
2900 GOSUB 800: IF YARD(2) > 0 AND FUMBL < > 1 THEN 2920
2910 GOSUB 530: GOSUB 570: GOTO 1580
2920 IF YARD(2) > 0 THEN GOSUB 930
2930 IF YARD(2) < = 0 AND YARD(1) = OFF THEN PRINT "AND R
UN BACK FOR A"
2940 IF YARD(2) < = 0 AND YARD(1) = OFF THEN 4030
2950 FIRST(AN) = FIRST(AN) - 1: IF YARD(2) > 0 THEN 3660
2960 YARD(1) = AN:YARD(2) = 20: PRINT "AND THEY START AT THE
IR 20": GOTO 3660

```

```

2970 REM
2980 REM ***DETERMINES IF PASS COMPLETE***
2990 REM
3000 IF THIS > 1 THEN 3040
3010 COMPL = 67 + PZ
3020 IF DFS = 1 THEN COMPL = 79 + PZ
3030 GOTO 3080
3040 IF THIS > 2 THEN 3070
3050 COMPL = 37: IF DFS = 2 THEN COMPL = 59
3060 GOTO 3080
3070 COMPL = 27: IF DFS = 3 THEN COMPL = 54
3080 IF PASS > COMPL THEN 3160
3090 REM
3100 REM ***INCOMPLETE***
3110 REM
3120 INCOMP = 1:ITIME = ITIME - 3 - FN A(7):HUDDL = 0:DOWN
= DOWN + 1: PRINT : PRINT TEAMS(AN):" PASS INCOMPLETE": GOSU
B 570: GOTO 1580
3130 REM
3140 REM ***COMPLETE***
3150 REM
3160 IF THIS = 1 THEN GAIN = TSS + FN A(20)
3170 IF THIS = 2 THEN GAIN = TSS + FN A(8)
3180 IF THIS = 3 THEN GAIN = TSS + FN A(15)
3190 GOSUB 640:BNUS = BNUS * 2 - 10: IF BNUS > 0 THEN GAIN =
GAIN + BNUS
3200 COMP(AN) = COMP(AN) + 1:ITIME = (ITIME - 7) - FN A(10)
:HUDDL = 20:RPASS(AN) = RPASS(AN) + GAIN
3210 IF YARD(1) = OFF THEN 3260
3220 IF YARD(2) + GAIN > 50 THEN 3240
3230 YARD(2) = YARD(2) + GAIN: GOTO 3280
3240 YARD(1) = OFF:YARD(2) = 50 - (YARD(2) + GAIN - 50): IF
YARD(2) > 0 THEN 3280
3250 PRINT : PRINT TEAMS(AN):" PASS COMPLETE!": GOTO 4030
3260 YARD(2) = YARD(2) - GAIN: IF YARD(2) > 0 THEN 3280
3270 GOTO 3290
3280 W$ = W$: IF GAIN < 0 THEN W$ = W$
3290 IF GAIN < 0 THEN GAIN = ABS(GAIN)
3300 PRINT : PRINT TEAMS(AN):" PASS COMPLETE!": PRINT "PLA
Y "W$;" "YARD(2) YARDS": PRINT "TO THE "YARD(1):" "YARD(2)
3310 IF YARD(2) < = 0 THEN 4030
3320 IF W$ = W$ THEN GAIN = - GAIN
3330 YSTGO = YSTGO - GAIN: IF YSTGO < = 0 THEN 3660
3340 DOWN = DOWN + 1: GOSUB 640: GOSUB 570: GOTO 1580
3350 REM
3360 REM ***PUNT OR QUICK KICK***
3370 REM
3380 THIS = 8
3390 GOSUB 930:LNTH = FN A(25) + 28 + SAFETY: IF DOWN = 4
AND THIS = 9 THEN THIS = 8
3400 IF SAFETY > 0 THEN 3420
3410 PUNT(AN) = PUNT(AN) + 1:YPUNT(AN) = YPUNT(AN) + LNTH
3420 SAFETY = 0: IF THIS = 9 THEN LNTH = LNTH + 2 + FN A(
4)
3430 HUDDL = 0: IF LNTH + YARD(2) > 50 AND YARD(1) = AN THE
N 3490
3440 IF YARD(1) = AN THEN 3480
3450 YARD(2) = YARD(2) - LNTH: IF YARD(2) < = 0 THEN 3470
3460 GOTO 3500
3470 SET = AN:AN = OFF:OFF = SET:ITIME = ITIME - FN A(5): G
OTO 1540
3480 YARD(2) = YARD(2) + LNTH: GOTO 3500
3490 YARD(1) = OFF:YARD(2) = 50 - (YARD(2) + LNTH - 50): IF
YARD(2) < = 0 THEN 3470
3500 PRINT :SET = AN:AN = OFF:OFF = SET: IF THIS = 9 THEN
PRINT TEAMS(OFF):" QUICK KICKS "YARD(2) YARDS": PRINT "TO T
HE "YARD(1):" "YARD(2)
3510 PRINT : IF THIS = 8 THEN PRINT TEAMS(OFF):" PUNTS "YL
NTH" YARDS": PRINT "TO THE "YARD(1):" "YARD(2)
3520 GOSUB 640:RTURN = FN A(12) + BNUS: IF THIS = 9 THEN R
TURN = RTURN - 2 - FN A(3)
3530 IF RTURN < 0 THEN RTURN = 0
3540 ITIME = (ITIME - 7) - FN A(10): IF RTURN + YARD(2) > 5
0 AND YARD(1) = AN THEN 3610
3550 IF YARD(1) = AN THEN YARD(2) = YARD(2) + RTURN
3560 IF YARD(1) = OFF THEN YARD(2) = YARD(2) - RTURN
3570 PRINT : IF YARD(2) < = 0 THEN PRINT TEAMS(AN):" RETU
RNS IT FOR A "
3580 PRINT : IF YARD(2) < = 0 THEN 4030
3590 PRINT : PRINT TEAMS(AN):" RETURNS IT "YARD(2) YARDS":
PRINT "TO THE "YARD(1):" "YARD(2)
3600 GOSUB 640:FIRST(AN) = FIRST(AN) - 1: GOTO 3660
3610 YARD(1) = OFF:YARD(2) = 50 - (YARD(2) + RTURN - 50): IF
YARD(2) < = 0 THEN 3570
3620 GOTO 3590
3630 REM
3640 REM ***FIRST DOWN***
3650 REM
3660 DOWN = 1:FIRST(AN) = FIRST(AN) + 1:YSTGO = 10
3670 IF YARD(1) = OFF AND YARD(2) < 10 THEN YSTGO = YARD(2)
3680 PRINT : PRINT "FIRST DOWN ON THE "YARD(1):" "YARD(2)
YARD(2): GOSUB 530: GOSUB 570: GOSUB 580: GOTO 1580
3690 PRINT : PRINT "BALL TURNS OVER TO "YARD(1):" PRINT
"ON DOWNS AT THE "YARD(1):" "YARD(2)
3700 HUDDL = 0:SET = AN:AN = OFF:OFF = SET: GOTO 3660
3710 REM
3720 REM ***FIELD GOAL ATTEMPT***
3730 REM
3740 FLD(AN) = FLD(AN) + 1
3750 IF YARD(2) > 0 THEN PROB = 90
3760 IF YARD(2) > 20 THEN PROB = 80
3770 IF YARD(2) > 30 THEN PROB = 45
3780 IF YARD(2) > 40 THEN PROB = 25
3790 IF YARD(1) = AN AND YARD(2) > 40 THEN PROB = 10
3800 IF YARD(1) = AN AND YARD(2) < 40 THEN PROB = 2
3810 IF FN A(100) < PROB THEN 3920
3820 PRINT : PRINT "FIELD GOAL ATTEMPT FAILS,"
3830 SET = AN:AN = OFF:OFF = SET: IF YARD(1) = OFF THEN 3880
3840 IF YARD(2) < 20 THEN 3860
3850 PRINT : PRINT TEAMS(AN):" START FROM THE "YARD(1):"
YARD(2):YARD(1) = AN:DOWN = 1:YSTGO = 10:HUDDL = 0:
FOR I = 1 TO 500: NEXT : GOSUB 530: GOSUB 570: GOTO 1580
3860 PRINT : PRINT TEAMS(AN):" START FROM THEIR OWN 20"
3870 YARD(1) = AN:YARD(2) = 20:DOWN = 1:YSTGO = 10:HUDDL = 0
: GOSUB 530: GOSUB 570: GOTO 1580

```



```

3880 LENGTH = 39 + FN A(10):HUDDL = 0: PRINT : PRINT "THE KICK IS SHORT!"
3890 IF LENGTH + YARD(2) > 50 THEN 3910
3900 YARD(2) = YARD(2) + LENGTH: GOTO 3520
3910 YARD(1) = AN:YARD(2) = 50 - (YARD(2) + LENGTH - 50): GOTO 3520
3920 SPEED= 255:MADE(AN) = MADE(AN) + 1: FLASH: PRINT : PRINT "**** FIELD GOAL **** FOR ";TEAMS(AN): NORMAL : SPEED= ISPD
3930 SCR(QTR,AN) = SCR(QTR,AN) + 3
3940 L1 = 0:L2 = 0: FOR I = 1 TO QTR:L1 = L1 + SCR(I,1):L2 = L2 + SCR(I,2): NEXT I: GOSUB 560
3950 SET = AN:AN = OFF:OFF = SET
3960 IF OVT = 1 THEN 4180
3970 IF ITIME < 0 AND FN B(QTR) = 0 THEN 4180
3980 GOTO 1410
3990 REM
4000 REM ***TOUCHDOWN***
4010 REM
4020 PRINT TEAMS(AN); " RECEIVES THE KICK": PRINT "ON THEIR ";YDL;" YARD LINE": PRINT "AND RUNS IT BACK FOR A"
4030 SPEED= 255: FLASH: PRINT : PRINT "***** TOUCHDOWN ***** FOR ";TEAMS(AN): NORMAL : SPEED= ISPD
4040 IF THIS > = 1 AND THIS < 4 THEN RPASS(AN) = RPASS(AN) + YARD(2)
4050 IF THIS > = 4 AND THIS < 7 THEN YRUSH(AN) = YRUSH(AN) + YARD(2)
4060 IF THIS > 0 THEN FIRST(AN) = FIRST(AN) + 1
4070 SCR(QTR,AN) = SCR(QTR,AN) + 6:HUDDL = 0: IF FN A(100) < 3 THEN 4090
4080 SCR(QTR,AN) = SCR(QTR,AN) + 1: PRINT : PRINT "POINT AFTER GOOD!": GOTO 4100
4090 PRINT : PRINT "POINT AFTER FAILED"
4100 GOTO 3940
4110 REM
4120 REM ***SAFETY***
4130 REM
4140 SCR(QTR,OFF) = SCR(QTR,OFF) + 2: PRINT : PRINT TEAMS(AN); " IS CAUGHT FOR A SAFETY!"
4150 SAFETY = 20:L1 = 0:L2 = 0: FOR I = 1 TO QTR:L1 = L1 + SCR(I,1):L2 = L2 + SCR(I,2): NEXT I
4160 GOSUB 560: IF OVT = 1 THEN 4180
4170 ITIME = (ITIME - 7) - FN A(5):YARD(1) = AN:YARD(2) = 20: GOTO 3380
4180 QTR = QTR + 1:QTR = QTR - 1:THIS = 99:ITIME = 60 * 15
4190 IF QTR = 5 THEN 4300
4200 IF QTR = 3 THEN 4220
4210 TEXT : HOME : VTAB 12: GOSUB 1300: PRINT " END OF QUARTER ";QTR;" : "; GOSUB 1300: FOR I = 1 TO 1000: NEXT I: GOSUB 250: GOTO 1580
4220 TEXT : HOME : GOSUB 1300: PRINT "END OF THE FIRST HALF "; GOSUB 1300: PRINT :TMOU(1) = 3:TMOU(2) = 3:WARN = 0
4230 IF BEGIN = 2 THEN 4250
4240 AN = 2:OFF = 1: GOTO 4260
4250 AN = 1:OFF = 2
4260 L1 = SCR(1,1) + SCR(2,1):L2 = SCR(1,2) + SCR(2,2)
4270 PRINT : PRINT "SCORE AT HALF TIME": PRINT TEAMS(1); TAB(13);SCR(1,1); TAB(17);SCR(2,1); TAB(21);"-" ; TAB(25);L1
4280 PRINT TEAMS(2); TAB(13);SCR(1,2); TAB(17);SCR(2,2); TAB(21);"-" ; TAB(25);L2
4290 PRINT : PRINT "STATISTICS AT HALFTIME": GOTO 4380
4300 TEXT : HOME : HTAB 5: GOSUB 1300: PRINT " BANG!! "; GOSUB 1300: PRINT : PRINT TAB(6);"THAT'S THE END OF THE GAME!"
4310 L1 = SCR(1,1) + SCR(2,1) + SCR(3,1) + SCR(4,1):L2 = SCR(1,2) + SCR(2,2) + SCR(3,2) + SCR(4,2)
4320 PRINT : PRINT "FINAL SCORE AT THE FOOTBALL BOWL": PRINT TEAMS(1); TAB(13);SCR(1,1); TAB(17);SCR(2,1); TAB(21);SCR(3,1); TAB(25);SCR(4,1); TAB(29);"-" ; TAB(33);L1
4330 PRINT TEAMS(2); TAB(13);SCR(1,2); TAB(17);SCR(2,2); TAB(21);SCR(3,2); TAB(25);SCR(4,2); TAB(29);"-" ; TAB(33);L2
4340 REM
4350 REM ***STATISTICS***
4360 REM
4370 PRINT : PRINT "FINAL STATISTICS"
4380 ZPLAY(1) = RUSH(1) + PASS(1) + LOSS(1):ZPLAY(2) = RUSH(2) + PASS(2) + LOSS(2)
4390 T1 = YRUSH(1) + RPASS(1):T2 = YRUSH(2) + RPASS(2):X1R = 0:X2R = 0:P1X = 0:P2X = 0
4400 IF RUSH(1) > 0 THEN X1R = YRUSH(1) / RUSH(1)
4410 IF RUSH(2) > 0 THEN X2R = YRUSH(2) / RUSH(2)
4420 IF PASS(1) > 0 THEN P1X = RPASS(1) / PASS(1)
4430 IF PASS(2) > 0 THEN P2X = RPASS(2) / PASS(2)
4440 XPUNT(1) = 0:XPUNT(2) = 0: IF PUNT(1) > 0 THEN XPUNT(1) = YPUNT(1) / PUNT(1)
4450 IF PUNT(2) > 0 THEN XPUNT(2) = YPUNT(2) / PUNT(2)
4460 PRINT TAB(17);TEAMS(1); TAB(28);TEAMS(2)
4470 PRINT "NO PLAYS": TAB(18);ZPLAY(1); TAB(29);ZPLAY(2)
4480 PRINT "FIRST DOWNS": TAB(18);FIRST(1); TAB(29);FIRST(2)
4490 PRINT "YARDS RUSH": TAB(18);YRUSH(1); TAB(29);YRUSH(2)
4500 PRINT "AVER RUSH": TAB(18);FN Z(X1R); TAB(29);FN Z(X2R)
4510 PRINT "COMP/ATT": TAB(18);COMP(1);"/";PASS(1); TAB(29);COMP(2);"/";PASS(2)
4520 PRINT "YARDS PASS": TAB(18);RPASS(1); TAB(29);RPASS(2)
4530 PRINT "AVER PASS": TAB(18);FN Z(P1X); TAB(29);FN Z(P2X)
4540 PRINT "Q.B. SACKS": TAB(18);VSACK(1); TAB(29);VSACK(2)
4550 PRINT "NO OF PUNTS": TAB(18);PUNT(1); TAB(29);PUNT(2)
4560 PRINT "AVER YARDS/PUNT": TAB(18);FN Z(XPUNT(1)); TAB(29);FN Z(XPUNT(2))
4570 PRINT "INTERCEPTIONS": TAB(18);NENT(1); TAB(29);NENT(2)
4580 PRINT "FUMBL/RECOV": TAB(18);FUM(1);"/";RECO(1); TAB(29);FUM(2);"/";RECO(2)
4590 PRINT "TOTAL YARDS": TAB(18);T1; TAB(29);T2
4600 PRINT "FIELD GOALS": TAB(18);MADE(1);"/";FLD(1); TAB(29);MADE(2);"/";FLD(2)
4610 PRINT "PENALTIES": TAB(18);NPEN(1);"/";PPEN(1); TAB(29);NPEN(2);"/";PPEN(2)
4620 IF THIS < > 12 THEN 4650

```

```

4630 PRINT : PRINT : SPEED= ISPD: PRINT "HIT ANY KEY TO CONTINUE "; GET AS
4640 GOSUB 250: GOTO 1580
4650 IF OQTR < > 2 THEN 4670
4660 PRINT : PRINT : PRINT "HIT ANY KEY TO CONTINUE "; GET AS: GOTO 1400
4670 IF L1 < > L2 THEN 4740
4680 FOR I = 1 TO 2800: NEXT I: HOME : VTAB 5: HTAB 4: PRINT "***** OVERTIME *****": VTAB 9: HTAB 5: PRINT "THE SCORE IS TIED AND THE GAME"
4690 PRINT : PRINT "MOVES TO A FIFTEEN MINUTE SUDDEN DEATH": PRINT : PRINT "OVERTIME PERIOD. YOU HAVE TWO TIMEOUTS.": PRINT : PRINT "GOOD LUCK"
4700 FOR I = 1 TO 3000: NEXT I
4710 ITIME = 900:QOQTR = QOQTR - 1:OVT = 1:QTR = QTR - 1
4720 WARN = 0:TMOU(1) = 2:TMOU(2) = 2:TIMSET = 0
4730 GOTO 4770
4740 POKE 9,Z1: CLEAR : ALPHA = 1:Z1 = PEEK (9)
4750 PRINT : INPUT "DO YOU WANT TO PLAY AGAIN? ";ANS$: IF MID$(ANS$,1,1) = "Y" THEN 110
4760 SPEED= 255: END
4770 TEXT : HOME
4780 PRINT "*****"
FOOTBALL 3.0 ** BY
** PRESTON R BLACK MD **
AND **
4790 PRINT ** RICHARD I MORRIS JR **
** COPYRIGHT 1982 BY MICRO-SPARC, INC. *****
4800 IF HPL$ = "PLAYS" THEN 4930
4810 POKE 34,10
4820 IF ALPHA = 1 THEN 5090
4830 IF OVT = 1 THEN 5100
4840 POKE 34,10: INPUT "DO YOU WANT INSTRUCTIONS? ";ANS$: IF MID$(ANS$,1,1) = "Y" THEN 4860
4850 GOTO 5030
4860 PRINT "FOOTBALL 3.0 IS AN INTERACTIVE STRATEGY GAME WHICH ALLOWS YOU TO PIT YOUR SKILL AS THE COACH OF A FOOTBALL TEAM AGAINST THE APPLE II COMPUTER. YOU CALL EACH PLAY AS YOU MARCH YOUR TEAM UP AND DOWN"
4870 PRINT "THE FIELD. YOU HAVE YOUR CHOICE OF THREE PASSING PLAYS AND FIVE RUNNING PLAYS. YOU ALSO HAVE A COMPLETE KICKING GAME WHICH INCLUDES PUNTS, QUICK KICKS AND FIELD GOAL ATTEMPTS. BUT"
4880 PRINT "BEWARE! IF YOU TRY A FIELD GOAL FROM OUTSIDE THE TWENTY YARD LINE, THE BALL WILL BE PLACED THERE IF YOU MISS. THERE ARE GAME BREAKING LONG RUNS AND LONG": GET AS
4890 HOME
4900 PRINT "BOMBS. THERE ARE FUMBLES, INTERCEPTIONS, QUARTERBACK SACKS, AND SAFETIES. THERE ARE DRIVE STOPPING PENALTIES. YOU CAN RUN YOUR TWO MINUTE DRILL BY USING YOUR TIME-OUTS WISELY. WHEN THE APPLE IS ON"
4910 PRINT "OFFENSE IT WILL CHOOSE PLAYS AUTOMATICALLY. WHEN YOU ARE ON OFFENSE YOU WILL BE PROMPTED BY 'NEW PLAY' AND YOU CAN PICK ONE OF FIFTEEN OPTIONS.": PRINT : PRINT "THE CURRENT PLAYS ARE:": GET AS
4920 PRINT
4930 POKE 34,9: PRINT : SPEED= 255
4940 PRINT TAB(5);"LPASS"; TAB(11);"-" ; LONG PASSING PLAY"
4950 PRINT TAB(5);"SPASS"; TAB(11);"-" ; SHORT PASSING PLAY"
4960 PRINT TAB(5);"SCRN"; TAB(11);"-" ; SCREEN PASS"
4970 PRINT TAB(5);"MRUN"; TAB(11);"-" ; RUN UP THE MIDDLE"
4980 PRINT TAB(5);"OTRUN"; TAB(11);"-" ; OFF TACKLE RUN"
4990 PRINT TAB(5);"DRAW"; TAB(11);"-" ; DRAW PLAY"
5000 PRINT TAB(5);"SWEEP"; TAB(11);"-" ; A SWEEP"
5010 PRINT TAB(5);"PUNT"; TAB(11);"-" ; A PUNT"
5020 PRINT TAB(5);"KICK"; TAB(11);"-" ; A QUICK KICK"
5030 PRINT TAB(5);"FLDGL"; TAB(11);"-" ; A FIELD GOAL ATTEMPT"
5040 PRINT TAB(5);"STAT"; TAB(11);"-" ; LIST OF CURRENT STATISTICS: PRINT TAB(5);"TMOU"; TAB(11);"-" ; TAKES A TIME-OUT: PRINT TAB(5);"PLAYS"; TAB(11);"-" ; LIST CURRENT PLAYS"
5050 PRINT TAB(40); GET AS
5060 HOME : POKE 34,10: IF HPL$ > < "PLAYS" THEN 5030
5070 GOSUB 250: GOTO 1580
5080 HOME : PRINT "IN ORDER TO INITIALIZE THE RANDOM: PRINT "NUMBER GENERATOR, PLEASE ENTER A ": INPUT "A NUMBER BETWEEN 1 AND 100: ";NN
5090 IF NN < 1 OR NN > 100 THEN 5030
5100 FOR I = 0 TO NN:X = RND (1): NEXT I
5110 PRINT : PRINT "PLEASE ENTER A DISPLAY SPEED": PRINT "FROM 1 (VERY SLOW TYPING): INPUT "TO 5 (VERY FAST TYPING): ";Z1
5120 IF Z1 > = 1 AND Z1 < = 5 THEN 5090
5130 VTAB 14: GOTO 5060
5140 PRINT : INPUT "WHAT IS THE NAME OF YOUR TEAM? ";TEAMS(2): PRINT : INPUT "WHAT TEAM IS THE APPLE II? ";TEAMS(1): HOME : PRINT "IT IS NOW TIME FLIP THE COIN": PRINT "AS THE CAPTAIN OF ";TEAMS(2);" YOU: PRINT "GET TO CALL IT "; INPUT "HEADS OR TAILS: ";ACHOIC$
5150 FLIP = FN A(2): IF FLIP = 1 THEN CHOICE$ = "H"
5160 IF FLIP = 2 THEN CHOICE$ = "T"
5170 IF LEFT$(ACHOIC$,1) = CHOICE$ THEN 5150
5180 AN = 1:OFF = 2:BEGIN = AN: PRINT : PRINT "SORRY ";TEAMS(2);" BUT ";TEAMS(1); PRINT "WINS THE TOSS AND WILL RECEIVE!": GOTO 5160
5190 AN = 2:OFF = 1:BEGIN = AN: PRINT TEAMS(2);" WINS THE TOSS": PRINT "AND WILL RECEIVE!"
5200 FOR I = 1 TO 1000: NEXT I:ISPD = 50 * Z1 + 5: SPEED= ISPD: GOTO 1340

```



Lower Case Letters (With Hi-Res Graphics)

by James Totten

Software Plus

1347 Monmouth Dr. Burlington, Ontario
Canada, L7P 3J7

There are various ways to obtain lower case letters on the Apple II and Apple II Plus computers. However, almost all of these require buying new RAM or ROM boards, or expensive software. But remember, I said **almost** all.

This idea came to me while writing a game in Hi-Res graphics. I needed letters and numbers on the top of the screen, and so I tried to figure out how to combine text and Hi-Res. But I suddenly realized: Why bother with trying to make a graphics/text combination? Why not just do with the letters what I had done with the aliens? So, I sat down and 'wrote' a **master 85-character shape table**.

THE MASTER SHAPE TABLE

This shape table contains literally every character present on the Apple's keyboard. Lower case letters are defined as standard, and upper case letters are tagged on the end of the table. This is because you'll probably use lower case more often than you will use upper case. **Each character is in its proper ASCII position (the ASCII character 43 is shape number 43), and that makes using the table very easy.**

ENTERING THE SHAPES

First, type **CALL-151** to enter the Monitor. Then you can begin to enter the shape tables directly into memory by typing **4000: 58 00 B2 00 C5 00** etc. from the memory listing. (See the Letters section for more information on entering machine code into memory.)

After you've typed in the character set (whew), you can save it with **BSAVE CHARACTERS, A\$4000, L1930**. Notice that the table is stored in Hi-Res **page two**. This can be changed if you wish; however you will have to alter the values in the two **POKEs** before using relocated shapes (see the demo program). You have to set **LOMEM** just past the Hi-Res page two if the shapes are to be used with a program.

Next, there is a small demo program using the shapes and full screen graphics in Hi-Res page one. When you finish analyzing the program, try some things on your own. You'll love the new letters!

THE DEMO

The demonstration program shows how the character shapes are used. This program can be modified to fit your needs, or even used as a screen printing subroutine in a major program. Again, if this is done, then **LOMEM** will have to be set just beyond Hi-Res page two. Here is a variable list:

- KS** — Key pressed
- DS** — DOS operator
- X,Y** — screen locations
- UC** — upper case indicator
- UL** — upper lock indicator
- CS** — character typed from keyboard
- CC** — character code (ASCII — DRAW value)
- L** — loop counter

PROGRAM DESCRIPTION

- 80-130** Loads the character table into memory (if needed), and completes the necessary **POKEs** and variable settings.
- 140** Plots cursor and gets a character (C\$).
- 160-180** Checks to see if C\$ is an upper case set or an upper case lock. If set, then UC = 1 for the next character.
- 190-200** Checks to see if C\$ is a cursor move.
- 210-220** Resets CC to the proper value of the shape.
- 230** If the character is a letter and there is no lower case restriction (UC or UL), then moves Y down by 1 line so that the letter will line up with the other characters. After drawing the character, Y is reset to its proper value.
- 240-260** Clears area and draws the character.
- 270-310** Checks to make sure that the cursor is not going above or below its maximum or minimum allowable value.

HOW TO USE THE DEMO PROGRAM

To use demo program, you must first have the character shapes in memory. However, this is taken care of by the demo program itself.

After the loading is finished, you'll be presented with a blank Hi-Res screen, and a small white line at the upper left corner. This is your cursor. It can be controlled with either arrow key (they work as they should; they will not erase any character), or by pressing any character on the keyboard. If the cursor should move past the right edge of the screen, it will reappear on the next line (on the left). The RETURN key works normally as well. A note of caution is in order, however. When the cursor reaches the bottom of the screen, it will automatically return to the top of the screen. Therefore, if you continue typing past the lower right corner of the screen, you'll begin over-printing the first line at the top of the screen.

When you wish to use an upper case letter, press the **ESC** key once. You'll hear a beep when you press the key, and then the **next letter** you type will be upper case. This will hold for only one letter. If you wish to do a string of upper case letters, **CTRL-N** will activate the upper case lock. Every letter after **CTRL N** will appear in upper case. To deactivate the upper case lock, type **CTRL-L**. Again, you'll hear a small beep and you'll return to lower case printing. You'll find that Lower Case Letters are useful for games and many other applications. Let us know the uses that you discover!



*4000. 478B

```

4000- 58 00 B2 00 C5 00 D8 00
4008- EC 00 02 01 15 01 29 01
4010- 3C 01 4F 01 62 01 75 01
4018- 8A 01 9D 01 B0 01 C3 01
4020- D6 01 E9 01 FE 01 12 02
4028- 26 02 3B 02 50 02 65 02
4030- 79 02 8D 02 A2 02 B6 02
4038- C9 02 DD 02 F1 02 06 03
4040- 19 03 2C 03 41 03 55 03
4048- 69 03 7D 03 91 03 A5 03
4050- B8 03 CC 03 DF 03 F2 03
4058- 06 04 19 04 2C 04 40 04
4060- 54 04 68 04 7C 04 8F 04
4068- A3 04 B6 04 C9 04 DD 04
4070- F1 04 05 05 1A 05 2E 05
4078- 41 05 54 05 67 05 7C 05
4080- 90 05 A3 05 B7 05 CC 05
4088- E0 05 F4 05 08 06 1C 06
4090- 30 06 43 06 57 06 6B 06
4098- 7F 06 94 06 A8 06 BC 06
40A0- D0 06 E4 06 F8 06 0D 07
40A8- 21 07 36 07 4B 07 5F 07
40B0- 74 07 49 09 1A 1B 1B 4A
40B8- 49 1A 1B 1B 4A 49 1A 1B
40C0- 1B 4A 49 02 00 09 4D 1A
40C8- 1B 1F 4A 4D 1A 1B 1F 4A
40D0- 4D 1A 1B 1B 4A 4D 02 00
40D8- 69 0D 1A 3B 3B 0A 0D 0D
40E0- 1A 1B 1B 4A 49 1A 1B 1B
40E8- 4A 49 02 00 69 0D 1A 3B
40F0- 3B 2A 2D 2D 1A 3B 3B 2A
40F8- 2D 2D 1A 3B 3B 0A 0D 0D
4100- 02 00 09 4D 1A 3F 3F 6A
4108- 4D 1A 3B 3F 4A 0D 15 1B
4110- 3F 77 69 11 00 6D 09 1A
4118- 1F 3B 4E 69 1A 1B 1F 0A
4120- 4D 11 3B 1F 73 09 2D 02
4128- 00 69 09 1A 1B 1F 6E 4D
4130- 1A 1B 3B 6A 0D 15 1B 1F
4138- 73 6D 15 00 49 0D 1A 1B
4140- 1F 0A 4D 11 1B 1B 53 49
4148- 11 1B 1B 53 49 11 00 09
4150- 4D 1A 1B 3B 6A 49 1A 1B
4158- 1B 6E 49 1A 1B 3B 4A 4D
4160- 02 00 09 4D 1A 3B 1B 4A
4168- 09 15 3B 1B 53 49 15 1B
4170- 1F 53 69 11 00 09 4D 1A
4178- 1F 1F 0E 2D 0D 1A 3F 3F
4180- 0E 2D 0D 1A 1F 1F 4E 4D
4188- 02 0D 49 09 1A 1B 1F 4A
4190- 4D 1A 3F 3F 4E 4D 1A 1B
4198- 1F 4A 49 02 00 49 09 1A
41A0- 1B 1B 4A 49 1A 1B 1B 4A
41A8- 6D 1A 3B 1F 0A 6D 11 00
41B0- 49 09 1A 1B 1B 4A 49 1A
41B8- 3F 3F 4E 49 1A 1B 1B 4A
41C0- 49 02 00 49 09 1A 1B 1B
41C8- 4A 49 1A 1B 1B 4A 49 1A
41D0- 1B 3F 0A 6D 11 00 49 09
41D8- 1A 1F 1B 4A 69 1A 1B 1F
41E0- 0A 4D 11 1B 1B 73 49 11
41E8- 00 29 6D 1A 1F 1B 6E 29
41F0- 15 3B 3B 33 6D 29 1A 1F
41F8- 1B 0E 2D 0D 02 00 09 4D
4200- 1A 1B 3F 4A 4D 1A 1B 1F
4208- 4A 4D 1A 1B 1F 0A 2D 0D
4210- 02 00 29 6D 1A 1F 1B 4E
4218- 09 15 1B 3F 53 4D 11 1B
4220- 1B 33 2D 2D 15 00 2D 2D
4228- 15 3B 1B 53 09 0D 1A 3B
4230- 1F 4A 09 15 3B 1B 73 2D
4238- 0D 02 00 49 0D 1A 3B 1F
4240- 0A 0D 0D 1A 3B 1B 2E 2D
4248- 2D 1A 3B 1B 4A 69 02 00
4250- 2D 2D 15 1B 1B 33 2D 6D
4258- 1A 1F 1B 4A 09 15 3B 1B
4260- 73 2D 0D 02 00 29 6D 1A
4268- 1F 1B 6E 49 1A 3B 3F 6E

```



```

4270- 09 15 3B 1B 73 2D 0D 02
427B- 00 2D 2D 15 3B 1B 53 09
4280- 0D 1A 1B 1F 0A 4D 11 1B
428B- 1B 57 4D 11 00 29 6D 1A
4290- 1F 1B 6E 09 15 1B 3F 17
429B- 4D 29 1A 1F 1B 0E 2D 0D
42A0- 02 00 29 6D 1A 1F 1B 6E
42AB- 09 15 3B 3F 57 49 15 3B
42B0- 1B 73 2D 0D 02 00 49 09
42BB- 1A 1B 3F 0A 6D 11 1B 1B
42C0- 53 6D 11 1B 3B 57 49 11
42CB- 00 49 09 1A 1B 3F 0A 6D
42D0- 11 1B 1B 53 6D 11 1B 3B
42DB- 17 6D 09 02 00 49 2D 1A
42E0- 3B 1F 0A 6D 11 1B 1B 77
42EB- 6D 11 1B 3F 53 09 2D 02
42F0- 00 49 09 1A 1B 1B 0A 2D
42FB- 0D 1A 1B 1B 0A 2D 0D 1A
4300- 1B 1B 4A 49 02 00 6D 09
430B- 1A 1B 3F 4A 6D 1A 3F 1B
4310- 4A 6D 1A 1B 3F 2A 4D 11
431B- 00 29 6D 1A 1F 1B 4E 09
4320- 15 1B 3F 53 4D 11 1B 1B
432B- 53 4D 11 00 29 6D 1A 1F
4330- 1B 6E 0D 15 3B 3F 33 0D
433B- 0D 15 1B 1B 73 2D 2D 02
4340- 00 49 09 1A 3B 3F 4A 09
434B- 15 3B 0F 17 4D 29 1A 3F
4350- 3F 4A 49 02 00 4D 09 1A
435B- 3B 1F 2E 4D 15 3B 1B 33
4360- 6D 29 1A 3B 1F 4E 49 02
436B- 00 49 09 1A 3B 3F 6A 09
4370- 15 1B 1B 33 4D 29 1A 3B
437B- 3F 4A 49 02 00 49 29 1A
4380- 1F 3F 6A 29 15 3B 1B 33
438B- 4D 2D 1A 1F 3F 4A 49 02
4390- 00 49 09 1A 3B 3F 6A 09
439B- 15 3B 3F 37 4D 09 1A 3B
43A0- 3F 4A 49 02 00 09 6D 1A
43AB- 1F 3B 0A 4D 11 1B 3B 77
43B0- 4D 11 1B 1B 57 49 11 00
43BB- 49 09 1A 1F 3F 6A 29 15
43C0- 3B 1F 73 6D 15 1B 1B 53
43CB- 2D 0D 02 00 4D 09 1A 1B
43D0- 1B 6E 6D 1A 1F 3B 6E 09
43DB- 15 3B 1B 73 49 11 00 09
43E0- 4D 1A 1B 1B 0A 6D 11 1B
43EB- 3B 53 69 11 1B 3F 57 49
43F0- 11 00 49 29 1A 1B 1B 4A
43FB- 29 15 3B 1B 53 49 15 3B
4400- 1B 73 2D 0D 02 00 4D 09
440B- 1A 3B 1B 6E 4D 1A 1B 3B
4410- 6E 4D 1A 3B 1B 4E 49 02
441B- 00 29 4D 1A 1B 1F 4A 4D
4420- 1A 1B 1F 4A 4D 1A 3B 3F
442B- 4A 49 02 00 49 09 1A 3B
4430- 3B 6A 0D 15 3B 3B 33 0D
443B- 0D 15 3B 3B 73 49 11 00

```

```

4440- 49 09 1A 3B 1F 2E 4D 15
444B- 3B 1B 33 4D 29 1A 1F 1B
4450- 4E 49 02 00 49 09 1A 3B
445B- 3F 6A 09 15 3B 1B 33 4D
4460- 29 1A 3B 3F 4A 49 02 00
446B- 49 09 1A 3B 1F 2E 4D 15
4470- 3B 1B 37 0D 6D 1A 1B 1B
447B- 6E 49 02 00 49 09 1A 1F
4480- 3F 6A 29 15 3B 1F 73 6D
448B- 15 3B 1B 53 49 15 00 49
4490- 09 1A 3B 1F 2E 4D 15 1B
449B- 1B 33 4D 09 1A 1B 1B 4E
44A0- 49 02 00 49 09 1A 3F 3F
44AB- 6A 49 1A 3B 3F 4A 09 15
44B0- 1B 3F 77 49 11 00 69 09
44BB- 1A 1B 3F 0E 4D 11 1B 1B
44C0- 57 4D 15 1B 3F 53 49 11
44CB- 00 49 09 1A 1F 1B 6E 09
44D0- 15 3B 1B 33 4D 2D 1A 1F
44DB- 3F 4A 49 02 00 49 09 1A
44E0- 1F 1B 6E 09 15 1B 1F 57
44EB- 00 0D 1A 1B 1F 4A 49 02
44F0- 00 49 09 1A 1F 1F 6E 0D
44FB- 15 3B 3B 33 0D 0D 15 1B
4500- 1F 57 49 11 00 49 09 1A
450B- 1F 1B 0E 0D 0D 1A 1B 1F
4510- 0A 0D 0D 1A 1F 1B 4E 49
451B- 02 00 49 09 1A 1F 1B 6E
4520- 09 15 3B 1F 73 6D 15 3B
452B- 1B 53 2D 0D 02 00 49 09
4530- 1A 3F 3F 4E 69 1A 1B 1F
453B- 0A 4D 11 3B 3F 77 49 11
4540- 00 29 4D 1A 3B 1B 4A 69
454B- 1A 1F 1B 4A 69 1A 3B 1B
4550- 0A 6D 11 00 09 4D 1A 3B
455B- 3B 6A 09 15 1B 1B 53 49
4560- 11 1B 1B 53 49 11 00 09
456B- 4D 1A 3B 3B 6A 09 15 3B
4570- 1B 33 2D 2D 15 3B 1B 33
457B- 4D 29 02 00 2D 6D 1A 1F
4580- 3B 0A 4D 15 1B 3F 57 4D
458B- 15 3B 1B 17 2D 6D 02 00
4590- 09 6D 1A 1F 3B 6A 49 1A
459B- 1B 1B 6E 49 1A 1F 3B 4A
45A0- 6D 02 00 2D 6D 1A 1F 3B
45AB- 0A 4D 15 3B 1B 57 4D 15
45B0- 3B 1B 17 2D 6D 02 00 2D
45BB- 2D 15 3B 1B 33 4D 09 1A
45C0- 1B 3F 6E 49 1A 1F 1B 2E
45CB- 2D 2D 02 00 2D 2D 15 3B
45D0- 1B 33 4D 09 1A 1B 3F 6E
45DB- 49 1A 1B 1B 6E 49 02 00
45E0- 29 6D 1A 1F 1B 6E 49 1A
45EB- 3F 1F 6E 09 15 3B 1B 73
45F0- 2D 0D 02 00 4D 29 1A 1F
45FB- 1B 6E 09 15 3B 3F 37 4D
4600- 29 1A 1F 1B 6E 09 15 00
460B- 29 6D 1A 1B 1F 4A 4D 1A

```

```

4610- 1B 1F 4A 4D 1A 1B 1F 0A
461B- 2D 0D 02 00 09 2D 15 1B
4620- 1F 53 09 0D 1A 3B 1B 4A
462B- 69 1A 3B 1B 0E 6D 11 00
4630- 4D 29 1A 3B 1B 6E 4D 1A
463B- 1B 3B 6E 4D 1A 3B 1B 6E
4640- 09 15 00 6D 09 1A 1B 3B
464B- 0A 4D 11 1B 1B 57 4D 11
4650- 3B 1B 17 2D 2D 15 00 4D
465B- 29 1A 3F 3B 6E 0D 15 3B
4660- 1B 33 4D 29 1A 1F 1B 6E
466B- 09 15 00 4D 29 1A 1F 1B
4670- 2E 4D 15 3B 3B 33 4D 2D
467B- 1A 1F 1B 6E 09 15 00 29
4680- 6D 1A 1F 1B 6E 09 15 3B
468B- 1B 33 4D 29 1A 1F 1B 0E
4690- 2D 0D 02 00 2D 6D 1A 1F
469B- 3B 0A 4D 15 1B 3F 57 4D
46A0- 11 1B 1B 17 6D 09 02 00
46AB- 29 6D 1A 1F 1B 6E 09 15
46B0- 3B 1B 33 0D 0D 15 1B 1F
46BB- 73 6D 15 00 2D 6D 1A 1F
46CB- 3B 0A 4D 15 1B 3F 57 0D
46CB- 0D 1A 1F 3B 2A 4D 15 00
46DB- 29 6D 1A 1F 1B 6E 49 1A
46DB- 3B 3F 4A 09 15 3B 1B 73
46E0- 2D 0D 02 00 2D 2D 15 3B
46EB- 3B 73 69 11 1B 3B 53 69
46F0- 11 1B 3B 53 2D 0D 02 00
46FB- 4D 29 1A 1F 1B 6E 09 15
4700- 3B 1B 33 4D 29 1A 1F 1B
470B- 0E 2D 0D 02 00 4D 29 1A
4710- 1F 1B 6E 09 15 3B 1B 33
471B- 4D 29 1A 3B 3B 4A 4D 02
4720- 00 4D 29 1A 1F 1B 6E 09
472B- 15 3B 1B 33 0D 0D 15 3B
4730- 1F 37 4D 29 02 00 4D 29
473B- 1A 1F 1B 0E 0D 0D 1A 1B
4740- 1F 0A 0D 0D 1A 1F 1B 6E
474B- 09 15 00 4D 29 1A 1F 1B
4750- 0E 0D 0D 1A 1B 1F 4A 4D
475B- 1A 1B 1F 4A 4D 02 00 2D
4760- 2D 15 3B 1B 73 09 0D 1A
476B- 1B 1F 0A 4D 11 3B 1B 33
4770- 2D 2D 15 00 2D 2D 15 3B
477B- 3F 37 2D 2D 15 3B 3F 37
4780- 2D 2D 15 3B 3F 37 2D 2D
478B- 15 00 0C 20

```

```

10 REM *****
20 REM # HI-RES CHAR. DEMO #
30 REM # BY JAMES TOTTON #
40 REM # COPYRIGHT (C) 1982 #
50 REM # BY MICRO-SPARC INC #
60 REM # LINCOLN, MA. 01773 #
70 REM *****
80 D$ = CHR$ (13) + CHR$ (4)
90 TEXT : HOME : PRINT : PRINT "HIGH-WRITER": PRINT :
  PRINT "BY JAMES TOTTON": PRINT : INVERSE :
  PRINT "NOW LOADING SHAPE TABLE FROM DISK.": PRINT
100 NORMAL
110 PRINT D$ "BLOOD CHARACTERS": POKE 232,0: POKE 233,
  64
120 X = 2:Y = 5: SCALE= 1: ROT= 0
130 HGR : HCOLOR= 3: POKE - 16302,0: UC = 0:
140 HCOLOR= 3: HPLLOT X,Y + 8 TO X + 5,Y + 8: HOME : GET
  C$
150 IF C$ = CHR$ (14) THEN CALL - 198:UL = 1: GOTO
  140

```

```

160 IF C$ = CHR$ (27) THEN UC = 1: CALL - 198: BOTO
  140
170 IF C$ = CHR$ (12) THEN CALL - 198:UL = 0: BOTO
  140
180 IF C$ = CHR$ (13) THEN HCOLOR= 0: HPLLOT X,Y + 8
  TO X + 5,Y + 8: X = 2:Y = Y + 10: GOSUB 270: GOTO
  140
190 IF C$ = CHR$ (8) THEN HCOLOR= 0: HPLLOT X,Y + 8 TO
  X + 5,Y + 8: X = X - 7: GOSUB 270: GOTO 140
200 IF C$ = CHR$ (21) THEN HCOLOR= 0: HPLLOT X,Y + 8
  TO X + 5,Y + 8: X = X + 7: GOSUB 270: GOTO 140
210 CC = ASC (C$) - 31
220 IF (UL OR UC) AND (ASC (C$) > 64 AND ASC (C$) <
  91) THEN CC = CC + 28
230 IF ((UL = 0) AND (UC = 0)) AND (ASC (C$) > 64 AND
  ASC (C$) < 91) THEN HCOLOR= 0: FOR L = X TO X +
  5: HPLLOT L,Y TO L,Y + 7: NEXT L: HCOLOR= 3:Y = Y +
  1: DRAW CC AT X,Y:Y = Y - 1: GOTO 250
240 HCOLOR= 0: FOR L = X TO X + 5: HPLLOT L,Y TO L,Y +
  7: NEXT L: HCOLOR= 3: DRAW CC AT X,Y: IF UC THEN
  UC = 0
250 HCOLOR= 0: HPLLOT X,Y + 8 TO X + 5,Y + 8: X = X + 7
  : GOSUB 270
260 GOTO 140
270 IF X > 270 THEN X = 2:Y = Y + 10
280 IF Y > 180 THEN Y = 5
290 IF X < 2 THEN X = 268:Y = Y - 10
300 IF Y < 5 THEN Y = 5
310 RETURN

```


TIPS 'N TECHNIQUES

Apple Scroller

by Bill Fortenberry
2906 36 Ave.
Meridan, MS 39301

One of the most confusing things to the beginning assembly language programmer has to be the subject of memory addressing. Out of the nine methods offered by the 6502 (Apple processor chip), indirect and indexed addressing are probably the most difficult to grasp. The advantage of these two forms is the ability to handle large amounts of data with a small amount of code and as we all know, compact machine code is spelled S-P-E-E-D.

INDEXED ADDRESSING

Indexed addressing can be directly compared to using **arrays** in BASIC. You have a starting point (the base address) and a pointer into the array (the index). For example, the Applesoft statement **X=A(5)** will cause the computer to find the **fifth item** in the array you named **A** and store that number in the variable **X**. In assembly language this would translate to something like **LDA \$305,Y** which tells the computer to go to the memory location **\$305 PLUS** whatever is in the **Y** register and store this value in the accumulator. In the above example, if the **Y** register contained the value **\$03**, the Accumulator would be loaded with whatever is in location **\$308 (\$305 + \$03)**.

INDIRECT ADDRESSING

Indirect addressing has no direct counterpart in BASIC, although it can easily be illustrated. Say you want to go visit Jim but you don't know where Jim lives. You do know that Fred knows where Jim lives, so you go to Fred's and ask him where Jim lives and then head over to Jim's. Because you didn't know Jim's exact address, you had to go **INDIRECTLY** through Fred. In short, indirect addressing means that **you go to one memory address and in that location you find the address of the number you really want**.

THE SCROLL DEMO

The following program uses both of these methods together to **scroll** the text screen **right, left, or down**. Since the Apple screen consists of known memory locations (this information is found on page 16 of the reference manual), we can move the data in these locations anywhere we want.

The program uses an array that holds the screen addresses. These addresses are the left column of the screen only, but by adding an index to this address, all forty columns can be addressed. Because the 6502 only allows indirect indexing from the zero page, all addresses must be pulled from the array and moved to the zero page before being used. All that remains is to pull the data from one screen location and move it to the next one.

This program was assembled at location **\$6000**. To use the program, just enter the system monitor and type in the hex numbers in the left column. **(For directions on entering machine code into your Apple see the instructions in the Letters column of this issue.)** You can then save them with the command

BSAVE SCROLL,A \$6000,L 258. The program is executed with the **CALL** statement. There are three addresses to call, depending on the direction you want the screen to go. **To scroll down, type CALL 24576**. **Left-Scrolling** would be exercised with **CALL 24658**, and **CALL 24721** will **scroll right**.

If you call the program once, only one row (or column) will scroll off the screen. This allows you to print, scroll, print, scroll until a new screen appears.

To clear the whole screen, use a statement such as: **FOR I=1 TO 40:CALL 24657:NEXT I**.



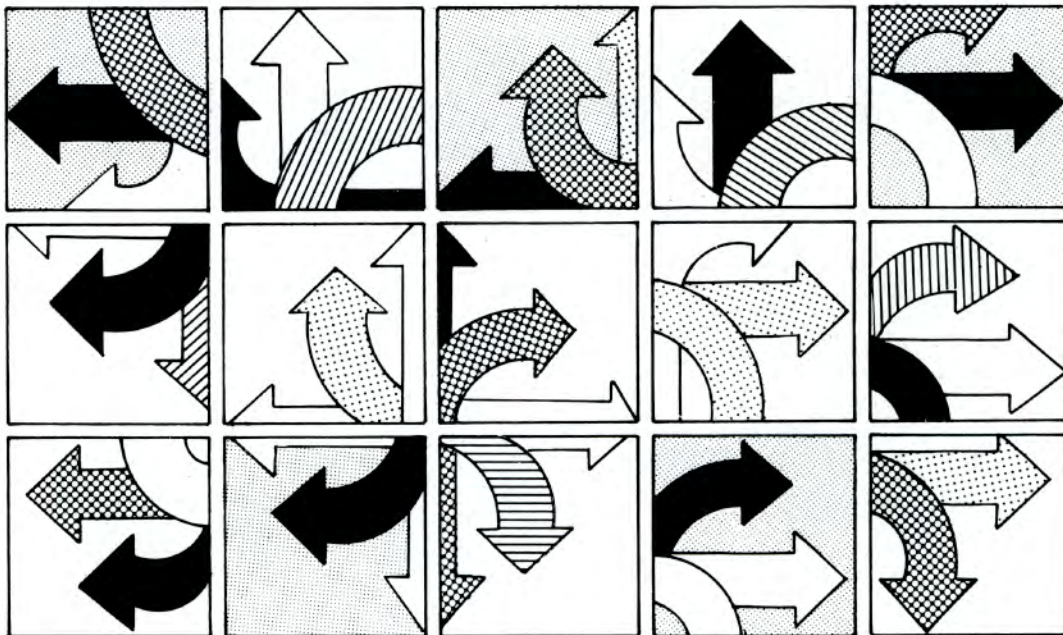
```
SOURCE FILE: SCREEN SCROLL
00FF:      1 A1 EQU $FF
00FE:      2 A2 EQU $FE
00FD:      3 A3 EQU $FD
00FC:      4 A4 EQU $FC
----- NEXT OBJECT FILE NAME IS SCREEN SCROLL
6000:      5 ORG $6000
6000:      6 *****
6000:      7 * SCREEN SCROLL DOWN
6000:      8 * BY BILL FORTENBERRY
6000:      9 * COPYRIGHT 1982 BY MICRO-SPARC, INC
6000:     10 * APPLE ASSEMBLER (DOS TOOL KIT)
6000:     11 *
6000:     12 * ENTRY POINTS
6000:     13 * SCROLL ADDRESS
6000:     14 * DOWN $6000
6000:     15 * LEFT $6052
6000:     16 * RIGHT $6091
6000:     17 *****
6000:     18 * START OF SCROLL DOWN
6000:A2 00 19 SETUP LDX #0
6002:BD D2 60 20 LDA DATA,X ;SET BEGINNING ADDRESSES
6005:85 FF 21 STA A1
6007:E8 22 INX
6008:BD D2 60 23 LDA DATA,X
600B:85 FE 24 STA A2
600D:E8 25 INX
600E:BD D2 60 26 LDA DATA,X
6011:85 FD 27 STA A3
6013:E8 28 INX
6014:BD D2 60 29 LDA DATA,X
6017:85 FC 30 STA A4
6019:E8 31 INX
601A:A0 00 32 START1 LDY #0
601C:B1 FC 33 CHANGE LDA (A4),Y ;DROP CHAR
601E:91 FE 34 STA (A2),Y ;ONE LINE
6020:C8 35 INY
6021:C0 28 36 CPY #40 ;WHOLE ROW?
6023:D0 F7 37 BNE CHANGE ;DO MORE
6025:18 38 CLC
6026:A5 FD 39 LDA A3
6028:65 FC 40 ADC A4
602A:C9 04 41 CMP #04 ;IS ADDRESS = $400
602C:F0 17 42 BEQ TROW ;LAST ROW?
602E:A5 FD 43 LDA A3
6030:85 FF 44 STA A1
6032:A5 FC 45 LDA A4
6034:85 FE 46 STA A2 ;TOP TO BOTTOM
6036:BD D2 60 47 LDA DATA,X ;GET NEW ADDRESS
6039:85 FD 48 STA A3
603B:E8 49 INX
603C:BD D2 60 50 LDA DATA,X
603F:85 FC 51 STA A4
6041:E8 52 INX
6042:4C 1A 60 53 JMP START1 ;DO NEW ROW
6045:A9 A0 54 TROW LDA #160 ;ASCII BLANK
6047:A0 00 55 LDY #0
6049:99 00 04 56 ERASE STA $400,Y ;BLANK TOP ROW
604C:C8 57 INY
604D:C0 28 58 CPY #40
604F:D0 F8 59 BNE ERASE
6051:60 60 RTS ;ALL DONE
6052: 61 * SCROLL RIGHT TO LEFT
6052:A2' 00 62 START2 LDX #0
6054:BD D2 60 63 GETAD LDA DATA,X ;GET ADDRESS
6057:85 FF 64 STA A1
6059:E8 65 INX
605A:BD D2 60 66 LDA DATA,X
605D:85 FE 67 STA A2
605F:E8 68 INX
6060:A0 00 69 LDY #0 ;SETUP POINTERS
6062:8C D0 60 70 STY I
6065:C8 71 INY
6066:8C D1 60 72 STY J ;INITILIZE POINTERS
6069:AC D1 60 73 LEFT LDY J
606C:B1 FE 74 LDA (A2),Y ;MOVE LETTER LEFT
606E:AC D0 60 75 LDY I
6071:91 FE 76 STA (A2),Y
6073:EE D0 60 77 INC I ;BUMP POINTERS
6076:EE D1 60 78 INC J
6079:AD D1 60 79 LDA J
607C:C9 28 80 CMP #40 ;CHECK FOR LAST COLUMN
607E:D0 E9 81 BNE LEFT
6080:A9 A0 82 LDA #160 ;ASCII BLANK
6082:AC D0 60 83 LDY I
6085:91 FE 84 STA (A2),Y ;BLANK OUT RIGHT COL
6087:A5 FF 85 LDA A1
6089:18 86 CLC
608A:65 FE 87 ADC A2
608C:C9 04 88 CMP #04 ;CHECK FOR TOP LINE
608E:D0 C4 89 BNE GETAD
6090:60 90 RTS ;ALL DONE
6091: 91 * SCROLL LEFT TO RIGHT
```



```

6091:A2 00      92 START3 LDX #0
6093:BD D2 60   93 FINDAD LDA DATA,X ;GET ADDRESS OF ROW
6096:85 FF      94      STA A1
6098:E8         95      INX
6099:BD D2 60   96      LDA DATA,X
609C:85 FE      97      STA A2
609E:E8         98      INX
609F:A0 26     99      LDY #38 ;SET POINTERS
60A1:BC D0 60  100     STY I
60A4:C8         101     INY
60A5:BC D1 60  102     STY J
60A8:AC D0 60  103 RIGHT LDY I
60AB:B1 FE      104     LDA (A2),Y ;MOVE LETTER RIGHT
60AD:AC D1 60  105     LDY J
60B0:91 FE      106     STA (A2),Y
60B2:CE D0 60  107     DEC I ;BUMP POINTERS DOWN
60B5:CE D1 60  108     DEC J
60B8:AD D1 60  109     LDA J
60BB:C9 00     110     CMP #0 ;LAST COL?
60BD:D0 E9     111     BNE RIGHT
60BF:AC D1 60  112     LDY J
60C2:A9 A0     113     LDA #160 ;ASCII BLANK
60C4:91 FE      114     STA (A2),Y ;BLANK LEFT COL
60C6:A5 FF      115     LDA A1
60C8:18         116     CLC
60C9:65 FE      117     ADC A2
60CB:C9 04     118     CMP #04 ;LAST ROW?
60CD:D0 C4     119     BNE FINDAD
60CF:60         120     RTS ;ALL DONE
60D0:         121 * TEMP STORAGE
60D0:00         122 I DFB #00
60D1:00         123 J DFB #00
60D2:         124 * SCREEN ADDRESS TABLE
60D2:07 D0 07  125 DATA DFB #07,$D0,#07,$50
60D5:50         126     DFB #06,$D0,#06,$50,#05,$D0,#05,$50
60D6:06 D0 06  127     DFB #04,$D0,#04,$50,#07,$AB,#07,$2B
60D7:50 05 D0  128     DFB #06,$AB,#06,$2B,#05,$AB,#05,$2B
60D8:05 50     129     DFB #04,$AB,#04,$2B,#07,$80,#07,$00
60D9:04 D0 04  130     DFB #06,$80,#06,$00,#05,$80,#05,$00,#04,$80,#04,$00
60E1:50 07 AB  131     DFB #06,$80,#06,$00,#05,$80,#05,$00,#04,$80,#04,$00
60E4:07 28     132     DFB #06,$80,#06,$00,#05,$80,#05,$00,#04,$80,#04,$00
60E6:06 AB 06  133     DFB #06,$80,#06,$00,#05,$80,#05,$00,#04,$80,#04,$00
60E9:28 05 AB  134     DFB #06,$80,#06,$00,#05,$80,#05,$00,#04,$80,#04,$00
60EC:05 28     135     DFB #06,$80,#06,$00,#05,$80,#05,$00,#04,$80,#04,$00
60EE:04 AB 04  136     DFB #06,$80,#06,$00,#05,$80,#05,$00,#04,$80,#04,$00
60F1:28 07 B0  137     DFB #06,$80,#06,$00,#05,$80,#05,$00,#04,$80,#04,$00
60F4:07 00     138     DFB #06,$80,#06,$00,#05,$80,#05,$00,#04,$80,#04,$00
60F6:06 B0 06  139     DFB #06,$80,#06,$00,#05,$80,#05,$00,#04,$80,#04,$00
60F9:00 05 B0  140     DFB #06,$80,#06,$00,#05,$80,#05,$00,#04,$80,#04,$00
60FC:05 00 04  141     DFB #06,$80,#06,$00,#05,$80,#05,$00,#04,$80,#04,$00
60FF:B0 04 00  142     DFB #06,$80,#06,$00,#05,$80,#05,$00,#04,$80,#04,$00

```





ROBERT
TINNEY

APPLE RECIPE BOX

Apple Recipe Box

by J.L. Costenbader
309 E. 1850 South
Bountiful, UT 84010

INTRODUCTION

When I purchased my Apple II Plus late last year, it was only with the approval of my wife. Part of the 'hard sell' that I laid on her was that she would find the computer helpful to her around the house. I couldn't find a way to get the computer to vacuum the floors, and it definitely wouldn't do windows, so I was left to discover some other task that would relieve my wife of some work. That's when the Apple RECIPE BOX was born. We had many good recipes scattered about the kitchen in assorted books, old boxes and on scraps of food-stained paper. Additionally, my wife sometimes found it confusing when she needed to double or triple a recipe or (worse yet) cut one in half!!! Well, our Apple solved all that. Now our recipes are stored neatly on a floppy disk. The program will double, triple or even cut a recipe in half, and when a friend asks for a copy of her favorite recipe, our trusty Paper Tiger rapidly prints one out for her. I even managed to let my wife know how I feel by leaving a note for her in line 5010.

SYSTEM REQUIREMENTS

The RECIPE BOX is written in Applesoft BASIC. It was programmed on a 48K machine, but should run on a 32K machine with some very minor changes. A printer is not necessary, but by modifying two lines the program should work with any printer. It is currently set up for an IDS-445.

OPERATING THE PROGRAM

When the program is run, it first tries to RENAME the INDEX.FILE, INDEX.FILE. If it finds that this is the first time the program has been run, the data files are initialized; otherwise pertinent index information is loaded into memory. The following menu is then presented:

- 1 = ENTER A NEW RECIPE
- 2 = CHANGE A RECIPE
- 3 = PRINT INDEX OF RECIPES
- 4 = PRINT A RECIPE
- 5 = EXIT SYSTEM

FUNCTION CHOICE =

If you choose 1=ENTER A NEW RECIPE, the Apple will ask for the name of the recipe. The name is limited to 40 characters as indicated by the dots. However, any character is legal (the input routine will be discussed later). Note that the back arrow can be used to backspace over characters; however, the forward arrow will not work. You must retype information you wish to copy. The program asks for the number of servings and then presents a list of categories for the table of contents. Select the category that best fits the recipe.

Next the computer asks for the ingredients. You may type in any number of ingredients and you can terminate the list by typing <RETURN>. The ingredients are limited to 35 characters and any character is allowable.

ENTERING RECIPES

In order to make the doubling and tripling of the recipe work properly the quantities must be entered according to specific rules:

First, the quantity must be entered first; i.e.

1 LARGE ONION.

Second, fractions must be entered thusly:

1/2 TEASPOON MACE.

Third, whole numbers with a fraction are entered with a dash between the whole number and the fraction: 1-1/2 LBS. GROUND BEEF.

Last, variable quantities are entered with a dash between the numbers: 6-8 MEDIUM APPLES.

The important thing to remember is that the quantity must be entered with no spaces. Use a dash for a separator within the quantity, or else the doubling routine will not work properly. The quantities, however, should be separated from the rest of the ingredient description by a space. The maximum quantity allowable is 999 units; otherwise the computer assumes you are feeding an army (line 90).

Correct: 1-1/2 cups water

Incorrect: 1 1/2 cups water

Incorrect: 1-cup flour (A space is required after the 1)

By the way, if there is no quantity associated with the ingredient, that's OK since the multiply routine will ignore it. An example would be: SALT TO TASTE.

PREPARATION INSTRUCTIONS

Once you have approved the ingredients, the computer will ask for the PREPARATION INSTRUCTIONS. Here you may type to your heart's content (up to 16 lines). Any character is legal and the backspace arrow works as it did in the ingredients section. Don't worry about the end of the screen - just keep typing as if it weren't there. The entry routine will break the words at the screen boundary. When you are finished with the instructions, press the RETURN key and the computer will pause while it rearranges the instructions in memory. If you choose to change the instructions, you must retype them. Once you are satisfied, the recipe will be saved to the disk and the main menu will again be displayed.

CHANGING A RECIPE

If you choose 2=CHANGE A RECIPE the program will ask for the page number of the recipe, which can be found by selecting option 3=PRINT INDEX. If you inadvertently type the wrong page number, you will be informed that the page number doesn't exist and you will be returned to the main menu. Typing either the <ESC> or <RETURN> key will also return you to the main menu. If you have typed a proper page number, you will be given the following choices:

DO YOU WISH TO CHANGE

1. NAME OF RECIPE
2. NUMBER OF SERVINGS
3. CATEGORY
4. INGREDIENTS
5. PREPARATION INSTRUCTIONS
6. DELETE A RECIPE

WHICH=

Each of these options will give you the chance to make changes to recipes previously entered. If you choose: DELETE A RECIPE, you will be given the chance to change your mind just in case that's not the recipe you want to delete.

PRINTING THE INDEX

Choice 3=PRINT INDEX OF RECIPES will give you the option to print the index of all the recipes as they appear in order on the disk or print the recipes only for a specific category. The option is given to list the index to the printer or to the screen. If you choose to list to the screen, the listing will stop as the screen fills. At this time, you may select a page number to be printed or just hit the <RETURN> key to continue viewing the index. If you choose to see a page number, it is the same as selecting option 4=PRINT A RECIPE.

PRINTING/SIZING THE RECIPE

Option 4=PRINT A RECIPE allows you to print a recipe to the screen or to the line printer. The screen clears and you are given the following choices:

1. RECIPE SIZE AS LISTED
2. RECIPE DOUBLED
3. RECIPE TRIPLED
4. RECIPE CUT IN HALF

WHICH?

Then you are asked if you want the listing to go to the screen or to the printer. If the ingredients have been entered properly, the quantities in the ingredients will be changed to show your size choice. Also the number of servings will be changed to reflect the recipe size. Once the listing is completed you are returned to the main menu.

Option choice 5=EXIT SYSTEM clears the screen, leaves a message for your bride and returns to Applesoft.

PROGRAM DESCRIPTION

For those of you who are interested, here is a program description so that you can make your own improvements and modifications. Use caution!! If your wife catches you making changes and improving the program, you may be forced into involuntary servitude and she'll have you writing other programs for her. You may even lose your computer to your wife altogether!!

There were several problems in the development of this program. First was the question of input. Recipes are notorious for all kinds of 'illegal' Applesoft characters that cannot be entered using normal INPUT statements. Nor can these characters be read from the disk file.

Secondly, lists of ingredients and preparation instructions vary greatly in length, and with my limited programming experience, I had to find a way to enter these lists without using up a lot of disk space. (I was moderately successful.)

Thirdly, I had to find a way to parse out the quantities from the ingredients so that the recipes could be doubled, etc.

continued on next page

Apple Recipe Box (Cont.)

KEYBOARD INPUT ROUTINES

Two separate input routines were written to take care of the 'illegal' characters in Applesoft. The **standard data entry routine** is found at lines 150-178. This routine is used for almost all data entry except for the preparation instructions. The number of characters typed is counted by **I**, which is initialized to zero. The number of characters allowed is stored in **NC**. The bell is rung and data input begins by **GETting** a character and storing it in array **JS(I)**. If the **<ESC>** key is pressed (**CHR\$(27)**), the **GOSUB** return is **POPPed** off and control is returned to the main menu.

The **backspace (CHR\$(21))** is supported but the **forward arrow (CHR\$(8))** is not. When the **<RETURN>** key is pressed, **IS** is loaded with the individual characters from **JS(I)**. **IS** is returned as the response string and **I** holds the character count.

The **other keyboard entry routine** is found at lines 100-142. It is used exclusively by the **preparation instructions** routine. A method was needed to input a large number of characters unbroken by a carriage return, so we used the text screen buffer. The address of each line (0-23) on the screen is computed and stored in array **L%()** by the initialization routine at lines 9290-9320. Line 102 **GETs** a character into **JS(CL)**, where **CL** is the count of the characters on an input line. Lines 104-112 handle the backspace arrow and line 114 terminates the input when a carriage return is read. Line 116 disallows all other control characters. Line 118 prints the character and checks to see if it is a space (**CHR\$(32)**). If it is a space, the character count is stored in **LS(NW%)**. Line 120 checks to see if we are at the right edge of the screen. If we are, we back up to the last space written (**LS(NW%)**) with an **HTAB** and clear to the end of the line (line 122). Lines 124-132 print the leftovers from the previous line and check for the last line (maximum of 16).

Once the input is complete, lines 134-142 read directly from screen memory and build the preparation instructions line by line in array **WDS()**. Upon exit, **NW%** contains the number of lines stored in **WDS**.

DISK INPUT ROUTINES

The disk input routines, with the exception of the initialization routines, are found in lines 700 to 875. Standard input statements are used for most variables with three exceptions: line 720 inputs the name of the recipe-**NR\$()**; line 815 inputs the ingredients-**IG\$()**; and line 865 inputs the preparation instructions-**WDS()**. Since illegal Applesoft characters were allowed to be typed in from the keyboard, now normal input statements will not read these fields from the disk.

A machine language routine, which uses advanced system techniques that were described on page 120 of **NIBBLE EXPRESS-VOL 1**, was **POKEd** into decimal location 800 (\$0320). **NXTCHAR** at location \$FD75 is called and it reads one field from the disk and stores it at \$0200. The **X register** contains the number of characters read +1 (which is the carriage return). Notice that variable **CV\$** is initialized in line 10 and it is the first string variable defined in the program. By doing this we have established **CV\$** as the first variable in Applesoft's variable table and the pointer at \$69 and \$6A point to this variable. The first

two bytes, starting at \$69, contain the variable name in ASCII code, and the third byte contains the length of the string. So we transfer the **X-register** to the **A-register** and store the string length in \$69 + 2. The fourth and fifth bytes of a string variable contain a pointer to the start of the actual string, so we change that pointer to point to the **NXTCHAR** buffer at location \$200.

Essentially what we have done is to change all the pointers for variable CV\$ to point at the NXTCHAR input buffer. Since **NXTCHAR** doesn't care what characters are input we are free to read commas, colons or anything else. The only drawback is that **NXTCHAR** sets the high bit in each ASCII character, and the machine language routine quickly strips off that bit with **AND #7F**. Control is then returned to the BASIC program.

The program variable is then loaded into an array by using **XS=MIDS(CV\$,1)**. This must be done immediately, as the next read or keyboard input will destroy the information in the **NXTCHAR** buffer.

This routine doesn't appear to slow down disk read time and can be useful in any application where you want to read 'illegal' characters from the disk. **Make sure, however, that the input string variable is defined first in the program.** An assembly language listing of this routine is included for information only, as it is put into memory by the **POKEs** in lines 9160-9170.

DISK FILE STRUCTURE

There are three disk files used for data storage: **INDEX.FILE**, **L60**, **RECIPE.FILE**, and **OPEN.RECORDS**. Each of these is a random access file whose respective length is defined by **L\$="L60"**; **LR\$="L45"** and **LO\$="L14"**. Record 0 of the index file contains two important pointers: First **N** is the number of recipes in the index file. Second, **NX%** points to the end of the **RECIPE.FILE** + 1. The remainder of the records in the index file contain the following information:

1. **TR%()=type of recipe (1 to 9)**
2. **NR\$()=name of the recipe**
3. **NS%=number of servings**
4. **PI%=pointer to the start of the ingredients**
5. **PP%=pointer to the start of the preparation instructions.**

The **RECIPE.FILE** contains all of the ingredients and preparation instructions in records that are 45 characters in length. Each record contains two fields: First, **NR**, which is a pointer to the next record, and second, the ingredient or instruction itself. This way the records are chained together until **NR=0**, which then indicates the last record of this set of ingredients or instructions. For example, **PI% = 4** points to record 4 of the **RECIPE.FILE**. This record will contain the first ingredient of the recipe. Record 4 may look like this:

5, 1/2 CUP SUGAR

This indicates that the next ingredient may be found at record 5 and the first ingredient is 1/2 CUP SUGAR. Record 5 may look like this:

0, 1 TSP. MACE

The zero indicates the last ingredient of this recipe. The preparation instructions are stored on the same file in the same fashion,

where **PP%** points to the first record of the instructions.

The **OPEN.RECORDS** file keeps track of the number of records that may have been previously deleted and are lying unused. Record 0 of this file is the count of these records, and each subsequent record of this file points to an empty record in the **RECIPE.FILE**. If the first record of this file is zero, then there are no abandoned records and the next data is written to the **RECIPE.FILE** starting at record **NX%**. The write routines at lines 600 and 650 make heavy use of these files.

RECIPE SUBROUTINES

The rest of the program is pretty much straightforward file manipulation.

Lines 30-70 contain the main menu which drives the rest of the program.

Lines 75-90 check for ingredient quantities greater than 999.

Lines 200-210 and 225-240 are the one and two line title subroutines used in screen and printer presentation. Liberal use is made of "text windowing" by **POKEing** memory location 34 (the top line of the text window). These routines center and print a title **TL\$()** (the second line in 225) on the screen (**PR=0**) or on the 80 column printer (**PR=1**).

Lines 300-315 print a prompt, **PM\$**, and then **GET** a number greater than zero but not greater than **UL**. The choice is returned in **CH**. This is used primarily by menu choices. See line 60.

The printer setup is in lines 325-335. It is currently set up for an Apple Parallel interface in slot 1 connected to an IDS-445 Paper Tiger. You can check the printer manual for your printer and substitute any code changes that are appropriate, but this setup should handle most of the popular printers as it stands.

Screen scrolling is controlled by the routine at 350-370. Two entries are used, depending on what message needs to be printed. It can be entered at either 350 or 355. The **<ESC>** key is checked and if it is pressed control is returned to the main menu at line 30. **C** is used as a screen line count in the print routines.

Lines 410 and 420 are error messages pertaining to data stored in the index file.

Disk write routines are in lines 400-690. See the variable table for an explanation of the variables. Also since I'm very lazy, string variables are used for DOS commands. For example, **OP\$** stands for **CTRL-D,OPEN**.

At lines 900 and 950 are the routines that delete the ingredients and preparation instructions by placing the deleted record number in the **OPEN.RECORDS** file.

THE MAIN PROGRAM

The program starts by defining **CV\$** so that it may be used in the special input routines. A jump is made to line 9000 which is the initialization routine. Here, the variables are dimensioned and the data statements are read. If you wish to change any recipe categories, it may be done in lines 9130 and 9140. If you wish to add more categories, make sure you change the number of headings, **NH**, in line 9100.

Next, an attempt is made to **RENAME** the index file with the same name. If there is such a file on the disk, there is no problem and the program continues by reading the names of the recipes, **NR\$()**, and the type of recipes, **TR%()**, into memory. If this is the first time the

program has been run a **FILE NOT FOUND** message (**error# 6**) will be generated and control is passed to the routine at **line 7000**. If, in fact, the index file was not found, the error routine will create a **new index** file and pass control to the main menu at **line 30**.

The only other error that is treated in the error routine is a **DISK FULL**. When this happens you are informed that there is no more room on the disk and you must initialize a new disk. All other errors are treated as non-recoverable and the program ends. You may want to add a more sophisticated error handling routine at **line 7000**.

FIRST TIME RECIPES

Recipes are entered for the first time at **lines 1000-1290**. This routine is straightforward and the variables used can be found in the attached variable table. Notice that ample opportunity is given to the user to correct errors at input time by asking after each entry if the input is correct. This is to prevent making massive changes later.

If errors are made during input, a chance to correct the recipes is given in **lines 2000-2960**.

Lines 3000-3570 allow the user to print either a **list of recipes by page number or by category**. Each choice allows **output to the printer or to the screen (GOSUB 375)**. If output is sent to the screen, the index stops as the screen fills and the user can press the **<RETURN>** key to continue the list (**lines 3150 and 3480**), or he may elect to print a recipe by selecting the page number.

If the option is chosen to **print out a recipe**, control is transferred to the routine at **line 4500**. Recipe size is selected by **lines 4550-**

4560 and is stored in **R3**. The printer option is called for in **line 4570** and then the information is read from the disk. If the recipe size is other than "as listed" (**RS=1**) then a jump is made to **line 6000** to multiply the ingredients; otherwise, **lines 4610-4770** neatly print out the recipe for the cook to take to the kitchen.

The routine at **lines 4000-4030** merely sets a page number and jumps to the print routine at **line 4500**.

Line 5000 ends the program with a short message to the cook as the screen clears.

SIZING RECIPES

The multiply routine for the ingredients is found in **lines 6000-6260**. A routine to **convert the fractions** is at **lines 6500-6590**.

First the number of servings (**NS%**) is changed by the value of **RS**. Remember that **RS=4** means to cut the recipe in half. **Line 6060** checks to see if the first character in the ingredient is a number. If it is not a number there is no sense multiplying it, so we get the next ingredient. (It may be something like **DASH OF SALT**.)

Lines 6080-6110 then search character by character until a **space** is found. **NN%** is the number of characters and each character is stored in array **CS()**. **Lines 6110-6140** **evaluate the number**. If there is only one number, it is assumed to be a single whole number. The **denominator (DE%)** is assigned value = **1** and the **numerator (NU%)** is the number itself (**VAL (CS(1))**). If there are two characters, the only possibility is a whole number greater than 10. This is handled by **line 6130**.

If there are only three characters and the second one is a **'/'**, it is assumed to be a **simple**

fraction, e.g., **1/2**. **Line 6140** takes care of this.

If there are only three characters and the second is not a **'/'** then it is assumed to be of the form **6-8**, as in **6-8 LARGE APPLES**.

Each number is separately multiplied by **lines 6160-6170**. **Lines 6180-6240** parse out complex fractions like **2-1/2** and convert the whole number (**WN\$**) into fractional form. Thus, **2-1/2 becomes 5/2**, when **NU%=5** and **DE%=2**.

All of the **fraction multiplication** takes place in **lines 6500-6590**. If the recipe size is tripled or doubled then numerator (**NU%**) is multiplied by the appropriate number (**line 6520**). If the recipe is to be cut in half, the denominator is multiplied by two (**line 6510**). The fraction is then reduced by **line 6530**. Then, since recipes rarely have unusual fractions like **3/7**, a very simple fraction reduction method is used. The unreduced fraction is compared to those in array **RE\$()** and if a match is found, the reduced equivalent fraction is found in array **FA\$()** (**line 6550**). The new quantity is then loaded into **NM\$** and it is exchanged for the old quantity in **line 6240**. Strange, but it works!!

CONCLUSION

This was an enjoyable program to write as it endeared my wife a little bit more to our Apple. It helped give her a better understanding of the capabilities of the machine and helped me develop a few more programming skills. Next on the list is the refinement of a shopping list program tailored to her favorite grocery store. It's hard to tell where it will end. Who knows? Someday we may have His and Hers Apples!!

continued on next page

Recipe Box

by J.L. Costenbader
Copyright © 1982
by Micro SPARC Inc.

```

10 CV% = "A"
20 GOTO 9000
30 PRINT : PRINT CL$: PRINT D$;"PR#0": POKE 34,0: HOME
   : PRINT TAB(15);"RECIPE BOX": FOR X = 1 TO 40: PRINT
   : "="; NEXT : PRINT : PR = 0
40 POKE 32,4: PRINT : PRINT "1=ENTER A NEW RECIPE": PRINT
   : "2=CHANGE A RECIPE ": PRINT "3=PRINT INDEX OF RECIP
   : ES"
50 PRINT "4=PRINT A RECIPE": PRINT "5=EXIT SYSTEM"
60 POKE 32,0: VTAB 16:PM$ = "FUNCTION CHOICE: ":UL = 5
   : GOSUB 300
70 ON CH GOTO 1000,2000,3000,4000,5000
75 ND = 0:CK = 1
80 CK$ = MID$(I$,CK,1): IF CK$ < "0" OR CK$ > "9" THEN
   : 90
85 CK = CK + 1:ND = ND + 1: GOTO 80
90 IF ND > 3 THEN CALL BL: PRINT "IT LOOKS AS IF YOU'R
   : E FEEDING AN ARMY": PRINT "USE A QUANTITY LESS THAN
   : 1000": PRINT
95 RETURN
100 HOME : VTAB VT: HTAB 1:CL = 1:NM% = 1:LS(1) = 1
102 GET J$(CL)
104 IF J$(CL) = CHR$(8) AND NM% = 1 AND CL = 1 THEN 1
   : 02
106 IF J$(CL) < > CHR$(8) THEN 114
108 IF CL > 1 THEN PRINT CHR$(8);CL = CL - 1: GOTO
   : 102
110 IF NM% > 1 THEN NM% = NM% - 1:VT = VT - 1:CL = LS(N
   : M%): VTAB VT: HTAB CL
112 GOTO 102
114 IF J$(CL) = CHR$(13) THEN LS(NM%) = CL: GOTO 134
116 IF ASC (J$(CL)) < 32 THEN 102
118 PRINT J$(CL); IF J$(CL) = CHR$(32) THEN LS(NM%) =
   : CL
120 CL = CL + 1: IF CL < 40 THEN 102
122 HTAB LS(NM%): CALL - 868:VT = VT + 1: VTAB VT: HTAB
   : 1

```

```

124 IF NM% = 16 THEN 134
126 FOR X = LS(NM%) + 1 TO 40: PRINT J$(X);: NEXT
128 CL = X - LS(NM%) - 1:NM% = NM% + 1:LS(NM%) = 1
130 IF NM% = 16 THEN CALL BL: INVERSE : FLASH : HTAB 1
   : 0: PRINT "LAST LINE": NORMAL : HTAB 1
132 GOTO 102
134 PRINT : PRINT : HTAB 5: PRINT "ONE MOMENT PLEASE"
136 LI = VT - NM% - 1: FOR X = 1 TO NM%:WD$(X) = "": FOR
   : X1 = 1 TO LS(X)
138 J = PEEK (L$(LI + X) + X1 - 1):J$ = CHR$(J - 128)

140 WD$(X) = WD$(X) + J$: NEXT : NEXT
142 RETURN
150 I = 0: CALL BL
152 REM : STANDARD DATA ENTRY
154 IF I < 0 THEN I = 0
156 IF NC < 6 GOTO 160
158 IF I > NC - 5 THEN CALL BL
160 GET J$(I)
162 IF J$(I) = CHR$(27) THEN POP : GOTO 30
164 IF J$(I) = CHR$(13) THEN I = I - 1: GOTO 178
166 IF J$(I) = CHR$(21) THEN 160
168 IF J$(I) = CHR$(8) AND I = 0 THEN PRINT "": GOTO
   : 154
170 IF J$(I) = CHR$(8) THEN PRINT J$(I);:I = I - 1: GOTO
   : 154
172 IF ASC (J$(I)) < 32 THEN GOTO 160
174 IF NC = 1 THEN PRINT "": GOTO 160
176 PRINT J$(I);:I = I + 1: GOTO 154
178 PRINT : PRINT :I$ = "": FOR J = 0 TO 1:I$ = I$ + J$
   : (J): NEXT : RETURN
200 POKE 34,0: HOME :TB = 20 * (PR + 1) - INT ( LEN (T
   : L$) / 2): IF TB > 0 THEN PRINT TAB( TB);
205 PRINT TL$
210 FOR X = 1 TO 40 * (PR + 1): PRINT "="; NEXT : PRINT
   : IF PR = 0 THEN POKE 34,3
215 RETURN
225 POKE 34,0: HOME :TB = 20 * (PR + 1) - INT ( LEN (T
   : L$) / 2): IF TB > 0 THEN PRINT TAB( TB);
230 PRINT TL$
235 TB = 20 * (PR + 1) - INT ( LEN (T1$) / 2): IF TB >
   : 0 THEN PRINT TAB( TB);

```

continued on next page

Apple Recipe Box (Cont.)

APPLE RECIPE BOX VARIABLE TABLE

BL	Address of monitor routine to beep speaker
C	Screen line count for screen print routines
CS	Characters parsed from ingredients to form the quantities
CC%	Character count in preparation instructions
CH	Choice number selected from subroutine at 300
CL\$	DOS command to close file
CS	Address of monitor routine to clear to end of screen
CV\$	Return string from machine disk read (CALL 800)
D\$	CHR\$(4) (CTRL - D)
DE%	Denominator in fractions for multiply routine
DH	Number of characters to the first dash in the quantity

RES()	Fractions that will need to be reduced
FA\$()	Fractional equivalents of the above in reduced form
HD\$()	The categories available for recipe index
NH	Number of headings (HD\$) available
HT	HTAB position
VT	VTAB position
IS	Character string returned from input routine
IG\$	Ingredients
NI%	Number of ingredients
IX\$	"INDEX.FILE.L60"
L\$	Record length of the index file
LF\$	Record length of the recipe file
LO\$	Record length of open records file
N	Number of recipes stored on disk
NR\$()	Name of the recipe
NS%	Number of servings
NU%	Numerator for multiply routine
NX%	Pointer to the last record on the recipe file + 1

OP\$	DOS command to open a file
PI%	Pointer to the first record of the ingredients
PM\$	Prompt string for subroutine 300
PN	Page number of currently active recipe
PP%	Pointer to the first record of the preparation instructions
PR	PR=0 - output to screen. PR=1 - output to printer
RD\$	DOS command to read disk
RS	Multiplication factor to increase recipe
TL\$,T1\$	Titles to be printed at top of page
TR%()	Type of recipe (1 to NH)
UL	Max number of choices in subroutine 300
WD\$()	Fields containing preparation instructions
NW%	Number of fields necessary to hold preparation instructions
WR\$	DOS command to write to disk

The remaining variables are work variables and are used only within particular sub-routines.



```

240 PRINT T1$
245 FOR X = 1 TO 40 * (PR + 1): PRINT "="; NEXT : PRINT
: IF PR = 0 THEN POKE 34,4
250 RETURN
300 PRINT PM$;
305 GET CH$: IF CH$ = CHR$(29) THEN POP : GOTO 30
310 CH = VAL (CH$); IF CH < 1 OR CH > UL THEN 305
315 PRINT : RETURN
325 PRINT D$;"PR#";PR: PRINT CHR$(29)
330 IF PR > 0 THEN PRINT CHR$(9);"80N"
335 RETURN
350 VTAB 22: HTAB 1: PRINT "HIT 'RETURN' TO CONTINUE..."
:;: GOTO 360
355 VTAB 22: PRINT "HIT 'RETURN' FOR MORE...";
360 GET CH$: IF CH$ = CHR$(27) THEN POP : GOTO 30
365 IF ASC (CH$) < > 13 THEN 360
370 PRINT : HOME : C = 0: RETURN
375 HOME : PRINT TAB(13);"OUTPUT CHOICES": PRINT : PRINT
: 1=OUTPUT TO SCREEN": PRINT " 2=OUTPUT TO
: PRINTER"
380 VTAB 15:PM$ = "OUTPUT CHOICE: ":UL = 2: GOSUB 300
385 PR = CH - 1: RETURN
410 HOME : VTAB 8: PRINT "THERE ARE NO RECIPES IN THE
FILE": GOSUB 350: RETURN
420 HOME : VTAB 8: PRINT "THAT PAGE NUMBER IN NOT IN CO
MPUTER": GOSUB 350: RETURN
450 PRINT OP$;IX$;L$
460 PRINT WR$;IX$;"R";PN: PRINT TR$(PN): PRINT NR$(PN)

470 PRINT NS$: PRINT PI$: PRINT PP%
480 PRINT CL$: RETURN
500 PRINT OP$;IX$;L$
510 PRINT WR$;IX$;"R"
515 IF N = 0 THEN N = 1
520 PRINT N: PRINT NDC
530 PRINT CL$;IX$: RETURN
550 PRINT OP$;IX$;L$
560 PRINT RD$;IX$;"R"
570 INPUT N: INPUT NDC
580 PRINT CL$;IX$
590 RETURN
600 PRINT OP$;RF$;LF$
602 PRINT OP$;RD$;LO$
604 PRINT RD$;RD$;"R"
606 INPUT RN:NR = NDC:PI% = NDC
608 FOR X = 1 TO NI%:OP = NR
610 IF RN = 0 THEN NR = OP + 1: GOTO 624
612 PRINT RD$;RD$;"R";RN
614 INPUT OP:RN = RN - 1: IF X = 1 THEN PI% = OP
616 IF X = NI% THEN NR = 0: GOTO 626

```

```

618 IF RN = 0 THEN NR = NDC: GOTO 626
620 PRINT RD$;RD$;"R";RN
622 INPUT NR
624 IF X = NI% THEN NR = 0
626 PRINT WR$;RF$;"R";OP
628 PRINT NR
630 PRINT IG$(X): NEXT
632 PRINT WR$;RD$;"R"
634 PRINT RN
636 PRINT CL$
638 IF OP > = NDC THEN NDC = OP + 1
640 RETURN
650 PRINT OP$;RF$;LF$
652 PRINT OP$;RD$;LO$
654 PRINT RD$;RD$;"R"
656 INPUT RN:NR = NDC:PP% = NDC
658 FOR X = 1 TO NW%:OP = NR
660 IF RN = 0 THEN NR = OP + 1: GOTO 674
662 PRINT RD$;RD$;"R";RN
664 INPUT OP:RN = RN - 1: IF X = 1 THEN PP% = OP
666 IF X = NW% THEN NR = 0: GOTO 676
668 IF RN = 0 THEN NR = NDC: GOTO 676
670 PRINT RD$;RD$;"R";RN
672 INPUT NR
674 IF X = NW% THEN NR = 0
676 PRINT WR$;RF$;"R";OP
678 PRINT NR
680 PRINT WD$(X): NEXT
682 PRINT WR$;RD$;"R"
684 PRINT RN
686 PRINT CL$
688 IF OP > = NDC THEN NDC = OP + 1
690 RETURN
700 PRINT OP$;IX$;L$
710 PRINT RD$;IX$;"R";PN
720 INPUT TR$(PN): CALL 800:NR$(PN) = MID$(CV$,1)
730 INPUT NS$: INPUT PI$: INPUT PP%
740 PRINT CL$;IX$: RETURN
800 PRINT OP$;RF$;LF$:NI% = 1:RN = PI%
805 PRINT RD$;RF$;"R";RN
810 RI(NI%) = RN
815 INPUT RN: CALL 800:IG$(NI%) = MID$(CV$,1)
820 IF RN THEN NI% = NI% + 1: GOTO 805
825 PRINT CL$;RF$: RETURN
850 PRINT OP$;RF$;LF$:NW% = 1:RN = PP%
855 PRINT RD$;RF$;"R";RN
860 RD(NW%) = RN
865 INPUT RN: CALL 800:WD$(NW%) = MID$(CV$,1)
870 IF RN THEN NW% = NW% + 1: GOTO 855
875 PRINT CL$;RF$: RETURN

```



```

900 PRINT OP$;R0$;L0$
905 PRINT RD$;R0$;"R"
910 INPUT RN
915 FOR X = 1 TO WS%
920 PRINT WR$;R0$;"R";X + RN
925 PRINT RD(X): NEXT
930 PRINT WR$;R0$;"R"
935 PRINT RN + WS%
940 PRINT CL$;R0$: RETURN
950 PRINT OP$;R0$;L0$
955 PRINT RD$;R0$;"R"
960 INPUT RN
965 FOR X = 1 TO N1%
970 PRINT WR$;R0$;"R";X + RN
975 PRINT RI(X): NEXT
980 PRINT WR$;R0$;"R"
985 PRINT RN + N1%
990 PRINT CL$;R0$: RETURN
1000 TL$ = "ENTER RECIPES": GOSUB 200
1010 PRINT : PRINT "WHAT IS NAME OF RECIPE ": PRINT :N
C = 40: VTAB 8: HTAB 1: PRINT LEFT$(DT$,NC): HTAB
1: VTAB 8: GOSUB 150:NA$ = I$
1020 TL$ = NA$: GOSUB 200
1030 POKE 32,4: PRINT : FOR X = 1 TO NH: PRINT X;". ";H
D$(X): NEXT : POKE 32,0: PRINT
1040 HTAB 1: VTAB NH + 7: CALL CS:PM$ = "WHAT TYPE ":UL
= NH: GOSUB 300:T = CH
1050 HOME : PRINT "HOW MANY SERVINGS? ":NC = 2: GOSUB
150:NS% = VAL (I$)
1060 HOME : PRINT : PRINT "CATEGORY=";HD$(T): PRINT : PRINT
"THIS WILL SERVE ";NS%
1070 PRINT : PRINT "IS THIS CORRECT? ":GET I$: IF I$ <
> "Y" THEN 1000
1080 TI$ = "INGREDIENTS": GOSUB 225
1090 N1% = 1:NC = 35
1100 PRINT "ENTER INGREDIENTS ('RETURN' TO QUIT)": PRINT
1110 PRINT N1%;") ": HTAB 4: PRINT LEFT$(DT$,NC): CALL
- 998: HTAB 4: GOSUB 150: IF I$ = CHR$(13) GOTO
1140
1120 GOSUB 75: IF ND > 3 THEN 1110
1130 IG$(N1%) = I$:N1% = N1% + 1:VT = VT + 2: GOTO 1110
1140 N1% = N1% - 1: HOME : IF N1% = 0 THEN 1090
1150 FOR X = 1 TO N1%: PRINT X;") ":IG$(X): NEXT
1160 PRINT : PRINT "IS THIS CORRECT ":GET I$: IF I$ =
"Y" THEN 1210
1170 PRINT : PRINT :PM$ = "WHICH INGREDIENT IS INCORREC
T ":UL = N1%: GOSUB 300:X = INT (CH)
1180 HOME : PRINT "CHANGE": PRINT : PRINT X;". ":IG$(X
): PRINT : PRINT "TO": PRINT
1190 PRINT X;". ":GOSUB 150: GOSUB 75: IF ND > 3 THEN
1190
1200 IG$(X) = I$: HOME : GOTO 1150
1210 TI$ = "PREPARATION INSTRUCTIONS": GOSUB 225
1220 HT = 1:VT = 5: POKE 34,4: GOSUB 100
1230 HOME : FOR X = 1 TO NM%: PRINT WD$(X): NEXT
1240 VTAB 22: HTAB 1: PRINT "IF THIS IS CORRECT HIT 'RE
TURN'":GET I$: IF I$ = CHR$(13) THEN GOTO 1260
1250 GOTO 1220
1260 PRINT :PN = 0: FOR X = 1 TO N: IF TR$(X) = 0 THEN
PN = X
1270 NEXT : IF PN = 0 THEN N = N + 1:PN = N
1280 NR$(PN) = NA$:TR$(PN) = T: HOME : PRINT : PRINT "WR
ITING TO DISK": POKE 34,0
1290 GOSUB 600: GOSUB 650: GOSUB 450
1300 GOSUB 500
1310 GOTO 30
2000 TL$ = "CHANGE AN ENTRY": GOSUB 200
2005 IF N < 1 THEN GOSUB 410: GOTO 30
2010 IF I$ = CHR$(27) THEN 30
2015 VTAB 6: PRINT "WHAT PAGE NO. DO YOU WISH TO CHANGE
? ":NC = 3: GOSUB 150
2020 PN = VAL (I$): IF PN = 0 GOTO 30
2025 IF PN > N THEN GOSUB 420: GOTO 2015
2030 IF PN = 0 THEN GOTO 30
2035 GOSUB 700:TL$ = "CHANGE":TI$ = NR$(PN): GOSUB 225
2040 PRINT : PRINT "DO YOU WISH TO CHANGE": POKE 32,4:
PRINT : PRINT "1. NAME OF RECIPE ": PRINT "2. NUMB
ER OF SERVINGS": PRINT "3. CATEGORY"
2045 PRINT "4. INGREDIENTS": PRINT "5. PREPARATION INST
RUCTIONS"
2050 PRINT "6. DELETE RECIPE": PRINT : POKE 32,0
2055 PM$ = "WHICH":UL = 6: GOSUB 300:C1 = CH
2060 ON C1 GOTO 2100,2200,2300,2400,2800,2900
2100 HOME : PRINT "CHANGE NAME FROM ": PRINT : PRINT "
":NR$(PN)
2110 PRINT : PRINT "TO": PRINT : PRINT " ":NC =
40: GOSUB 150:NR$(PN) = I$

```

```

2120 HOME : PRINT "NAME: ";NR$(PN): PRINT "SERVES ";NS%
: PRINT "CATEGORY=";HD$(TR$(PN))
2130 PRINT : PRINT : PRINT "IS THIS CORRECT ? ":GET
I$: IF I$ = "N" THEN GOTO 2060
2140 PRINT : GOSUB 450: GOTO 30
2200 HOME : PRINT "CHANGE NUMBER OF SERVINGS": PRINT : PRINT
" FROM: ";NS%: PRINT
2210 PRINT " TO: ":NC = 2: GOSUB 150:NS% = VAL (
I$): GOTO 2120
2300 HOME : PRINT : FOR X = 1 TO NH: PRINT X;") ":HD$(X
): NEXT : PRINT
2310 PRINT "CHANGE HEADING FROM NO. ";TR$(PN): PRINT : PRINT
" TO: ":UL = NH: GOSUB 305:TR$(PN
) = CH
2320 GOTO 2120
2400 HOME : PRINT "DO YOU WISH TO ": PRINT " 1. CHA
NGE AN INGREDIENT": PRINT " 2. ADD AN INGREDIEN
T"
2410 PRINT :PM$ = "WHICH? ":UL = 2: GOSUB 300: GOSUB 80
0:N1% = N1%: IF CH = 2 THEN 2590
2420 C = 0
2430 HOME : FOR X = 1 TO N1%: PRINT X;". ":IG$(X):C = C
+ 1: IF C > 16 THEN GOSUB 355
2440 NEXT
2450 PRINT : PRINT "CHANGE WHICH INGREDIENT? ":NC = 2:
GOSUB 150:X = VAL (I$)
2460 IF X < 1 OR X > N1% THEN 2450
2470 HOME : PRINT "CHANGE INGREDIENT ";X;" FROM: ": PRINT
: PRINT IG$(X)
2480 PRINT "TO: ": PRINT :NC = 35: GOSUB 150: GOSUB 75:
IF ND > 3 THEN 2400
2490 IG$(X) = I$
2500 C = 0: HOME : FOR X = 1 TO N1%: PRINT X;") ":IG$(X)
:C = C + 1: IF C > 15 THEN GOSUB 355
2510 NEXT
2520 VTAB 22: PRINT "IS THIS CORRECT? ":GET I$: PRINT
: IF I$ < > "Y" THEN 2430
2530 PRINT OP$;RF$;LF$;RI(N1% + 1) = 0
2540 FOR X = 1 TO N1%
2550 PRINT WR$;RF$;"R";RI(X)
2560 PRINT RI(X + 1)
2570 PRINT IG$(X): NEXT
2580 PRINT CL$;RF$: GOTO 30
2590 HOME : PRINT "LAST INGREDIENT WAS:"
2600 PRINT N1%;") ":IG$(N1%): PRINT
2610 N1% = N1% + 1
2620 PRINT N1%;") ":NC = 35: GOSUB 150: GOSUB 75: IF N
D > 3 THEN 2620
2630 IF I$ = CHR$(13) THEN 2650
2640 IG$(N1%) = I$: GOTO 2610
2650 N1% = N1% - 1:C = 0: HOME
2660 FOR X = 1 TO N1%: PRINT X;") ":IG$(X):C = C + 1: IF
C > 15 THEN GOSUB 355
2670 NEXT
2680 VTAB 22: PRINT "IS THIS CORRECT? ":GET I$: PRINT
: IF I$ < > "Y" THEN N1% = N1%: GOTO 2590
2690 GOSUB 950: GOSUB 600: GOSUB 450
2700 GOSUB 500: GOTO 30
2800 GOSUB 850
2810 TL$ = "REENTER PREPARATION INSTRUCTIONS"
2820 TI$ = NR$(PN): GOSUB 225
2830 HT = 1:VT = 5: GOSUB 100
2840 HOME : FOR X = 1 TO NM%: PRINT WD$(X): NEXT
2850 VTAB 22: HTAB 1: PRINT "IF THIS IS CORRECT HIT 'RE
TURN' ":GET I$: PRINT : IF I$ < > CHR$(13) THEN
GOTO 2800
2860 GOSUB 900: GOSUB 650: GOSUB 450
2870 GOSUB 500
2880 GOTO 30
2900 HOME : PRINT : PRINT "ARE YOU SURE YOU WANT TO DEL
ETE ": PRINT
2910 PRINT NR$(PN);"...?":GET I$
2920 PRINT I$: IF I$ < > "Y" THEN GOTO 2000
2930 GOSUB 800: GOSUB 850:N1% = N1%:WS% = NM%
2940 GOSUB 900: GOSUB 950
2950 NR$(PN) = "DELETED":NS% = 0:TR$(PN) = 0
2960 GOSUB 450: GOTO 30
3000 TL$ = "INDEX": GOSUB 200
3010 IF N < 1 THEN GOSUB 410: GOTO 30
3020 PRINT : PRINT "INDEX SELECTIONS": POKE 32,5: PRINT
3030 PRINT "1= LIST ALL RECIPES BY PAGE NUMBER": PRINT
"2= LIST RECIPES BY CATEGORY"
3040 PRINT "3= RETURN TO MAIN MENU": POKE 32,0: PRINT :
PRINT
3050 PM$ = "INDEX CHOICE":UL = 3: GOSUB 300:C1 = CH

```

continued on next page

Apple Recipe Box (Cont.)

```

3060 IF CH = 3 THEN 30
3070 ON C1 GOTO 3100,3400
3080 TL$ = "INDEX": GOSUB 200
3100 GOSUB 375: GOSUB 325: GOSUB 200: C = 0
3110 FOR PN = 1 TO N: IF TR$(PN) = 0 THEN 3190
3120 PRINT PN;" "; SPC( PN < 10); SPC( PN < 100); SPC(
PN < 1000);NR$(PN)
3130 IF PR > 0 THEN 3190
3140 C = C + 1: IF C < 16 THEN 3190
3150 VTAB 22: PRINT "ENTER PAGE NUMBER DESIRED"
3160 VTAB 23: PRINT "OR HIT 'RETURN' FOR MORE: ";:NC =
3: GOSUB 150
3170 IF I$ < > CHR$(13) THEN PN = VAL(I$): GOTO 32
20
3180 C = 0: HOME: PRINT
3190 NEXT
3200 IF PR > 0 THEN 30
3210 PRINT: PRINT "PAGE NUMBER DESIRED: ";:NC = 3: GOSUB
150
3220 GOSUB 4500
3230 GOTO 30
3400 TL$ = "CATEGORY INDEX": GOSUB 200: C = 0
3410 POKE 32,4: PRINT: FOR X = 1 TO NH: PRINT X;" ";H
D$(X): NEXT: POKE 32,0: PRINT
3420 HTAB 1: VTAB NH + 7: CALL CS:PM$ = "WHICH HEADING:
":UL = NH: GOSUB 300:T = CH
3430 GOSUB 375: GOSUB 325
3440 TL$ = HD$(T): GOSUB 200
3450 FOR PN = 1 TO N: IF TR$(PN) < > T THEN GOTO 3520
3460 PRINT PN;" "; SPC( PN < 10); SPC( PN < 100); SPC(
PN < 1000);NR$(PN): IF PR > 0 THEN 3520
3470 C = C + 1: IF C < 16 THEN 3520
3480 VTAB 22: PRINT "ENTER PAGE NUMBER DESIRED"
3490 VTAB 23: PRINT "OR HIT 'RETURN' FOR MORE: ";:NC =
3: GOSUB 150
3500 IF I$ < > CHR$(13) THEN PN = VAL(I$): GOTO 35
60
3510 C = 0: HOME: PRINT
3520 NEXT
3530 IF PR > 0 THEN 30
3540 IF C = 0 THEN PRINT "THERE ARE NO RECIPES IN THIS
CATEGORY": GOSUB 350: GOTO 30
3550 PRINT: PRINT "PAGE NUMBER DESIRED: ";:NC = 3: GOSUB
150
3560 GOSUB 4500
3570 GOTO 30
4000 HOME: TL$ = "PRINT RECIPES": GOSUB 200
4010 IF N < 1 THEN GOSUB 410: GOTO 30
4020 PRINT "WHAT PAGE NUMBER? ";:NC = 3: GOSUB 150
4030 GOSUB 4500
4040 GOTO 30
4500 PN = VAL(I$): IF PN = 0 GOTO 30
4510 IF PN > N THEN GOSUB 420: GOTO 30
4520 FOR X = 1 TO 16:WD$(X) = "": NEXT
4530 IF TR$(PN) = 0 THEN GOSUB 420: POP: GOTO 30
4540 HOME: PRINT "DO YOU WANT: "; POKE 32,4: PRINT: PRINT
"1. RECIPE SIZE AS LISTED"
4550 PRINT "2. RECIPE DOUBLED": PRINT "3. RECIPE TRIPLE
D": PRINT "4. RECIPE CUT IN HALF": PRINT:UL = 4:PM
$ = "WHICH? "
4560 GOSUB 300:RS = CH: PRINT: POKE 32,0
4570 GOSUB 375: GOSUB 325
4580 GOSUB 700: GOSUB 800: GOSUB 850: C = 0
4590 IF RS = 1 THEN GOTO 4610
4600 GOSUB 6000
4610 IF PR > 0 THEN NR$(PN) = CHR$(31) + CHR$(1) +
NR$(PN) + CHR$(2) + CHR$(29)
4620 TL$ = NR$(PN):T1$ = "SERVES " + STR$(NS%): GOSUB
225
4630 FOR X = 1 TO N1%: PRINT X;" "; SPC(X < 10);IG$(X
): IF PR > 0 THEN 4650
4640 C = C + 1: IF C > 14 THEN GOSUB 355
4650 NEXT
4660 IF PR > 0 THEN GOTO 4690
4670 IF C > 11 THEN GOSUB 355: GOTO 4700
4680 PRINT: C = C + 1: GOTO 4700
4690 PRINT: TL$ = "PREPARATION INSTRUCTIONS": GOSUB 200
4700 IF PR = 0 THEN PRINT TAB(8);"PREPARATION INSTRU
CTIONS": PRINT: C = C + 2
4710 IF PR > 0 THEN 4760
4720 FOR X = 1 TO N2%: PRINT WD$(X)
4730 C = C + 1: IF C > 14 THEN GOSUB 355
4740 NEXT

```

```

4750 GOSUB 350: RETURN
4760 FOR X = 1 TO N2% STEP 2: PRINT WD$(X);" ";WD$(X +
1): NEXT
4770 IF PR > 0 THEN PRINT: PRINT: PRINT: PRINT: PRINT
: PRINT: PRINT D$;"PR#0":PR = 0: RETURN
5000 HOME: VTAB 11: HTAB 10: SPEED = 100
5010 CALL BL: PRINT "JAN LOVES NOREEN": CALL BL: SPEED =
255: CALL BL: END
6000 IF RS > 3 THEN NS% = NS% / 2: GOTO 6020
6010 NS% = NS% * RS
6020 IF NS% < 1 THEN NS% = 1
6030 FOR X = 1 TO N1%
6040 NN% = 1
6050 C$(1) = LEFT$(IG$(X),1)
6060 IF ASC(C$(1)) < 49 OR ASC(C$(1)) > 57 THEN 625
0: REM 1ST CHARACTER WAS NOT A NUMBER
6070 NN% = NN% + 1
6080 C$(NN%) = MID$(IG$(X),NN%,1)
6090 IF C$(NN%) = CHR$(32) THEN NN% = NN% - 1: GOTO 6
120
6100 NN% = NN% + 1
6110 GOTO 6080
6120 IF NN% = 1 THEN DE% = 1:NU% = VAL(C$(1)): GOTO 6
230
6130 IF NN% = 2 THEN DE% = 1:NU% = VAL(C$(1) + C$(2))
: GOTO 6230
6140 IF NN% = 3 AND C$(2) = "/" THEN DE% = VAL(C$(3))
:NU% = VAL(C$(1)): GOTO 6230
6150 IF NN% > 3 THEN 6180
6160 DE% = 1:NU% = VAL(C$(1)): GOSUB 6500
6170 T1$ = NM$:DE% = 1:NU% = VAL(C$(3)): GOSUB 6500:NH
$ = T1$ + "-" + NM$: GOTO 6240
6180 WN$ = "": FOR Y = 1 TO NN%: IF C$(Y) = "-" THEN DH =
Y
6190 NEXT
6200 IF DH > NN% THEN 6250
6210 FOR W = 1 TO DH - 1:WN$ = WN$ + C$(W): NEXT
6220 DE% = VAL(C$(DH + 3)):NU% = VAL(C$(DH + 1)) + VAL
(WN$) * DE%
6230 GOSUB 6500
6240 IG$(X) = NM$ + RIGHT$(IG$(X), LEN(IG$(X)) - NN%)
6250 NEXT
6260 RETURN
6500 WN% = 0:TF$ = ""
6510 IF RS = 4 THEN DE% = DE% * 2: GOTO 6530
6520 NU% = NU% * RS
6530 IF DE% < = NU% THEN WN% = INT(NU% / DE%):NU% =
NU% - WN% * DE%
6540 IF NU% = 0 THEN NM$ = STR$(WN%): GOTO 6590
6550 TF$ = STR$(NU%) + "/" + STR$(DE%): FOR Y = 1 TO
13: IF TF$ = RE$(Y) THEN TF$ = FA$(Y)
6560 NEXT
6570 IF WN% = 0 THEN NM$ = TF$: GOTO 6590
6580 NM$ = STR$(WN%) + "-" + TF$
6590 RETURN
7000 ET = PEEK(222):LN = PEEK(218) + PEEK(219) * 2
56: POKE 216,0
7010 IF ET = 0 OR ET > 15 THEN 7180: REM APPLESOFT ERR
OR
7020 IF ET < > 6 THEN 7120
7030 PRINT: PRINT OP$;IX$;L$
7040 PRINT WR$;IX$;"R0"
7050 PRINT N: PRINT NDC
7060 PRINT CL$;IX$
7070 PRINT OP$;RO$;LO$
7080 PRINT WR$;RO$;"R0"
7090 PRINT 0
7100 PRINT CL$;RO$
7110 TR$(1) = 1: GOTO 9290
7120 IF ET < > 9 THEN 7100
7130 CALL BL: CALL BL
7140 TEXT: HOME: VTAB 8: PRINT "THERE IS NO MORE ROOM
ON THIS DISK!!"
7150 PRINT: PRINT "INITIALIZE A NEW DISK AND CONTINUE"
7160 PRINT "ADDING RECIPES"
7170 END
7180 TEXT: HOME: VTAB 8: PRINT "ERROR NUMBER ";ET;" I
N LINE ";LN
7190 IF ET = 0 OR ET > 15 THEN PRINT "SEE APPLESOFT MA
NUAL, PAGE 81": GOTO 7210
7200 PRINT "SEE DOS MANUAL, PAGE 115"
7210 END
9000 HOME: VTAB 8: HTAB 13: PRINT "RECIPE BOX": PRINT
: HTAB 11: PRINT "JL COSTENBADER": VTAB 23: PRINT "
** COPYRIGHT 1982 BY MICROSPARC, INC **"
9010 DIM WD$(16),TR$(300),NR$(300),IG$(25),HD$(10),J$(8
0)

```


DISK DOCTOR

Apple Disk Doctor

by Ben E. Colley
Rt. #1, Box 67B
Hallsville, MO 65255

INTRODUCTION

What do you call a program that helps diagnose diskette illnesses and provide a remedy? I call this program "APPLE DISK DOCTOR", a diskette utility that is easy to use and easy to enhance.

DISK DOCTOR is a member of the ZAP family of utilities. It allows you to accurately display, edit, and then write information **directly** to the sectors of the disk that you select. You can modify Disk Directories, read and write control characters, examine hidden names, and a whole host of other things.

DISK DOCTOR is written primarily in Applesoft Basic and it has a short machine language program which it uses to interface with DOS. It requires a 48K Apple II with Applesoft in ROM or an Apple II Plus. It also requires at least one Disk II. With minor modifications, DISK DOCTOR will work with DOS 3.2.1 as well as its standard DOS 3.3 mode.

Before using DISK DOCTOR, it may help you to become familiar with the information on disk storage found on pages 123-137 of the DOS 3.3 manual.

Caution! Since DISK DOCTOR allows direct writing of disk sectors it should be tested and tried with a scratch/test diskette in order to become thoroughly familiar with it. It should **not** be used with a valuable diskette until this familiarity has been established.

As you review the listing you will probably notice the lack of GOTO statements. Disk Doctor is a very structured, modular program using lots of GOSUB and FOR...NEXT logic. Let's go through the options and see what Disk Doctor can do with your diskettes.

DISK DOCTOR DUMP/ZAP

Option #1 - Dump Sector. This section (lines 1000-1310) reads a sector and displays it on the screen, or the sector may be dumped to the printer if you have one (see line 1120 for printer modifications if you do not have your printer in slot #1, or use other than the Apple parallel printer interface). The routine displays the sector in **hexadecimal and character format** in 40 column mode for screen displays, and in 80 column mode for printed dumps.

Option #2 - Zap Sector. Lines 2000-2995 allow you to **change any byte** on the disk. The data is displayed one-half sector at a time; you may flip back and forth between the 2 pages by keying **CTRL-P**. The Zap routine recognizes **five sub-commands**, namely:

<CTRL-M> - Move mode permits you to quickly **tab right, left, up, or down** via the **I, J, K, M** keys, positioning the cursor at the display cell you wish to edit. The cursor will wrap around the screen any time you tab beyond the display bounds.

<CTRL-E> - Edit mode is where the "zapping" takes place. After positioning the cursor at the desired cell, **CTRL-E** will move the cursor into the data cell. You may then key in the necessary hexadecimal digits directly over the current data. Each sector byte may be changed a nibble (half a byte) at a time. Disk Doctor calls a machine language routine (described later) to perform a nibble zap. As you would expect, the display is updated to

reflect the new sector data in hex and character.

You may key in any valid hex digit (**0-F**) or use the right and left arrows to skip over data you do not wish to change. This mode also wraps around the screen when you backspace, forward space, or key past display bounds.

<CTRL-P> - Display alternate half of sector.

<CTRL-W> - Write the sector back to disk and return to MENU.

<CTRL-X> - Return to MENU without writing sector.

As is probably apparent, you will spend most of your time in either MOVE or EDIT mode. At any time, you may switch to another mode, or flip display pages and remain in the current mode.

RECOVER DELETED FILES

Option #3 - Recover File. Because we all make occasional mistakes and delete files that should not be deleted, **option 3 (lines 3000-3550)** patches up the directory entry for the deleted file. Like most file recovery programs, the VTOC (Volume Table of Contents) bit map is NOT changed; you must still load and save the file in question.

This means that when you have used DISK DOCTOR to recover a deleted file, the very first thing you should do is to **LOAD** the file and then **SAVE** it to the disk again. This sets up the VTOC to reserve the proper sectors to contain the file.

Also, please be cautioned that a deleted file must be recovered **before any other data or programs are written to the disk**. If you have written anything to the disk in the intervening period between deleting and recovering a file, the written data may have **overwritten** some of the sectors of the disk that were occupied by the deleted file.

Disk Doctor only looks for deleted files in this option; if you tell it to recover a file that has not been deleted, you will get a "NOT FOUND" message.

REMOVE DOS FOR MORE SPACE

Option #4 - Remove DOS. If you are like most Apple users I know, you only boot your system with one or two diskettes. The rest all have a copy of DOS on them that is simply taking up valuable disk space. **Option 4 (lines 4000-4210)** does two things: 1) makes available an **additional 32 sectors** (tracks 1 and 2), and 2) places a "mini-boot" routine on track 0, sector 0. (DOS will not use track 0, and that space cannot be recovered.) If you try to boot a disk that has had the DOS removed in this manner you will get a message indicating that there is no DOS on the disk. The assembly listing for the "boot" program accompanies the article; the object code is maintained in Disk Doctor in an array called **NDB (No DOS Boot)**.

SLOT/DRIVE CONTROL

Option #5 - Change Slot and/or Drive. Since I have moved to a two drive system, I wanted to be able to use both drives. If you change drives and then end the program, the slot and drive will revert to the values they had when you ran Disk Doctor. This is because Disk Doctor only changes the **IOB**, and not the DOS internal values for the slot and drive. (lines 5000-5090).

Option #6 - End. Select option 6 when you are ready to exit Disk Doctor.

HOW IT WORKS

When you first **RUN** Disk Doctor, initialization is performed by **lines 10000-10280**. From that time on, **lines 20-50** control the program's flow. First the **MENU** is presented via a **GOSUB** to line 500, and when a valid selection is keyed, control is **RETURNed** at line 600. Then, based on the selection (**SEL**), one of the six subroutines outlined above is called to perform the requested function.

Notice the "endless loop" at lines 20-50. I use this technique heavily in all of my programs, and Disk Doctor is no exception. This little trick eliminates the need for countless GOTO statements! (ED NOTE: This approach speeds up program execution since a GOTO statement forces the program to search for the line number starting at the beginning — lowest line number — of the program each time a GOTO is executed.)

Another trick to eliminate GOTO statements is simulating the **ELSE statement** (which Applesoft does not have). The following example shows how to handle "ELSE" conditions without using GOTO's.

IF cond-A THEN statement
IF NOT cond-A THEN statement

Using the "simulated ELSE" eliminates the line number search (referred to above). It should be apparent that restricting the use of GOTO statements will help speed program execution. Enough rambling; let's look at the machine language subroutines used by Disk Doctor.

THE MACHINE LANGUAGE

Four machine language routines are used during execution of Disk Doctor. The first one is used only during initialization to obtain the address of DOS's **IOB** (Input/Output Block). The second provides the **RWTS** interface. The third provides the hex dump interface. The fourth routine is the nibble zap used by option two.

The routines reside in **page three of memory** at locations **770-833 (\$302-\$341)**, and are **POKEd** into memory by line 10020. Entry points to the three frequently used routines are at **780(\$30C)**, **800(\$320)**, and **810(\$32A)**, and are referred to by Disk Doctor as **RWTS**, **PRBYTE**, and **ZAP**, respectively. The initialization only routine at **\$302** is not named. A source listing of these routines accompanies the article. While the routines are fairly clear, some explanation may be helpful.

RWTS (READ WRITE TRACK SECTOR)

Disk Doctor interrogates the **IOB** (DOS Input/Output Block that controls the disk) error field after each call to the **RWTS** interface. **RWTS** (in DOS) returns a "clear" carry bit in the status register (**P**) but does not clear the **IOB** error field if the call is successful. The interface insures that the status register will be **zero (0)** if there were no errors. If there was an error, then **RWTS** (in DOS) sets this field and the carry bit. The interface simply exits to Applesoft with an error message (Line 11000) when a DOS error occurs.

continued on next page

Apple Disk Doctor (Cont.)

PRBYTE ROUTINE

The hex dump routine that is used to display the sector is really quite simple. The data byte to be displayed is **POKE**d into memory location 8. PRBYTE is then called (e.g. line 1160), and the interface routine calls the "real" (Apple Monitor) PRBYTE routine as described on page 61 of the Apple Reference manual, using the byte at location 8.

ZAP DISK

Because the machine language zap routine changes "nibbles" (half bytes) in the sector, it must know whether to change the upper nibble or the lower one. This information is passed as part of the new nibble data. If the new nibble value has the high bit set (i.e., value is greater than 127), then the left (high) nibble is to be "zapped", otherwise, the right (low) nibble is "zapped". Program statements 2500-2535 prepare for and make the **CALL** to zap the sector byte.

Information passed to the zap routine includes the displacement of the sector byte to change, the new nibble value, and the left or right nibble flag as just mentioned.

To pass the required parameters, Disk Doctor uses the simplest method I know, **POKE**-ing the variable data into known locations. Examination of the initialization at lines 10040 through 10220 shows the parameter fields are

named such that they may be referenced symbolically by the program. This makes the program easier to read to debug and later on, to enhance!

ENHANCEMENTS

I mentioned earlier in the article that Disk Doctor is easy to enhance. To add another option, simply: 1) change the MENU to display the option, 2) change the selection edit line (580) to allow the new option value, 3) change line 40 to call the option, and 4) put your option into Disk Doctor.

There is plenty of room to add option(s) in the 7000-9999 line number range. Remember, the options are called using the GOSUB command, so be sure to RETURN upon completion. What additional option might you wish to add? One feature that comes to my mind is a routine to **read sector X and write it to sector Y**. Another option that might prove useful on occasion is a **VTOC reconstruction** routine. There are a lot of options that could be added to Disk Doctor.

REFERENCE READING

A couple of last notes. If you are not familiar with the organization of information on Apple DOS diskettes, two very good references are, 1) the **Apple DOS manual (Chapter 9 and Appendix C)** and, 2) **Beneath Apple DOS, by Don Wirth and Pieter Lechner** (Copyright 1981 by Quality Software, Reseda, CA 91335).

Beneath Apple DOS is packed full of information, including a description of the boot process which armed me to write the "mini-boot" routine. You really should be comfortable with diskette storage concepts before you use Disk Doctor; otherwise you might really "ZAP" a disk!

DOS 3.2.1

As written, Disk Doctor is a DOS 3.3 utility. To use it with DOS 3.2.1 all you have to do is change the 16 in lines 1040 and 2020 to 13, change the 15 in line 3060 to 12, change the second 255 in line 4130 to 248, delete line 4120, and change line 4160 to RETURN. It works well for DOS 3.2.1 as well. Everything works; that is, except the "mini-boot" routine. Because DOS 3.3 boots in a significantly different way than DOS 3.2.1, do not use the "mini-boot". It simply will not work. On a positive note, Disk Doctor uses the page 3 DOS vectors exclusively to interface with DOS, and as such it should work with 32K Apples.

SUMMARY

While there are a lot of "disk zap" routines available, I think Disk Doctor will be a good addition to your library. Even if you already have a disk utility, I'm sure you can use some of the techniques implemented in Disk Doctor.



```
10 REM *****
11 REM * DISK DOCTOR *
12 REM * BY BEN COLLEY *
13 REM * COPYRIGHT (C) 1982 *
14 REM * BY MICRO-SPARC INC *
15 REM * LINCOLN, MA. 01773 *
16 REM *****
18 GOSUB 10000
20 FOR I = 1 TO 2 STEP 0
30 GOSUB 500
40 ON SEL GOSUB 1000,2000,3000,4000,5000,6000
50 NEXT
60 TEXT : HOME : END
500 HOME
505 VTAB 5: HTAB 15: INVERSE : PRINT " OPTIONS ":
    NORMAL
510 VTAB 8
520 HTAB 12: PRINT "1) DUMP SECTOR": PRINT
530 HTAB 12: PRINT "2) ZAP SECTOR": PRINT
540 HTAB 12: PRINT "3) RECOVER FILE": PRINT
550 HTAB 12: PRINT "4) REMOVE DOS": PRINT
560 HTAB 12: PRINT "5) SET DRIVE PARMS": PRINT
570 HTAB 12: PRINT "6) END": PRINT
580 FOR XX = 1 TO 2 STEP 0: VTAB 22: INPUT "ENTER
    SELECTION:";S$: IF S$ > "0" AND S$ < "7" THEN XX
    = 2
590 NEXT
600 SEL = VAL (S$): RETURN
1000 HOME
1010 FOR XX = 1 TO 2 STEP 0: VTAB 7: HTAB 5: INPU
    T "ENTER TRACK:";S$:TRK = VAL (S$): IF TRK > - 1
    AND TRK < 35 AND S$ < > "" THEN XX = 2
1020 IF NOT (S$ > = "0" AND S$ < = "9") THEN X
    X = 1
1030 NEXT
1040 FOR XX = 1 TO 2 STEP 0: VTAB 9: HTAB 5: INPU
    T "ENTER SECTOR:";S$:SEC = VAL (S$): IF SEC > -
    1 AND SEC < 16 AND S$ < > "" THEN XX = 2
1050 IF NOT (S$ > = "0" AND S$ < = "9") THEN X
    X = 1
1060 NEXT
1070 FOR XX = 1 TO 2 STEP 0: VTAB 11: HTAB 5: INP
    UT "LIST ON PRINTER? (Y/N):";S$: IF S$ = "Y" OR S$
    = "N" THEN XX = 2
1080 NEXT
```

```
1090 POKE ITRK,TRK: POKE ISEC,SEC
1100 POKE ICMD,RD: CALL RWTS
1110 IF PEEK (IERR) < > 0 THEN GOSUB 11000: RE
    TURN
1120 IF S$ = "Y" THEN INC = 16: PRINT CHR$ (4);"
    PR#1": PRINT CHR$ (9);"80N";: PRINT "TRACK=";TRK;
    " SECTOR=";SEC: PRINT
1130 IF S$ = "N" THEN INC = 8: HOME : PRINT "
    TRACK=";TRK;" SECTOR=";SEC: PRINT
1140 FOR XX = BUF TO BUF + 256 - INC STEP INC
1150 IF XX = BUF + 128 AND S$ = "N" THEN VTAB 22
    : INPUT "PRESS RETURN TO CONTINUE:";X$: VTAB 5: CA
    LL ESCF
1160 POKE BYTE,XX - BUF: CALL PRBYTE
1170 PRINT " :";
1180 FOR XY = 0 TO INC - 1
1190 POKE BYTE, PEEK (XX + XY): CALL PRBYTE: REM
    PRINT 1 BUFFER BYTE
1200 IF INT (XY / 2) * 2 < > XY THEN PRINT " "
    ;
1210 NEXT
1220 PRINT " :";
1230 FOR XY = 0 TO INC - 1
1240 XZ = PEEK (XX + XY): IF XZ > 127 THEN XZ = X
    Z - 128
1250 IF XZ < 32 THEN XZ = 46
1260 PRINT CHR$ (XZ);
1270 NEXT
1280 PRINT " :": NEXT
1290 IF S$ = "N" THEN VTAB 22: INPUT "PRESS RETU
    RN TO CONTINUE:";X$
1300 IF S$ = "Y" THEN PRINT : PRINT : PRINT CHR
    $ (4);"PR#0"
1310 RETURN
2000 HOME
2005 FOR XX = 1 TO 2 STEP 0: VTAB 7: HTAB 5: INPU
    T "ENTER TRACK:";S$:TRK = VAL (S$): IF TRK > - 1
    AND TRK < 35 AND S$ < > "" THEN XX = 2
2010 IF NOT (S$ > = "0" AND S$ < = "9") THEN X
    X = 1
2015 NEXT
2020 FOR XX = 1 TO 2 STEP 0: VTAB 9: HTAB 5: INPU
    T "ENTER SECTOR:";S$:SEC = VAL (S$): IF SEC > -
    1 AND SEC < 16 AND S$ < > "" THEN XX = 2
```



```

2025 IF NOT (S$ > = "0" AND S$ < = "9") THEN X
      X = 1
2030 NEXT
2035 POKE ITRK,TRK: POKE ISEC,SEC
2040 POKE ICMD,RD: CALL RWTS
2045 IF PEEK (IERR) < > 0 THEN GOSUB 11000: RE
      TURN
2050 HOME : PRINT "          TRACK=";TRK;"      SECTO
      R=";SEC: PRINT
2055 VTAB 22: HTAB 1: PRINT " >AGE FLIP  >OVE
      >DIT  >RITE  >EXIT";
2060 INVERSE : HTAB 1: VTAB 22
2065 PRINT "P";: HTAB 13:: PRINT "M";: HTAB 20: P
      RINT "E";: HTAB 27: PRINT "W";: HTAB 35: PRINT "X"
      ;
2070 NORMAL
2075 B1 = 0:B2 = 128:T = - 128: GOSUB 2800
2080 VK = 1:SEL = 13: REM MODE=MOVE
2085 FOR XX = 1 TO 2 STEP 0
2090 IF VK = 0 THEN GOSUB 2975
2095 IF SEL = 5 THEN GOSUB 2400
2100 IF SEL = 13 THEN GOSUB 2700
2105 IF SEL = 16 THEN B1 = B1 + T:B2 = B2 + T: GO
      SUB 2800
2110 IF SEL = 23 THEN XX = 2
2115 IF SEL = 24 THEN XX = 2
2120 NEXT
2125 IF SEL = 23 THEN POKE ICMD,WR: CALL RWTS: I
      F PEEK (IERR) < > 0 THEN GOSUB 11000
2130 RETURN
2393 REM
2395 REM EDIT MODE
2398 REM
2400 VTAB 3: HTAB 34: INVERSE : PRINT "EDIT";: NO
      RMAL
2402 VTAB 24: HTAB 6: PRINT "<0-F>-HEX DIGIT  OR
      <-, ->";
2405 FOR XY = 1 TO 2 STEP 0
2410 HT = HT + 1: REM MOVE TO NEXT NIBBLE
2415 IF HT > 24 THEN HT = 6:VT = VT + 1: IF VT >
      20 THEN VT = 5
2420 IF INT (HT / 5) * 5 = HT THEN HT = HT + 1:
      REM CHECK FOR COL 10,15,20
2425 GOSUB 2975:VK = 0
2430 IF (S$ > = "0" AND S$ < = "9") OR (S$ > =
      "A" AND S$ < = "F") THEN GOSUB 2500:VK = 1
2435 IF SEL = 13 THEN VK = 1:XY = 2: REM SWITCH
      TO MOVE
2440 IF SEL = 16 THEN B1 = B1 + T:B2 = B2 + T: GO
      SUB 2800: REM PAGE FLIP
2445 IF SEL = 23 THEN VK = 1:XY = 2: REM EXIT
2450 IF SEL = 24 THEN VK = 1:XY = 2: REM WRITE
2455 IF SEL = 21 THEN VK = 1: REM RIGHT ARROW
2460 IF SEL = 8 THEN VK = 1:HT = HT - 2: IF HT <
      5 THEN HT = 24:VT = VT - 1: IF VT < 5 THEN VT = 20
      : REM LEFT ARROW
2465 IF SEL = 8 AND ( INT ((HT + 1) / 5) * 5 = HT
      + 1) THEN HT = HT - 1: REM CHECK COL 10,15,20
2470 IF VK = 0 THEN HT = HT - 1: REM NO VALID KE
      Y
2475 NEXT
2480 REM SET HT TO NEXT CELL BEFORE LEAVING
2485 HT = INT ((HT + 4) / 5) * 5: IF HT > 20 THEN
      HT = 5:VT = VT + 1: IF VT > 20 THEN VT = 5
2488 VTAB 24: HTAB 1: PRINT SPC( 35);
2490 RETURN
2495 REM PREPARE FOR AND CALL ZAP
2496 REM CALC BUFFER DISPLACEMENT
2500 X1 = (VT - 5) * 8 + ZTBL(HT - 6) - 30
2505 IF PEEK (1536) = 184 THEN X1 = X1 + 128: RE
      M NIBBLE IN 2ND HALF OF SECTOR
2510 POKE BDISP,X1
2515 BIT = 0
2520 IF ZTBL(HT - 6) = ZTBL(HT - 5) THEN BIT = 12
      8: REM DET IF LEFT NIBBLE
2525 IF S$ < = "9" THEN POKE BYTE, VAL (S$) + B
      IT
2530 IF S$ > = "A" THEN POKE BYTE, ASC (S$) - 5
      5 + BIT

```

```

2535 CALL ZAP: REM ZAP THE NIBBLE
2540 REM NOW UPDATE DISPLAY
2544 VTAB VT
2545 HTAB HT: PRINT S$;: REM PRINT NIBBLE
2550 X1 = PEEK (BUF + X1): REM GET NEW VALUE
2555 VTAB VT: HTAB ZTBL(HT - 6): REM POS FOR CHAR
2560 IF X1 > 127 THEN X1 = X1 - 128
2565 IF X1 < 32 THEN X1 = 46
2570 PRINT CHR$( X1);: REM PRINT CHARACTER
2575 RETURN
2685 REM
2690 REM MOVE MODE
2695 REM
2700 VTAB 3: HTAB 34: INVERSE : PRINT "MOVE";: NO
      RMAL
2705 VTAB 24: HTAB 3: PRINT "<I>-UP <M>-DOWN <J>-
      LEFT <K>-RIGHT";
2710 FOR XY = 1 TO 2 STEP 0
2715 GOSUB 2975:VK = 0
2720 IF S$ = "K" THEN HT = HT + 5: IF HT > 20 THE
      N HT = 5:VT = VT + 1: IF VT > 20 THEN VT = 5
2725 IF S$ = "J" THEN HT = HT - 5: IF HT < 5 THEN
      HT = 20:VT = VT - 1: IF VT < 5 THEN VT = 20
2730 IF S$ = "I" THEN VT = VT - 1: IF VT < 5 THEN
      VT = 20
2735 IF S$ = "M" THEN VT = VT + 1: IF VT > 20 THE
      N VT = 5
2740 IF SEL = 5 THEN VK = 1:XY = 2
2745 IF SEL = 16 THEN B1 = B1 + T:B2 = B2 + T: GO
      SUB 2800
2750 IF SEL = 23 THEN VK = 1:XY = 2
2755 IF SEL = 24 THEN VK = 1:XY = 2
2760 NEXT
2765 VTAB 24: PRINT SPC( 38);
2770 RETURN
2800 POKE 35,21: VTAB 5: HTAB 1: CALL ESCF: POKE
      35,24: REM CLEAR MIDDLE OF SCREEN
2805 FOR X1 = BUF + B1 TO BUF + B2 - 8 STEP 8
2810 POKE BYTE,X1 - BUF: CALL PRBYTE
2815 PRINT " : ";
2820 FOR X2 = 0 TO 7
2825 POKE BYTE, PEEK (X1 + X2): CALL PRBYTE
2830 IF INT (X2 / 2) * 2 < > X2 THEN PRINT " "
      ;
2835 NEXT
2840 PRINT " :";
2845 FOR X2 = 0 TO 7
2850 X3 = PEEK (X1 + X2)
2855 IF X3 > 127 THEN X3 = X3 - 128
2860 IF X3 < 32 THEN X3 = 46
2865 PRINT CHR$( X3);
2870 NEXT
2875 PRINT " :";
2880 NEXT
2885 T = - T
2890 VK = 0:SEL = 0:VT = 5:HT = 5: REM INIT CURS
      OR POSITION
2895 RETURN
2975 VTAB VT: HTAB HT: REM POSITION CURSOR
2980 GET S$: PRINT CHR$( 0): REM PRINT NON-DISP
      LAY CHAR
2985 IF S$ < > "" THEN SEL = ASC (S$)
2990 IF S$ = CHR$( 0) THEN SEL = 0
2995 RETURN
3000 FOR XX = 1 TO 2 STEP 0: HOME : VTAB 7
3010 HTAB 3: INPUT "ENTER FILE NAME:";S$
3020 IF LEN (S$) < > 0 THEN XX = 2
3030 NEXT
3040 POKE ITRK,17
3050 XU = 0
3060 FOR XX = 15 TO 1 STEP - 1
3070 POKE ISEC,XX
3080 POKE ICMD,RD: CALL RWTS
3090 IF PEEK (IERR) < > 0 THEN GOSUB 11000: RE
      TURN
3100 FOR XY = 11 TO 221 STEP 35
3110 IF PEEK (BUF + XY) = 255 THEN GOSUB 3500

```

continued on page 193



BRENT IVERSON

QUASAR II

QUASAR II: A Machine Language Arcade Game

by Brent Iverson
9538 Pondwood Road
Boca Raton, FL 33433

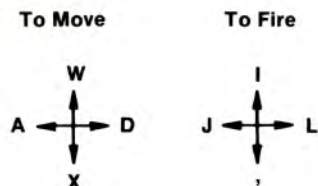
INTRODUCTION

A quasar in the Andromeda Nebula has spread galactic debris through many nearby quadrants. Your mission is to enter the Andromeda Nebula and clear as many quadrants of debris as you can with the three ships allotted to you.

The universe fades away before your eyes. Seconds later, your ship reappears, in the center of a quadrant of deep space. Stellar debris approaches your ship from all sides. Swiftly, you fire your thrusters, narrowly missing a fast-moving chunk of rock. A blast from one of your photon guns vaporizes another piece. Acting quickly, you destroy the remaining debris in the quadrant.

Your hyperspace automatically kicks in, as your instruments tell you that you are advancing to quadrant two. But this quadrant contains even more debris than the last one. . . and the debris is moving more quickly. Frantically, you fire your photon guns. How long can you hold out? How long will it be before. . . The End?

INSTRUCTIONS



As shown above, movement of your ship is controlled by the **four keys** in the **W-A-D-X diamond**. For example, to **move left**, press **A**. Pressing **A** again will make you move to the left **more quickly**. There is a limit to how fast you can move in any direction, though.

To **fire** your photon guns, use the four keys in the **I-J-L-, diamond**. For example, to **fire to the right**, press **L**. The instructions for this game are also given when you begin play.

ENTERING THE PROGRAM

First, enter the Applesoft program shown in **listing 1**. When you have finished checking it for errors, **SAVE** it. The name you save it under doesn't matter. . . so why not call it **QUASAR**, or something similar?

After you have **SAVED** it, enter the monitor and type in the **shape table** given in **listing 2**. When you have done so, save it to disk with the following command:

BSAVE QUASAR.SHAPES, A\$300,L\$0045

In this case, you must save the shapes under the name given, as that is the name that the program in listing 1 uses to look for them on your disk.

Finally, you can enter the machine language program for Quasar directly into memory using the instructions for entering

machine language programs in the **Letters** section of this issue. Having entered the program into memory you can save it to disk with the command:

BSAVE QUASAR.OBJ,A\$4000,L\$050E

Now you are ready to play. Warning: This game has been known to induce sweaty palms and increased pulse rate!

THE PROGRAM — GENERAL COMMENTS

Quasar II is a fast (machine language and Applesoft), hi-res arcade game. All of the action is in the machine language for speed. Applesoft is used between quadrants to generate random numbers, keep track of ships lost, and keep track of the current quadrant number. It is also used to print instructions at the start of a game. Writing assembler code for all of these tasks would have been a tedious job, particularly for the random numbers. Furthermore, the few seconds that Applesoft takes to generate these random numbers is just enough time for the player to catch his breath and glance down at his score. Therefore, all random numbers are generated in Applesoft.

THE MACHINE LANGUAGE PROGRAM

This part of the game consists of a short main program (whose primary function is to call all of the other routines in the program), and the subroutines that the main program calls.

These subroutines are primarily of two kinds: **crunching** and **checking**. The **crunching routines** (whose names end with "crunch") are used to move things around, checking for the limits of the screen. Note that since all coordinates are one-byte numbers (0/255), it is not necessary to check for horizontal wraparound. When an object's position exceeds 255, it automatically wraps around to 0, and vice-versa.

The crunch routines also call Applesoft hires routines to erase and draw what they have been moving. The crunch routines do not check for collisions between the player, his missile, and the space debris.

That is the function of the **checking routines**, whose names all end in "check." These routines **check for collisions** between the above-mentioned items. If a collision is detected, the routine will jump to an appropriate **collide routine**, whose function is to take care of the results of a collision.

In an arcade game such as this one, it is important to keep the game moving at the **same pace**, regardless of how many or how few things are on the screen at one time. This is accomplished in this program in two ways. First, the crunching routines continue to move all debris around the screen, whether or not the object "exists" at that time. If an object does not "exist" (meaning that it is inactive), the crunch routines do not draw it or erase it on the screen. That object will also not be considered for possible collisions by the check routines.

In other words, **all routines try to spend as much time on inactive objects as they do on active ones**. As it turned out, the Applesoft hi-res routines for drawing shapes took up a

great deal of time, compared to the rest of the program. To counteract this, I had to tack on a "wait" routine, which delays the program for a specified amount of time. This amount of time is related to how many objects are active at the time it is called vs. how many were active when the player first entered the quadrant. Further information on this and other routines can be found in the section describing the individual routines in the machine-language program.

MACHINE LANGUAGE ROUTINE DESCRIPTIONS

Main Program Loop: This loop calls the checking and crunching routines. It also checks for any of the conditions required for a return to Applesoft: Having no objects left (**num=0**) or having the player destroyed (**rtncode=1**). If either of these is true, it executes an **RTS** instruction, which returns control to the Applesoft part of the game.

Ocrunch: Moves all objects. If the code of the object is 1, meaning that the object is active, it erases and redraws it.

Ecrunch: If the explosion is still active (**Ecount** is greater than zero), it decrements **Ecount** and draws the explosion with **Hcolor=Ecount**.

Mcrunch: If the missile is active (**Mactive** is greater than zero), it moves and erases it. Then it decrements **Mactive**. If **Mactive** is still greater than zero, it redraws the missile at its new position.

Pcrunch: Moves the player. If a key was pressed, it resets the keyboard strobe and takes the appropriate action for that key. If **Pxvel** or **Pyvel** was changed, it makes sure that they are still within the legal limits of -3 and 3. If a missile was shot, it initializes the missile's position and initializes **Mactive**. Finally, it erases the player and redraws him.

Mcheck: Checks for a collision between the missile and all active objects. If there was one, it jumps to **Mcollide**.

Pcheck: Checks for a collision between the player and all active objects. If there was one, it jumps to **Pcollide**.

Mcollide: Initializes the explosion to be at the position of the object that was destroyed. Sets the code of that object to 2, rendering it inactive. Increments the score by one, then prints it, using the Applesoft routine to print a floating-point number. Sets **Mactive** to zero. Erases the missile and the object.

Pcollide: Sets **Rtncode** to 1. For **Temp2=\$20** to zero, it draws the player with **Hcolor=Temp2**, moves everything else around the screen, and makes an explosion sound. After it has done this, and the player is gone, it moves everything else around the screen \$40 more times.

Sound: Makes a noise. The pitch depends on the value of the variable tone.

Lastobj: Moves everything around the screen \$20 times, waiting for the last object's explosion to disappear.

Wait: Delay routine. The time of the delay depends on the value of [Max-Num].

continued on next page

QUASAR II: A Machine Language Arcade Game (Cont.)

MACHINE LANGUAGE VARIABLES

APPLE ROUTINES AND ADDRESSES USED

Kybd: location to read keyboard

Strobe: resets keyboard strobe

Hcolor: sets Hcolor to contents of X-register

Hposn: positions hi-res cursor. Sets X and Y registers to the horizontal coordinate (X=low, Y=high), and puts the vertical coordinate in the accumulator.

Draw: Draws a shape at the position of the Hi-Res cursor. Sets the X and Y registers to the starting address of the shape you want drawn (the address of the shape, not the address of the whole table); X=low, Y=high. Puts the desired rotation in the accumulator.

Hplot: Identical to Hposn, except that it plots a point at the location given. Setup the same as Hposn.

Ch: text cursor's horizontal displacement from left of text window.

Tab: text cursor's vertical position.

Prntfac: prints the contents of the floating-point "accumulator" (locations \$9D-\$A3) at the current text cursor position.

Givayf: puts the two-byte hex number defined by the A and Y registers into the floating-point "accumulator" (A=lsb, Y=msb).

note: For more details on how to call and use the routines mentioned above, I recommend William F. Luebbert's book, *What's Where In The Apple*, and Roger Wagner's *Assembly Lines* columns in *Softalk* magazine.

EXPLOSION VARIABLES

Ex: explosion x position

Ey: explosion y position

Ecount: number of turns left until the explosion ends. Also, the Hcolor used when the explosion is drawn in this turn.

OTHER VARIABLES

Score: two bytes used to hold the player score: 0-65025.

Index: temporary variable used to index arrays.

Char: keyboard input

Tone: pitch of sound to make when the sound routine is called.

Num: current number of active objects left.

Rincode: condition of return to Applesoft. 1=player destroyed, 2=all objects destroyed (advance to next quadrant).

Max: maximum number of objects on-screen. The number of objects that are active when you first move into a new quadrant.

Temp, Temp2, Temp3: temporary variables used in many ways.

APPLESOFT VARIABLES

AS: user response to questions

I: Index. A temporary variable used while generating random numbers.

Nu: number of objects to put in current quadrant. Linearly related to Q.

Q: number of current quadrant

S: number of ships left

SP: current maximum allowable speed for the debris in the quadrant. Linearly related to Q.

X,Y: variables used to hold random numbers.

OBJECT VARIABLES

Xpos: object x-position array

Ypos: object y-position array

Xvel: object x-velocity array

Yvel: object y-velocity array

Color: array of Hcolors to draw the objects in

Code: array of objects' codes: 1=move, erase, draw, and check for collisions with the missile and the player. 2=move only; do not check for collisions.

XX: object xpos

YY: object ypos

XV: object xvel

YV: object yvel

Tcol: object Hcolor

Tcode: object code

PLAYER VARIABLES

Pxpos: player x position

Pypos: player y position

Pxvel: player x velocity

Pyvel: player y velocity

Op xpos: player's former x position

Op ypos: player's former y position

MISSILE VARIABLES

Mactive: number of turns left until the missile disappears. If Mactive is zero, then the missile is inactive.

Mx: missile x position

My: missile y position

Mxvel: missile x velocity

Myvel: missile y velocity

Om x: missile's former x position

Om y: missile's former y position

Quasar Basic
by Brent Iverson
Copyright © 1982
by Micro SPARC Inc.

```
0 PRINT CHR$(4)"LOAD QUASAR.SHAPE"; PRINT CHR$(4)
  "LOAD QUASAR.OBJ"
10 GOSUB 5000
20 GOSUB 4000
30 CALL 16304
40 ON PEEK(20606) GOSUB 2000,3000
50 GOTO 20
1999 REM ---SHIP DESTROYED
2000 S = S - 1; VTAB 22; HTAB 39; PRINT S;
2010 IF S = 0 THEN RETURN
2020 POP : TEXT : VTAB 23; HTAB 4; PRINT "WOULD YOU LIKE
  TO PLAY AGAIN?"; POKE - 16368,0; GET A$
2030 IF A$ = "N" THEN HOME : END
2040 GOTO 10
2999 REM ---QUADRANT CLEARED
3000 Q = Q + 1
3010 VTAB 23; HTAB 8; PRINT "ADVANCING TO QUADRANT "Q;
3020 RETURN
3999 REM ---INIT,PART 2
4000 HGR : VTAB 22; HTAB 1
4010 PRINT "SCORE:"; SPC(6); "QUADRANT:"; SPC(6); "SHIP
  S LEFT:";
4020 HTAB 7; PRINT PEEK(20600) + PEEK(20601) * 256;
4030 HTAB 22; PRINT Q; HTAB 39; PRINT S;
4040 NUM = 3 + Q; IF NUM > 15 THEN NUM = 15
4050 SPD = Q / 4; IF SPD > 4 THEN SPD = 4
4060 FOR I = 0 TO NUM
4070 ON INT ( RND (1) * 4 + 1) GOTO 4080,4090,4100,411
  0
4080 X = RND (1) * 240 + 10; Y = RND (1) * 5 + 5; GOTO
  4120
4090 X = RND (1) * 5 + 5; Y = RND (1) * 140 + 10; GOTO
  4120
4100 X = RND (1) * 240 + 10; Y = 155 - RND (1) * 5; GOTO
  4120
4110 X = 250 - RND (1) * 5; Y = RND (1) * 140 + 10
4120 POKE 20400 + I,X; POKE 20496 + I,Y
4130 X = RND (1) * SPD + 1; IF RND (1) < .5 THEN X = 2
  56 - X
```

```
4140 POKE 20512 + I,X
4150 Y = RND (1) * SPD + 1; IF RND (1) < .5 THEN Y = 2
  56 - Y
4160 POKE 20528 + I,Y
4170 X = INT ( RND (1) * 6 + 1); IF X = 4 THEN 4170
4180 POKE 20544 + I,X; POKE 20560 + I,1
4190 NEXT
4200 POKE 20584,127; POKE 20585,80; POKE 20586,0; POKE
  20587,0
4210 POKE 20590,0; POKE 20599,0; POKE 20605,NUM; POKE 2
  0606,0; POKE 20607,NUM
4220 POKE - 16368,0
4230 VTAB 23; HTAB 1; CALL - 868
4240 RETURN
4999 REM ---INIT,PART 1
5000 POKE 20600,0; POKE 20601,0; Q = 1; S = 3; SCALE = 1
5010 TEXT : HOME : NORMAL : POKE 216,0
5020 HOME : INVERSE : HTAB 16; PRINT "QUASAR " CHR$(9
  1); NORMAL
5030 VTAB 3; HTAB 12; PRINT "BY BRENT IVERSON"
5040 VTAB 5; HTAB 6; PRINT "WOULD YOU LIKE INSTRUCTIONS
  ?"; GET A$; HTAB 1; CALL - 868
5050 IF A$ = "N" THEN HOME : RETURN
5060 VTAB 3; HTAB 1
5070 PRINT " DEEP IN THE ANDROMEDA NEBULA, A QUA
  SAR HAS EXPLODED, SPREADING DEBRIS IN ALL DIRECTIONS
  ."
5080 PRINT : PRINT " YOUR MISSION IS TO ENTER THE NE
  BULA,CLEARING AS MANY QUADRANTS OF DEBRIS AS YOU CA
  N WITH THE THREE SHIPS ALLOTTED TOYOU."
5100 PRINT : PRINT SPC(8)"TO MOVE" SPC(8)"TO FIRE"
5110 PRINT SPC(8)"-----" SPC(8)" "
5130 PRINT SPC(11)"W" SPC(14)"I"
5140 PRINT SPC(11)"." SPC(14)". "
5150 PRINT SPC(9)"A...D" SPC(10)"J...L"
5160 PRINT SPC(11)"." SPC(14)". "
5170 PRINT SPC(11)"X" SPC(14)". "
5180 PRINT : PRINT " EACH PARTICLE OF DEBRIS THAT YO
  U HIT IS WORTH ONE POINT."
5190 PRINT : PRINT SPC(7)"(PRESS ANY KEY TO START)"
5200 POKE - 16368,0; WAIT - 16304,128
5210 HOME : RETURN
```

1
1

APPLESOFT PROGRAM DOCUMENTATION

- 20: Bload the shape table and the machine language part of the game.
- 20: Gosub 5000 and 4000 to print the instructions and generate random numbers for a new quadrant of debris.
- 30: Call the machine language part of the game.
- 40: Look at the return code (held in memory location 20606) and take appropriate action.
- 50: Go back to 20 to start out a new quadrant.
- 2000: The return code must have been 1 for the program to get here, so the player's ship must have been destroyed. Decre-

ment the number of ships that he has left. If he has none left, then ask him if he'd like to play another game and take appropriate action for the response.

3000: In order for the program to get here, the return code must have been 2, meaning that the player cleared another quadrant of debris. Add one to the current quadrant number (Q), print a message telling him what quadrant he is about to enter, and then return.

4000: Initialization, part two. This routine is called every time the player is going to enter a new quadrant. It clears the screen, and then updates the display in the text window. It figures out the number of

objects to put in the quadrant, and the new maximum speed of these objects. Then it generates the starting positions, velocities, and colors for these objects. It initializes the player's position and velocity, the missile counter, and a few other variables. It then resets the keyboard strobe, clears line 23 of the text window, and returns.

5000: Initialization, part one. This is the initialization that takes place at the start of every game. It resets the number of ships, the quadrant number, and the score. It also presents the title page. If the player wants instructions, it gives them to him. Finally, it returns.



QUASAR • SHAPES

*300.345

```
0300- 03 00 08 00 16 00 2D 00
0308- 29 3E 3E 27 27 25 67 2D
0310- 2E 07 00 1C 27 FC 4D 20
0318- 1C 27 FC 36 BF 3E B7 2D
0320- AE 35 6E 24 65 2C 25 38
0328- 07 00 1A 15 DF 09 F5 1A
0330- 15 DF 22 DC 1A 0C 04 18
0338- 67 08 1C 4C 31 8D 0C 04
0340- 00 8D 60 50 00 00
```

SOURCE FILE: QUASAR.SOURCE

```
0000: 1 *****
0000: 2 # QUASAR.OBJ #
0000: 3 # BY BRENT IVERSON #
0000: 4 # COPYRIGHT (C) 1982 #
0000: 5 # BY MICRO-SPARC INC #
0000: 6 # LINCOLN, MA. 01773 #
0000: 7 # TOOLKIT ASSEMBLER #
0000: 8 *****
----- NEXT OBJECT FILE NAME IS QUASAR.SOURCE.OBJ0
4000: 9 ORG #4000
0000: 10 OBJ #4000
4000: 11 #
4000: 12 #-----
4000: 13 # APPLE ROUTINES AND ADDRESSES
4000: 14 #-----
4000: 15 #
C000: 16 KYBD EQU #C000
C010: 17 STROBE EQU #C010
F6F0: 18 HCOLOR EQU #F6F0
F411: 19 HPOSN EQU #F411
F601: 20 DRAW EQU #F601
F457: 21 HPL0T EQU #F457
0024: 22 CH EQU #0024
FB5B: 23 TABV EQU #FB5B
ED2E: 24 PRNTFAC EQU #ED2E
E2F2: 25 GIVAYF EQU #E2F2
4000: 26 #
4000: 27 #-----
4000: 28 # VARIABLES
4000: 29 #-----
4000: 30 #
4000: 31 # OBJECT VARIABLES
4000: 32 #
5000: 33 XPOS EQU #5000
5010: 34 YPOS EQU #5010
5020: 35 XVEL EQU #5020
5030: 36 YVEL EQU #5030
5040: 37 COLOR EQU #5040
5050: 38 CODE EQU #5050
4000: 39 #
5060: 40 XX EQU #5060
5061: 41 YY EQU #5061
5062: 42 XV EQU #5062
5063: 43 YV EQU #5063
5064: 44 TX EQU #5064
5065: 45 TY EQU #5065
5066: 46 TCOL EQU #5066
5067: 47 TCODE EQU #5067
4000: 48 #
4000: 49 # PLAYER VARIABLES
4000: 50 #
5068: 51 PXPOS EQU #5068
5069: 52 PYPOS EQU #5069
506A: 53 PXVEL EQU #506A
506B: 54 PYVEL EQU #506B
506C: 55 OPXPOS EQU #506C
506D: 56 OPYPOS EQU #506D
4000: 57 #
4000: 58 # MISSILE VARIABLES
4000: 59 #
506E: 60 MACTIVE EQU #506E
506F: 61 MX EQU #506F
5070: 62 MY EQU #5070
```

```
5071: 63 MXVEL EQU #5071
5072: 64 MYVEL EQU #5072
5073: 65 OMX EQU #5073
5074: 66 OMY EQU #5074
4000: 67 #
4000: 68 # EXPLOSION VARIABLES
4000: 69 #
5075: 70 EX EQU #5075
5076: 71 EY EQU #5076
5077: 72 ECOUNT EQU #5077
4000: 73 #
4000: 74 # OTHER VARIABLES
4000: 75 #
5078: 76 SCORE EQU #5078
507A: 77 INDEX EQU #507A
507B: 78 CHAR EQU #507B
507C: 79 TONE EQU #507C
507D: 80 NUM EQU #507D
507E: 81 RTNCODE EQU #507E
507F: 82 MAX EQU #507F
5080: 83 TEMP EQU #5080
5081: 84 TEMP2 EQU #5081
5082: 85 TEMP3 EQU #5082
4000: 86 #
4000: 87 #-----
4000: 88 # MAIN PROGRAM LOOP
4000: 89 #-----
4000: 90 #
4000: 91 START JSR OCRUNCH ;MOVE OBJECTS
4003: 20 E9 40 92 JSR ECRUNCH ;ERASE & DRAW EXPLOSION
4006: 20 17 41 93 JSR MCRUNCH ;MOVE MISSILE
4009: 20 EB 42 94 JSR MCHECK ;CHECK FOR COLLISIONS
400C: 20 BF 41 95 JSR PCRUNCH ;MOVE PLAYER
400F: 20 4F 43 96 JSR PCHECK ;CHECK FOR COLLISIONS
4012: AD 7E 50 97 RTNCHK LDA RTNCODE ;CHECK RTNCODE'S VALUE
4015: F0 01 98 BEQ OKAY ;IF 0, THEN IT WASN'T SET
4017: 60 99 RTS ;RETURN TO BASIC
4018: 20 17 41 100 OKAY JSR MCRUNCH ;MOVE MISSILE AGAIN
401B: 20 EB 42 101 JSR MCHECK ;CHECK FOR COLLISIONS
401E: AD 7D 50 102 NUMCHK LDA NUM ;HOW MANY OBJECTS ARE LEFT?
4021: D0 08 103 BNE END ;IF>0, THEN SKIP THIS
4023: A9 02 104 LDA #02 ;NO OBJECTS LEFT, SO
4025: BD 7E 50 105 STA RTNCODE ;SET RTNCODE AND
4028: 4C B9 44 106 RETURN JMP LASTOBJ ;JMP TO LASTOBJ
402B: 20 E0 44 107 END JSR WAIT ;SLOW EVERYTHING DOWN
402E: 4C 00 40 108 JMP START ;DO IT ALL AGAIN
4031: 109 #
4031: 110 #-----
4031: 111 # OCRUNCH
4031: 112 #-----
4031: 113 #
4031: A9 00 114 OCRUNCH LDA #00 ;INDEX=0
4033: BD 7A 50 115 STA INDEX
4036: AE 7A 50 116 OCRUNCH2 LDX INDEX ;ARRAY-->TEMPS
4039: BD 00 50 117 LDA XPOS,X ;XX=XPOS(INDEX)
403C: BD 60 50 118 STA XX
403F: BD 10 50 119 LDA YPOS,X ;YY=YPOS(INDEX)
4042: BD 61 50 120 STA YY
4045: BD 20 50 121 LDA XVEL,X ;XV=XVEL(INDEX)
4048: BD 62 50 122 STA XV
404B: BD 30 50 123 LDA YVEL,X ;YV=YVEL(INDEX)
404E: BD 63 50 124 STA YV
4051: BD 40 50 125 LDA COLOR,X ;TCOL=COLOR(INDEX)
4054: BD 66 50 126 STA TCOL
4057: BD 50 50 127 LDA CODE,X ;TCODE=CODE(INDEX)
405A: BD 67 50 128 STA TCODE
405D: 129 #
405D: 130 # FIND TX AND TY
405D: 131 #
405D: AD 60 50 132 OALC LDA XX ;TX=XX+XV
4060: 18 133 CLC
4061: 6D 62 50 134 ADC XV
4064: BD 64 50 135 STA TX
4067: AD 61 50 136 LDA YY ;TY=YY+YV
406A: 18 137 CLC
406B: 6D 63 50 138 ADC YV
406E: BD 65 50 139 STA TY
4071: 140 #
4071: 141 # SEE IF IT WENT OFFSCREEN
4071: 142 #
4071: C9 05 143 OPCHK1 CMP #05 ;TY<5?
4073: B0 0C 144 BCS OPCHK2 ;NO
4075: A9 9A 145 LDA #9A ;YES, SO WRAP
4077: 18 146 CLC
4078: 6D 63 50 147 ADC YV
407B: BD 65 50 148 STA TY
407E: 4C BE 40 149 JMP OACTIVE
4081: C9 9A 150 OPCHK2 CMP #9A ;TY>9A?
```

continued on next page

QUASAR II: A Machine Language Arcade Game (Cont.)

```

4083:90 09 151 BCC OACTIVE ;NO
4085:A9 05 152 LDA ;#05 ;YES, SO WRAP
4087:18 153 CLC
4088:6D 63 50 154 ADC YV
408B:BD 65 50 155 STA TY
408E: 156 #
408E: 157 # SEE IF IT IS ACTIVE
408E: 158 #
408E:AD 67 50 159 OACTIVE LDA TCODE ;TCODE=1?
4091:C9 01 160 CMP ;#01
4093:F0 03 161 BEQ OERASE ;YES, SO ERASE&DRAW IT
4095:4C CB 40 162 JMP OUPDATE ;NO, SO DON'T
4098: 163 #
4098: 164 # IF ACTIVE ERASE & DRAW IT
4098: 165 #
4098:A2 00 166 OERASE LDX ;#00 ;HCOLOR=0
4099:20 F0 F6 167 JSR HCOLOR
409D:AE 60 50 168 LDX XX ;POSITION CURSOR
40A0:A0 00 169 LDY ;#00
40A2:AD 61 50 170 LDA YY
40A5:20 11 F4 171 JSR HPOSN
40A8:A2 00 172 LDX ;#00 ;POINT TO SHAPE #1
40AA:A0 03 173 LDY ;#03
40AC:A9 00 174 LDA ;#00 ;ROT=0
40AE:20 01 F6 175 JSR DRAW ;ERASE IT
40B1:AE 66 50 176 ODRAW LDX TCOL ;HCOLOR=TCOL
40B4:20 F0 F6 177 JSR HCOLOR
40B7:AE 64 50 178 LDX TX ;POSITION CURSOR
40BA:A0 00 179 LDY ;#00
40BC:AD 65 50 180 LDA TY
40BF:20 11 F4 181 JSR HPOSN
40C2:A2 00 182 LDX ;#00 ;POINT TO SHAPE#1
40C4:A0 03 183 LDY ;#03
40C6:A9 00 184 LDA ;#00 ;ROT=0
40CB:20 01 F6 185 JSR DRAW ;DRAW IT
40CB:AE 7A 50 186 OUPDATE LDX INDEX
40CE:AD 64 50 187 LDA TX ;XPOS(INDEX)=TX
40D1:9D 00 50 188 STA XPOS, X
40D4:AD 65 50 189 LDA TY ;YPOS(INDEX)=TY
40D7:9D 10 50 190 STA YPOS, X
40DA: 191 #
40DA: 192 # INCREMENT INDEX
40DA: 193 #
40DA:EE 7A 50 194 INC INDEX ;INDEX=INDEX+1
40DD:AD 7A 50 195 LDA INDEX
40E0:CD 7F 50 196 CMP MAX ;INDEX>=MAX?
40E3:B0 03 197 BCS OEND ;YES, SO END
40E5:4C 36 40 198 JMP OCRUNCH2 ;NO, SO DO IT AGAIN
40E8: 199 #
40E8: 200 # THAT IS ALL
40E8: 201 #
40E8:60 202 OEND RTS
40E9: 203 #
40E9: 204 #
40E9: 205 #
40E9: 206 # EXPLOSION ROUTINE
40E9: 207 #
40E9: 208 #
40E9:AD 77 50 209 ECRUNCH LDA ECOUNT
40EC:F0 28 210 BEQ EEND ;IF ECOUNT=0 THEN END
40EE:CE 77 50 211 DEC ECOUNT ;ECOUNT=ECOUNT-1
40F1:AE 77 50 212 LDX ECOUNT ;HCOLOR=ECOUNT
40F4:20 F0 F6 213 JSR HCOLOR
40F7:AE 75 50 214 LDX EX ;POSITION CURSOR
40FA:A0 00 215 LDY ;#00
40FC:AD 76 50 216 LDA EY
40FF:20 11 F4 217 JSR HPOSN
4102:A0 03 218 LDY ;#03 ;POINT TO SHAPE#3
4104:A2 2D 219 LDX ;#2D
4106:A9 00 220 LDA ;#00 ;ROT=0
4108:20 01 F6 221 JSR DRAW ;DRAW EXPLOSION
410B:AD 77 50 222 LDA ECOUNT ;TONE=255-ECOUNT
410E:49 FF 223 EOR ;#FF
4110:BD 7C 50 224 STA TONE
4113:20 9D 44 225 JSR SOUND ;MAKE NOISE
4116: 226 #
4116: 227 # THAT IS ALL
4116: 228 #
4116:60 229 EEND RTS
4117: 230 #
4117: 231 #
4117: 232 # MCRUNCH
4117: 233 #
4117: 234 #
4117:AD 6E 50 235 MCRUNCH LDA MACTIVE ;MACTIVE>0?
411A:D0 01 236 BNE MSAVE ;YES
411C:60 237 RTS ;NO, SO RETURN
411D: 238 #
411D: 239 # SAVE CURRENT POSITION
411D: 240 #
411D:AD 6F 50 241 MSAVE LDA MX ;OMX=MX
4120:BD 73 50 242 STA OMX
4123:AD 70 50 243 LDA MY ;OMY=MY
4126:BD 74 50 244 STA OMY
4129: 245 #
4129: 246 # FIND NEW POSITIONS
4129: 247 #
4129:AD 6F 50 248 MCALC LDA MX ;MX=MX+MXVEL
412C:18 249 CLC
412D:6D 71 50 250 ADC MXVEL
4130:BD 6F 50 251 STA MX
4133:18 252 CLC
4134:AD 70 50 253 LDA MY ;MY=MY+MYVEL
4137:6D 72 50 254 ADC MYVEL
413A:BD 70 50 255 STA MY
413D: 256 #
413D: 257 # CHECK BOUNDARIES
413D: 258 #
413D:AD 70 50 259 MPCHK1 LDA MY
4140:C9 05 260 CMP ;#05 ;MY<5?
4142:B0 0C 261 BCS MPCHK2 ;NO
4144:A9 9A 262 LDA ;#9A ;YES, SO WRAP

```

```

4146:18 263 CLC
4147:6D 72 50 264 ADC MYVEL
414A:BD 70 50 265 STA MY
414D:4C 5D 41 266 JMP MERASE
4150:C9 9A 267 MPCHK2 CMP ;#9A ;MY>9A?
4152:90 09 268 BCC MERASE ;NO
4154:A9 05 269 LDA ;#05 ;YES, SO WRAP
4156:18 270 CLC
4157:6D 72 50 271 ADC MYVEL
415A:BD 70 50 272 STA MY
415D: 273 #
415D: 274 # ERASE MISSILE
415D: 275 #
415D:A2 00 276 MERASE LDX ;#00 ;HCOLOR=0
415F:20 F0 F6 277 JSR HCOLOR
4162:AE 73 50 278 LDX OMX ;POSITION CURSOR
4165:A0 00 279 LDY ;#00
4167:AD 74 50 280 LDA OMY
416A:20 57 F4 281 JSR HPLDT ;ERASE MISSILE
416D:CE 6E 50 282 DEC MACTIVE ;MACTIVE=MACTIVE-1
4170:AD 6E 50 283 LDA MACTIVE ;MACTIVE+1=0?
4173:F0 19 284 BEQ MEND ;YES, SO SKIP
4175:A2 03 285 MDRAW LDX ;#03 ;NO, SO DRAW IT
4177:20 F0 F6 286 JSR HCOLOR ;HCOLOR=3
417A:AE 6F 50 287 LDX MX ;POSITION CURSOR
417D:A0 00 288 LDY ;#00
417F:AD 70 50 289 LDA MY
4182:20 57 F4 290 JSR HPLDT ;DRAW IT
4185:AD 6E 50 291 MSOUND LDA MACTIVE ;TONE=MACTIVE
4188:BD 7C 50 292 STA TONE
418B:20 9D 44 293 JSR SOUND ;MAKE NOISE
418E:60 294 MEND RTS
418F: 295 #
418F: 296 #
418F: 297 # PCRUNCH
418F: 298 #
418F: 299 # SAVE OLD POSITION
418F: 300 #
418F:AD 68 50 301 PCRUNCH LDA PXPOS ;OPXPOS=PXPOS
4192:BD 6C 50 302 STA OPXPOS
4195:AD 69 50 303 LDA PYPOS ;OPYPOS=PYPOS
4198:BD 6D 50 304 STA OPYPOS
419B:AD 00 C0 305 GETKEY LDA KYBD
419E:C9 00 306 CMP ;#00 ;KEY PRESSED?
41A0:B0 03 307 BCS GOTKEY ;YES
41A2:4C 87 42 308 JMP PCALC ;NO
41A5:BD 10 C0 309 GOTKEY STA STROBE ;RESET STROBE
41A8:BD 7B 50 310 INTERP STA CHAR ;STORE KEY-CODE
41AB:A9 D7 311 KEYW LDA ;#D7 ;CHAR="W"?
41AD:CD 7B 50 312 CMP CHAR
41B0:D0 06 313 BNE KEYA ;NO
41B2:CE 6B 50 314 DEC PYVEL ;PYVEL=PYVEL-1
41B5:4C 59 42 315 LDA CHKYV
41B8:A9 C1 316 KEYA JMP PCALC
41BA:CD 7B 50 317 CMP CHAR
41BD:D0 06 318 BNE KEYD ;NO
41BF:CE 6A 50 319 DEC PXVEL ;PXVEL=PXVEL-1
41C2:4C 3C 42 320 JMP CHKXV
41C5:A9 C4 321 KEYD LDA ;#C4 ;CHAR="D"?
41C7:CD 7B 50 322 CMP CHAR
41CA:D0 06 323 BNE KEYX ;NO
41CC:EE 6A 50 324 INC PXVEL ;PXVEL=PXVEL+1
41CF:4C 3C 42 325 JMP CHKXV
41D2:A9 D8 326 KEYX LDA ;#D8 ;CHAR="X"?
41D4:CD 7B 50 327 CMP CHAR
41D7:D0 06 328 BNE CHKMA ;NO
41D9:EE 6B 50 329 INC PYVEL ;PYVEL=PYVEL+1
41DC:4C 59 42 330 JMP CHKYV
41DF:AD 6E 50 331 CHKMA LDA MACTIVE ;MACTIVE>0?
41E2:C9 01 332 CMP ;#01
41E4:90 03 333 BCC KEYI ;NO, SO CHECK OUT OTHER KEYS
41E6:4C 87 42 334 JMP PCALC ;MACTIVE>0, SO DON'T BOTHER
41E9:A9 C9 335 KEYI LDA ;#C9 ;CHAR="I"?
41EB:CD 7B 50 336 CMP CHAR
41EE:D0 00 337 BNE KEYJ ;NO
41F0:A9 FC 338 LDA ;#FC ;MYVEL=-4
41F2:BD 72 50 339 STA MYVEL
41F5:A9 00 340 LDA ;#00 ;MXVEL=0
41F7:BD 71 50 341 STA MXVEL
41FA:4C 76 42 342 JMP ACTIVE
41FD:A9 CA 343 KEYJ LDA ;#CA ;CHAR="J"?
41FF:CD 7B 50 344 CMP CHAR
4202:D0 00 345 BNE KEYL ;NO
4204:A9 00 346 LDA ;#00 ;MYVEL=0
4206:BD 72 50 347 STA MYVEL
4209:A9 FC 348 LDA ;#FC ;MXVEL=-4
420B:BD 71 50 349 STA MXVEL
420E:4C 76 42 350 JMP ACTIVE
4211:A9 CC 351 KEYL LDA ;#CC ;CHAR="L"?
4213:CD 7B 50 352 CMP CHAR
4216:D0 00 353 BNE KEYCOM ;NO
4218:A9 00 354 LDA ;#00 ;MYVEL=0
421A:BD 72 50 355 STA MYVEL
421D:A9 04 356 LDA ;#04 ;MXVEL=4
421F:BD 71 50 357 STA MXVEL
4222:4C 76 42 358 JMP ACTIVE
4225:A9 AC 359 KEYCOM LDA ;#AC ;KEY=","
4227:CD 7B 50 360 CMP CHAR
422A:F0 03 361 BEQ ISCOM ;YES
422C:4C 87 42 362 NONE JMP PCALC ;NO
422F:A9 00 363 ISCOM LDA ;#00 ;MXVEL=0
4231:BD 71 50 364 STA MXVEL
4234:A9 04 365 LDA ;#04 ;MYVEL=4
4236:BD 72 50 366 STA MYVEL
4239:4C 76 42 367 JMP ACTIVE
423C: 368 #
423C: 369 # CHECK PXVEL OR PYVEL
423C: 370 #
423C:AD 6A 50 371 CHKXV LDA PXVEL
423F:30 0C 372 BMI CHKVELX2 ;PXVEL<0
4241:C9 04 373 CHKVELX1 CMP ;#04 ;PXVEL>3?
4243:90 42 374 BCC PCALC ;NO
4245:A9 03 375 LDA ;#03 ;YES, SO REDUCE
4247:BD 6A 50 376 STA PXVEL
424A:4C 87 42 377 JMP PCALC
424D:C9 FD 378 CHKVELX2 CMP ;#FD ;PXVEL<-3?
424F:B0 36 379 BCS PCALC ;NO
4251:A9 FD 380 LDA ;#FD ;YES, SO INCREASE
4253:BD 6A 50 381 STA PXVEL

```



```

4256:4C 87 42 382 JMP PCALC
4259:AD 68 50 383 CHKYV LDA PYVEL
425C:30 0C 384 BHI CHKVELY2 ;PYVEL<0
425E:C9 04 385 CHKVELY1 CMP #004 ;PYVEL>3?
4260:90 25 386 BCC PCALC ;NO
4262:A9 03 387 LDA #003 ;YES, SO REDUCE
4264:8D 68 50 388 STA PYVEL
4267:4C 87 42 389 JMP PCALC
426A:C9 FD 390 CHKVELY2 CMP #0FD ;PYVEL<-3?
426C:B0 19 391 BCS PCALC ;NO
426E:A9 FD 392 LDA #0FD ;YES, SO INCREASE
4270:8D 68 50 393 STA PYVEL
4273:4C 87 42 394 JMP PCALC
4276: 395 #
4276: 396 # MISSILE NOW ACTIVE!
4276: 397 #
4276:AD 68 50 398 ACTIVE LDA PXPOS ;MX=PXPOS
4279:8D 6F 50 399 STA MX
427C:AD 69 50 400 LDA PYPOS ;MY=PYPOS
427F:8D 70 50 401 STA MY
4282:A9 14 402 LDA #14 ;MACTIVE=#14
4284:8D 6E 50 403 STA MACTIVE
4287: 404 #
4287: 405 # CALCULATE NEW POSITION
4287: 406 #
4287:18 407 PCALC CLC ;PXPOS=PXPOS+PXVEL
428B:AD 68 50 408 LDA PXPOS
428B:6D 6A 50 409 ADC PXVEL
428E:8D 68 50 410 STA PXPOS
4291:18 411 CLC ;PYPOS=PYPOS+PYVEL
4292:AD 69 50 412 LDA PYPOS
4295:6D 6B 50 413 ADC PYVEL
4298:8D 69 50 414 STA PYPOS
429B: 415 #
429B: 416 # CHECK FOR BORDERS
429B: 417 #
429B:C9 05 418 CHKTOP CMP #005 ;PYPOS<5?
429D:B0 0C 419 BCS CHKBOT ;NO
429F:A9 9A 420 LDA #9A ;YES, SO WRAP
42A1:18 421 CLC
42A2:6D 6B 50 422 ADC PYVEL
42A5:8D 69 50 423 STA PYPOS
42A8:4C 88 42 424 JMP ERASEPL
42AB:C9 9A 425 CHKBOT CMP #9A ;PYPOS>9A?
42AD:90 09 426 BCC ERASEPL ;NO
42AF:A9 05 427 LDA #05 ;YES, SO WRAP
42B1:18 428 CLC
42B2:6D 6B 50 429 ADC PYVEL
42B5:8D 69 50 430 STA PYPOS
42B8: 431 #
42B8: 432 # ERASE THE PLAYER
42B8: 433 #
42B8:A2 00 434 ERASEPL LDX #000 ;HCOLOR=0
42BA:20 F0 F6 435 JSR HCOLOR
42BD:AE 6C 50 436 LDX OPXPOS ;POSITION CURSOR
42C0:A0 00 437 LDY #000
42C2:AD 6D 50 438 LDA OPYPOS
42C5:20 11 F4 439 JSR HPOSN
42C8:A2 16 440 LDX #16 ;POINT TO SHAPE#2
42CA:A0 03 441 LDY #03
42CC:A9 00 442 LDA #00
42CE:20 01 F6 443 JSR DRAW ;ROT=0
42D1: 444 # ;ERASE PLAYER
42D1: 445 # DRAW THE PLAYER
42D1: 446 #
42D1:A2 03 447 DRAWPL LDX #003 ;HCOLOR=3
42D3:20 F0 F6 448 JSR HCOLOR
42D6:AE 68 50 449 LDX PXPOS ;POSITION CURSOR
42D9:A0 00 450 LDY #000
42DB:AD 69 50 451 LDA PYPOS
42DE:20 11 F4 452 JSR HPOSN
42E1:A2 16 453 LDX #16 ;POINT TO SHAPE#2
42E3:A0 03 454 LDY #03
42E5:A9 00 455 LDA #00
42E7:20 01 F6 456 JSR DRAW ;ROT=0
42EA: 457 # ;DRAW PLAYER
42EA: 458 # WE'RE DONE!
42EA: 459 #
42EA:60 460 PEND RTS
42EB: 461 #
42EB: 462 #
42EB: 463 # MCHECK
42EB: 464 #
42EB: 465 #
42EB:AD 6E 50 466 MCHECK LDA MACTIVE ;MACTIVE=0?
42EE:D0 01 467 BNE MCHECK1 ;NO
42F0:60 468 RTS ;YES, SO RETURN
42F1:A9 00 469 MCHECK1 LDA #000 ;INDEX=0
42F3:8D 7A 50 470 STA INDEX
42F6:AE 7A 50 471 MCHECK2 LDA CODE, X ;TCODE=CODE (INDEX)
42F9:8D 50 50 472 STA TCODE
42FC:8D 67 50 473 LDA XPOS, X ;XX=XPOS (INDEX)
42FF:8D 60 50 474 STA XX
4302:8D 60 50 475 STA YY ;YY=YPOS (INDEX)
4305:8D 10 50 476 LDA YPOS, X
4308:8D 61 50 477 STA YY ;MX-XX<?
430B:38 478 MCHECKX1 SEC
430C:AD 6F 50 479 LDA MX
430F:ED 60 50 480 SBC XX
4312:C9 04 481 CMP #004 ;YES
4314:90 07 482 BCC MCHECKY1 ;YES
4316:C9 FC 483 MCHECKX2 CMP #0FC ;MX-XX>FC?
4318:B0 03 484 BCS MCHECKY1 ;YES
431A:4C 40 43 485 JMP MCHECK3 ;NO, SO NO COLLISION POSSIBLE
431D:38 486 MCHECKY1 SEC ;MY-YY<?
431E:AD 70 50 487 LDA MY
4321:ED 61 50 488 SBC YY
4324:C9 04 489 CMP #004 ;NO
4326:B0 0A 490 BCS MCHECKY2 ;YES
4328:AE 67 50 491 LDX TCODE
432B:E0 02 492 CPX #002 ;IS OBJECT ACTIVE?
432D:B0 03 493 BCS MCHECKY2 ;NO
432F:4C 4D 43 494 JMP MCOLLIDE ;YES, SO IT COLLIDED
4332:C9 FC 495 MCHECKY2 CMP #0FC ;MY-YY>FC?
4334:90 0A 496 BCC MCHECK3 ;YES, SO NO COLLISION POSSIBLE
4336:AE 67 50 497 LDX TCODE
4339:E0 02 498 CPX #002 ;IS OBJECT ACTIVE?
433B:B0 03 499 BCS MCHECK3 ;NO
433D:4C 4D 43 500 JMP MCOLLIDE ;YES, SO IT COLLIDED

4340:EE 7A 50 501 MCHECK3 INC INDEX ;INDEX=INDEX+1
4343:AD 7A 50 502 LDA INDEX
4346:CD 7F 50 503 CMP MAX ;INDEX>MAX?
4349:B0 03 504 BCS MCHECK4 ;YES
434B:4C F6 42 505 JMP MCHECK2 ;NO, SO CHECK NEXT OBJECT
434E:60 506 MCHECK4 RTS ;RETURN
434F: 507 #
434F: 508 #
434F: 509 # PCHECK
434F: 510 #
434F: 511 #
434F:A9 00 512 PCHECK LDA #000 ;INDEX=0
4351:8D 7A 50 513 STA INDEX
4354:AE 7A 50 514 PCHECK2 LDX INDEX
4357:8D 00 50 515 LDA XPOS, X ;XX=XPOS (INDEX)
435A:8D 60 50 516 STA XX
435D:8D 10 50 517 LDA YPOS, X ;YY=YPOS (INDEX)
4360:8D 61 50 518 STA YY
4363:8D 50 50 519 LDA CODE, X ;TCODE=CODE (INDEX)
4366:8D 67 50 520 STA TCODE
4369:38 521 PCHECKX1 SEC
436A:AD 68 50 522 LDA PXPOS
436D:ED 60 50 523 SBC XX
4370:C9 06 524 CMP #006
4372:90 07 525 BCC PCHECKY1 ;YES
4374:C9 FB 526 PCHECKX2 CMP #0FB ;PXPOS-XX>FB?
4376:B0 03 527 BCS PCHECKY1 ;YES
4378:4C 9E 43 528 JMP PCHECK3 ;NO COLLISION POSSIBLE
437B:38 529 PCHECKY1 SEC ;PYPOS-YY<?
437C:AD 69 50 530 LDA PYPOS
437F:ED 61 50 531 SBC YY
4382:C9 06 532 CMP #006
4384:8D 0A 533 BCS PCHECKY2 ;NO
4386:AE 67 50 534 LDX TCODE ;YES
4389:E0 02 535 CPX #002 ;IS OBJECT ACTIVE?
438B:B0 03 536 BCS PCHECKY2 ;NO
438D:4C 39 44 537 JMP PCOLLIDE ;YES, SO COLLISION
4389:C9 FB 538 PCHECKY2 CMP #0FB ;PYPOS-YY>FB?
4392:90 0A 539 BCC PCHECK3 ;NO..NO COLLISION POSSIBLE
4394:AE 67 50 540 LDX TCODE ;YES
4397:E0 02 541 CPX #002 ;IS OBJECT ACTIVE?
4399:B0 03 542 BCS PCHECK3 ;NO
439B:4C 39 44 543 JMP PCOLLIDE ;YES, SO COLLISION
439E:EE 7A 50 544 PCHECK3 INC INDEX ;INDEX=INDEX+1
43A1:AD 7A 50 545 LDA INDEX ;INDEX>MAX?
43A4:CD 7F 50 546 CMP MAX
43A7:B0 03 547 BCS PCHECK4 ;YES
43A9:4C 54 43 548 JMP PCHECK2 ;NO, SO CHECK NEXT OBJECT
43AC:60 549 PCHECK4 RTS ;RETURN
43AD: 550 #
43AD: 551 #
43AD: 552 # MCOLLIDE
43AD: 553 #
43AD: 554 #
43AD:AD 77 50 555 MCOLLIDE LDA ECOUNT ;ECOUNT>0?
43B0:F0 19 556 BEQ P11 ;NO
43B2:A2 00 557 LDX #000 ;YES, SO ERASE CURRENT EXPLOSION
43B4:20 F0 F6 558 JSR HCOLOR ;HCOLOR=0
43B7:AE 75 50 559 LDX EX ;POSITION CURSOR
43BA:A0 00 560 LDY #000
43BC:AD 76 50 561 LDA EY
43BF:20 11 F4 562 JSR HPOSN
43C2:A2 2D 563 LDX #2D ;POINT TO SHAPE#3
43C4:A0 03 564 LDY #003
43C6:A9 00 565 LDA #000 ;ROT=0
43C8:20 01 F6 566 JSR DRAW ;ERASE IT
43CB:AD 60 50 567 P11 LDA XX ;EX=XX
43CE:8D 75 50 568 STA EX
43D1:AD 61 50 569 LDA YY ;EY=YY
43D4:8D 76 50 570 STA EY
43D7:A9 0A 571 LDA #00A ;ECOUNT=#0A
43D9:8D 77 50 572 STA ECOUNT
43DC:AE 7A 50 573 LDX INDEX ;CODE (INDEX)=2
43DF:A9 02 574 LDA #002
43E1:9D 50 50 575 STA CODE, X
43E4:CE 7D 50 576 DEC NUM ;NUM=NUM-1
43E7:18 577 P9 CLC
43EB:AD 78 50 578 LDA SCORE ;INCREMENT SCORE
43EB:69 01 579 ADC #001
43ED:8D 78 50 580 STA SCORE
43F0:90 08 581 BCC P10
43F2:AD 79 50 582 LDA SCORE+1
43F5:69 00 583 ADC #000
43F7:8D 79 50 584 STA SCORE+1
43FA:AC 78 50 585 P10 LDY SCORE ;PUT SCORE IN Y
43FD:AD 79 50 586 LDA SCORE+1 ;PUT SCORE+1 IN A
4400:20 F2 E2 587 JSR GIVAYF ;PUT A&Y IN FP ACCUM
4403:A9 06 588 LDA #006 ;HTAB 7
4405:85 24 589 STA CH
4407:A9 15 590 LDA #015 ;VTAB #15
4409:20 5B FB 591 JSR TABV
440C:20 2E ED 592 JSR PRNTFAC ;PRINT FP ACCUM
440F:A9 00 593 LDA #000 ;MACTIVE=0
4411:8D 6E 50 594 STA MACTIVE
4414:A2 00 595 LDX #000 ;HCOLOR=0
4416:20 F0 F6 596 JSR HCOLOR
4419:AE 6F 50 597 LDA MX ;POSITION CURSOR
441C:A0 00 598 LDY #000
441E:AD 70 50 599 LDA MY
4421:20 57 F4 600 JSR HPLOT ;ERASE MISSILE
4424:AE 60 50 601 LDX XX ;PUT CURSOR AT OBJECT'S POS
4427:A0 00 602 LDY #000
4429:AD 61 50 603 LDA YY
442C:20 11 F4 604 JSR HPOSN
442F:A2 00 605 LDX #000 ;POINT TO SHAPE#1
4431:A0 03 606 LDY #003
4433:A9 00 607 LDA #000 ;ROT=0
4435:20 01 F6 608 JSR DRAW ;ERASE OBJECT
4438:60 609 RTS ;RETURN
4439: 610 #
4439: 611 #
4439: 612 # PCOLLIDE
4439: 613 #
4439: 614 #
4439:A9 01 615 PCOLLIDE LDA #001 ;RTNCODE=1
443B:8D 7E 50 616 STA RTNCODE
443E:A9 20 617 P13 LDA #020 ;TEMP2=#20

```

continued on next page

QUASAR II: A Machine Language Arcade Game (Cont.)

```

4440:8D B1 50 618 STA TEMP2
4443:AE B1 50 619 P14 LDX TEMP2 ;HCOLOR=TEMP2
4446:20 F0 F6 620 JSR HCOLOR
4449:AE 68 50 621 LDX PXPOS ;POSITION CURSOR
444C:A0 00 622 LDY #000
444E:AD 69 50 623 LDA PYPOS
4451:20 11 F4 624 JSR HPOSN
4454:A2 16 625 LDX #16 ;POINT TO SHAPE#2
4456:A0 03 626 LDY #003
4458:A9 00 627 LDA #000 ;ROT=0
445A:20 01 F6 628 JSR DRAW ;DRAW PLAYER
445D:20 31 40 629 JSR OCRUNCH ;MOVE OBJECTS
4460:20 E9 40 630 JSR ECRUNCH ;DRAW EXPLOSION
4463:20 17 41 631 JSR MCRUNCH ;MOVE MISSILES
4466:20 E0 44 632 JSR WAIT ;SLOW IT DOWN
4469:AD B1 50 633 LDA TEMP2 ;TONE=TEMP2
446C:8D 7C 50 634 STA TONE
446F:20 9D 44 635 JSR SOUND ;MAKE NOISE
4472:CE B1 50 636 DEC TEMP2 ;TEMP2=TEMP2-1
4475:AD B1 50 637 LDA TEMP2 ;TEMP2+1>=1?
4478:18 638 CLC
4479:69 01 639 ADC #001
447B:C9 01 640 CMP #001
447D:B0 C4 641 BCS P14 ;YES
447F:A9 00 642 LDA #000 ;TEMP2=0
4481:8D B1 50 643 STA TEMP2
4484:20 31 40 644 P17 JSR OCRUNCH ;THE PLAYER HAS
4487:20 E9 40 645 JSR ECRUNCH ;FINISHED EXPLODING,
448A:20 17 41 646 JSR MCRUNCH ;SO MOVE THE OTHER
448D:20 E0 44 647 JSR WAIT ;THINGS AROUND FOR AMHILE
4490:EE B1 50 648 INC TEMP2 ;TEMP2=TEMP2+1
4493:AD B1 50 649 LDA TEMP2 ;TEMP2>=40?
4496:C9 40 650 CMP #40
4498:90 EA 651 BCC P17 ;NO, SO DO IT SOME MORE
449A:4C AC 43 652 JMP PCHECK4 ;JUMP TO END OF PCHECK ROUTINE
449D: 653 #
449D: 654 #-----
449D: 655 # SOUND
449D: 656 #-----

449D: 657 #
449D:A9 00 658 SOUND LDA #000 ;TEMP=0
449F:8D 00 50 659 STA TEMP
44A2:A2 00 660 SOUND2 LDX #000 ;X=0
44A4:EB 661 SOUND3 INX ;X=X+1
44A5:8A 662 TXA ;A=X
44A6:CD 7C 50 663 CMP TONE ;A=TONE?
44A9:90 F9 664 BCC SOUND3 ;NO, SO WAIT LONGER
44AB:AD 30 C0 665 SOUND4 LDA #C030 ;MAKE CLICK
44AE:EE 00 50 666 SOUND5 INC TEMP ;TEMP=TEMP+1
44B1:AD 00 50 667 LDA TEMP ;TEMP<3?
44B4:C9 03 668 CMP #003
44B6:90 EA 669 BCC SOUND2 ;YES, SO DO IT AGAIN
44B8:60 670 SOUND6 RTS ;NO, SO RETURN
44B9: 671 #
44B9: 672 #-----
44B9: 673 # LASTOBJ
44B9: 674 #-----
44B9:A9 00 675 LASTOBJ LDA #000 ;TEMP3=0
44BB:8D 02 50 676 STA TEMP3
44BE:20 31 40 677 LASTOBJ2 JSR OCRUNCH ;SLOW THINGS DOWN
44C1:20 E9 40 678 JSR ECRUNCH ;DRAW EXPLOSION
44C4:20 17 41 679 JSR MCRUNCH ;MOVE MISSILES
44C7:20 0F 41 680 JSR PCRUNCH ;MOVE PLAYER
44CA:20 E0 44 681 JSR WAIT ;SLOW THINGS DOWN
44CD:20 E0 44 682 JSR WAIT
44D0:A9 00 683 LDA #000 ;TEMP=0
44D2:8D 00 50 684 STA TEMP
44D5:EE 02 50 685 INC TEMP3 ;TEMP3=TEMP3+1
44D8:AD 02 50 686 LDA TEMP3 ;TEMP3>=20?
44DB:C9 20 687 CMP #20
44DD:90 DF 688 BCC LASTOBJ2 ;NO, SO DO IT AGAIN
44DF:60 689 LASTOBJ3 RTS ;YES, SO RETURN TO MAIN LOOP
44E0: 690 #
44E0: 691 # WAIT
44E0: 692 #-----
44E0:38 693 WAIT SEC
44E1:AD 7F 50 694 LDA MAX ;ACCU=MAX-NUM
44E4:ED 7D 50 695 SBC NUM
44E7:F0 0B 696 P4 BEQ P4 ;IF MAX-NUM=0 THEN SKIP TO END
44E9:A2 FE 697 LDX #FE ;X=#FE
44EB:CA 698 PB DEX ;X=X-1
44EC:D0 FD 699 BNE PB ;IF X>0 THEN DO IT AGAIN
44EE:38 700 SEC ;ACCU=ACCU-1
44EF:E9 01 701 SBC #001
44F1:4C E7 44 702 JMP P4
44F4:60 703 P6 RTS ;RETURN

```

DOS Command Entry Points

By Mark Chatow
6510 Le Bec Place
Rancho Palos Verdes,
CA 90274

This disk protection method was developed as a backup to other protection methods. When a disk command is entered, it goes through the DOS keyboard input routine. Basically the routine checks to see if the entered command matches a standard command in memory. If it does, then the routine checks for any names or numbers that must accompany the command. When it is finished checking, the routine retrieves the starting address of the command from a table. It then jumps to this starting address. The command is then executed.

COMMAND CHANGING

For example, when the **LOAD** command is executed, the routine checks for the name of the program to be loaded. It then jumps to **A413(42003 in decimal)**. By changing the **A413** byte to **4C** you can jump to any location you wish, whenever the **LOAD** command is entered. Here is an example of how this could be used to protect a disk from unauthorized use: If the **A413** byte is changed to **4C** and then the starting address of the **delete** routine (**\$A263**) is given, you can change **LOAD** to **DELETE**. (Since the **RUN** command does a

LOAD and then goes to the starting address of the program loaded, you will automatically change the **RUN** command any time that you change the **LOAD** command.)

In the monitor the change would look like this; **A413:4C 63 A2**. Notice that the address must be entered in "least significant-most significant" byte order.

KEEP A LIST

As you can see, it is easy to change commands to suit your tastes if you know their starting addresses. I have included a list of starting addresses for the DOS commands. After several changes, further command changing can get confusing, so it is a good idea to keep an organized list of the commands you have changed and what you have changed them to.

Once the commands have been changed, you should insert a blank diskette in the disk drive. This is the disk that will have the protection on it. If the disk is likely to be used by others, then an appropriate "HELLO" program should be used. Your HELLO program might print something such as, "Warning.. this disk has been protected against unauthorized usage." This might scare the average user for fear of damaging the disk. It also serves as a reminder to you that the disk is protected. When you have finished the HELLO program, then initialize your disk. From then on the changes will be present every time you boot with that disk.

I hope that the protection method described here is as helpful for you as it is for me. If followed correctly, it should keep the average user from tampering with your disks.

COMMANDS ADDRESSES

COMMANDS	ADDRESSES
APPEND	A298
BLOAD	A35D
BRUN	A38E
BSAVE	A331
CATALOG	A56E
CHAIN	A4F0
CLOSE	A2EA
DELETE	A263
EXEC	A5C6
FP	A57A
INIT	A54F
IN#	A22E
INT	A59E
LOCK	A271
LOAD	A413
OPEN	A2A3
PR#	A229
RENAME	A281
RUN*	A413
SAVE	A397
UNLOCK	A275
VERIFY	A27D

*RUN does a **LOAD** then jumps to the starting location of the program you are running. Changing **LOAD** will also change **RUN**.

TRAC-I the Income System (Cont.)

```

9870 PRINT CHR$(9);"40N"
9872 PRINT CHR$(2); CHR$(30)
9875 A = 0: TA = 0
9880 PRINT TAB(11); "INCOME RECORDS $:" " " ; RMO; "/" ; R
DY; "/" ; RYR
9885 IF OM = 3 THEN PRINT TAB(7); "SORTED " ; A#
9890 PRINT : PRINT " " ; REFERENCE $ A# MM/DD AMT"

9895 FOR X = 1 TO 40: PRINT "=" ; NEXT X: PRINT
9900 PGCT = 0
9905 PGCT = PGCT + 1
9910 IF PRYPE = 1 AND PGCT > 16 THEN GOSUB 2800: HOME :
GOTO 9880: REM NEW PAGE
9915 A = A + 1: IF A > R THEN GOTO 9955
9920 PK = 1: V% = STR$(A); GOSUB 80
9925 PK = 5: V% = W$(A,6); GOSUB 80
9930 PK = 20: V% = W$(A,1); GOSUB 80
9935 PK = 24: V% = W$(A,3) + "/" + W$(A,4); GOSUB 80
9940 W = VAL(W$(A,5)); TA = TA + W
9945 P = W: GOSUB 2400: PK = 36 - C: V% = STR$(W); GOSUB 8
5
9950 GOTO 9905
9955 PRINT : PK = 22: V% = "TOTALS$"; GOSUB 80
9960 P = TA: GOSUB 2400: PK = 36 - C: V% = STR$(TA); GOSUB
85
9965 IF PRYPE = 1 THEN VTAB 24: INVERSE : INPUT "HIT RE
TURN FOR MAIN MENU"; X$: NORMAL : TEXT : GOTO 1000
9970 PRINT CHR$(9);"40N"
9975 PRINT D$;"PR#0"
9980 TEXT : GOTO 1000
10000 REM ***** INCOME PROFILE
10010 TEXT : FOR X = 30 TO 49: FOR N = 1 TO 12: BAL(X,N) =
0: NEXT N, X
10020 FOR X = 1 TO 12: YTD(X) = 0: NEXT X: REM USE FOR MO
NTH TOTALS
10030 SW = 1: GOTO 2500: REM ADD INCOME DETAIL
10040 SW = 4: GOTO 3000: REM LOAD ACCT NAMES
10050 IF PRYPE = 1 THEN GOSUB 12000: REM SCREEN PROFIL
E RPT
11000 REM *** PRINTER PROFILE REPORT
11010 PRINT D$;"PR#1"
11020 PRINT CHR$(9);"130N"
11025 IF PB% = "1" THEN PRINT CHR$(2); CHR$(31)
11030 IF PB% = "2" THEN PRINT CHR$(27); CHR$(20)
11035 IF PB% = "3" THEN PRINT CHR$(15)
11100 PK = 50: V% = "YEAR-TO-DATE INCOME PROFILE " + STR$
(RMO) + "/" + STR$(RDY) + "/" + STR$(RYR): GOSUB
85
11110 POKE 36,0: PRINT : PRINT "INCOME DESCR" ; TAB(10)
; MNAME$(1);
11120 FOR X = 2 TO 12: PRINT TAB(07); MNAME$(X); NEXT X
: PRINT
11130 FOR X = 1 TO 130: PRINT "=" ; NEXT X: PRINT

```

```

11200 FOR A = 30 TO 49
11210 IF ACCT$(A) = "" THEN Y = 0: FOR X1 = 1 TO 12: Y = B
AL(A,X1) + Y: NEXT X1: IF Y = 0 THEN GOTO 11280
11220 PK = 1: V% = ACCT$(A); GOSUB 80
11230 FOR X = 1 TO 12
11240 IF BAL(A,X) = 0 THEN GOTO 11270
11250 P = BAL(A,X): GOSUB 2400: PK = (X * 9) + 15 - C: V% =
STR$(BAL(A,X)); GOSUB 80
11260 YTD(X) = YTD(X) + BAL(A,X)
11270 NEXT X: PK = 0: SY = 0: PRINT
11280 NEXT A
11290 FOR X = 1 TO 130: PRINT "=" ; NEXT X: PRINT
11300 PK = 1: V% = "TOTALS$"; GOSUB 80
11310 FOR X = 1 TO 12: P = YTD(X): GOSUB 2400: PK = (X * 9)
+ 15 - C: V% = STR$(YTD(X)); GOSUB 80: NEXT X: PK =
0: SY = 0: PRINT
11335 IF PB% = "1" THEN PRINT CHR$(2); CHR$(30): REM
TURN OFF CONDENSED PRINT
11340 IF PB% = "2" THEN PRINT CHR$(27); CHR$(19)
11345 IF PB% = "3" THEN PRINT CHR$(146): REM TURN OF
F CONDENSED PRINT
11350 PRINT CHR$(9);"40N"
11360 PRINT D$;"PR#0"
11400 TEXT : GOTO 1000
12000 REM *** SCREEN PROFILE REPORT
12010 FOR X = 1 TO 12 STEP 3
12020 HOME : HTAB 8: PRINT "YTD INCOME PROFILE $": PRINT

12030 PRINT "INCOME DESC " ; MNAME$(X); " " ; MNAME$(
X + 1);
12040 FOR A = 30 TO 49
12050 IF ACCT$(A) = "" THEN Y = 0: FOR X1 = X TO (X + 2):
Y = Y + BAL(A,X1): NEXT X1: IF Y = 0 THEN GOTO 12130
12060 PK = 0: V% = LEFT$(ACCT$(A),12); GOSUB 80
12070 P = BAL(A,X): GOSUB 2400: PK = 16 - C: V% = STR$(BAL
(A,X)); GOSUB 80
12080 P = BAL(A,X + 1): GOSUB 2400: PK = 26 - C: V% = STR$(
BAL(A,X + 1)); GOSUB 80
12090 P = BAL(A,X + 2): GOSUB 2400: PK = 36 - C: V% = STR$(
BAL(A,X + 2)); GOSUB 85
12100 YTD(X) = YTD(X) + BAL(A,X)
12110 YTD(X + 1) = YTD(X + 1) + BAL(A,X + 1)
12120 YTD(X + 2) = YTD(X + 2) + BAL(A,X + 2)
12130 NEXT A
12200 PRINT " " ; PK = 0: V% = "TOTALS$"; GOSUB 80
12205 P = YTD(X): GOSUB 2400: PK = 16 - C: V% = STR$(YTD(X
)); GOSUB 80
12210 P = YTD(X + 1): GOSUB 2400: PK = 26 - C: V% = STR$(Y
TD(X + 1)); GOSUB 80
12220 P = YTD(X + 2): GOSUB 2400: PK = 36 - C: V% = STR$(Y
TD(X + 2)); GOSUB 85
12240 VTAB 23: INVERSE : PRINT "HIT ANY KEY TO CONTINUE";
: GET X$: NORMAL
12250 NEXT X
12300 TEXT : GOTO 1000
20000 DATA JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,D
EC

```

J

continued from page 85

Initializing Arrays with Specific Values (Cont.)

```

03AC D00B 185 BNE LABL5
03AE 186 ;
03AE 187 ; IS IT ANOTHER ARRAY TO INITIALIZE ?
03AE 188 ;
03AE 200700 189 JSR RELOAD
03B1 C92C 190 CMP #"/" ; USE THE SAME SPECIFIC VALUE
03B3 F0CB 191 BEQ LABL6
03B5 C9CB 192 CMP #SLASH ; USES ANOTHER SPECIFIC VALUE
03B7 F095 193 BEQ BEGINU
03B9 C929 194 CMP #' ' ; IT'S THE END
03BB F003 195 BEQ ENDU
03BD 4CC9DE 196 JMP ERROR
03C0 197 ;
03C0 198 ENDU:
03C0 4CB100 199 JMP CHRG ; RETURN TO BASIC..
03C3 200 ;
03C3 201 ; ROUTINE SEARCHING THE ADDRESS OF THE BEGINNING
03C3 202 ; AND THE END OF THE STORAGE AREA OF THE ARRAY
03C3 203 ;
03C3 204 SEARCH:
03C3 A004 205 LDY #004 ; LOAD THE DIMENSION NUMBER
03C5 B198 206 LDA (POINTL),Y
03C7 0A 207 RCL ; MULTIPLY IT BY TWO
03C8 18 208 CLC
03C9 6905 209 ADC #005 ; ADD FIVE BYTES
03CB 210 ;
03CB 211 ; - TWO FOR NAME
03CB 212 ; - TWO FOR OFFSET POINTER
03CB 6598 213 ADC POINTL ; - ONE FOR DIMENSION NUMBER
03CD 8594 214 STA ARRAYL
03CF C8 215 INV
03D0 A59C 216 LDA POINTH
03D2 6900 217 ADC #000
03D4 8595 218 STA ARRAYH
03D6 219 ;
03D6 18 220 CLC ; ADD OFFSET POINTER TO NEXT ARRAY
03D7 A002 221 LDY #002
03D9 B198 222 LDA (POINTL),Y
03DB 6598 223 ADC POINTL
03DD 8500 224 STA ENDST
03DF C8 225 INV
03E0 B198 226 LDA (POINTL),Y
03E2 659C 227 ADC POINTH
03E4 8501 228 STA ENDST+001
03E6 60 229 RTS
03E7 230 ;
03E7 231 ;
03E7 232 ;
03E7 233 STOP:

```

```

*****
INITIA DEMO
BY
P. FRANCOIS
COPYRIGHT (C) 1982
MICRO-SPARC, INC.
*****

```

JLIST

```

10 POKE 1013,76: POKE 1014,0: POKE 1015,3: REM INITIALIZATION OF AMPERSAND "&
"
20 TEXT : HOME : PRINT CHR$(4)"BLORD INITIA": REM CHR$(4)=CTRL D
30 VTAB 2: PRINT TAB(6);"ARRAYS BEFORE INITIALIZATION": PRINT
40 I$ = "AFTER": G% = 15: D = 99.999
50 DIM A$(3),B$(3),C$(3)
60 FOR I = 0 TO 3: A$(I) = "BEFORE": B$(I) = I + 1: C$(I) = I * 10.78: NEXT I
70 GOSUB 150
80 PRINT TAB(2);"ARRAYS AFTER ZERO=NULL INITIALIZATION": PRINT
90 & Z(A$,B$,C)
100 GOSUB 150
110 PRINT TAB(7);"ARRAYS AFTER INITIALIZATION": PRINT TAB(8);"WITH CHARACTER
ISTIC VALUE": PRINT
120 & U(I$(I),A$ / (G%),B$ / (D%),C)
130 GOSUB 150
140 END
150 FOR I = 0 TO 3: PRINT "A$(I)=",A$(I); "B$(I)=",B$(I); "C$(I)=",C$(I); "IC
(I) NEXT I: PRINT
160 RETURN

```

JRUN

ARRAYS BEFORE INITIALIZATION

```

A$(0)=BEFORE B$(0)=1 C(0)=0
A$(1)=BEFORE B$(1)=2 C(1)=10.78
A$(2)=BEFORE B$(2)=3 C(2)=21.56
A$(3)=BEFORE B$(3)=4 C(3)=32.34

```

ARRAYS AFTER ZERO=NULL INITIALIZATION

```

A$(0)= B$(0)=0 C(0)=0
A$(1)= B$(1)=0 C(1)=0
A$(2)= B$(2)=0 C(2)=0
A$(3)= B$(3)=0 C(3)=0

```

ARRAYS AFTER INITIALIZATION
WITH CHARACTERISTIC VALUE

```

A$(0)=AFTER B$(0)=15 C(0)=99.999
A$(1)=AFTER B$(1)=15 C(1)=99.999
A$(2)=AFTER B$(2)=15 C(2)=99.999
A$(3)=AFTER B$(3)=15 C(3)=99.999

```


APPLE CAL

The Time Machine (Cont.)

```

20100 FOR I = 1 TO 2 STEP 8
20110 VTAB 8: FOR J = 9 TO 28
20120 HTAB J: PRINT " ";
20130 FOR D = 1 TO L: NEXT D
20140 HTAB J: PRINT "*";
20150 NEXT J
20160 IF PEEK ( - 16384 ) > 127 THEN 20350
20170 FOR J = 8 TO 11
20180 VTAB J: HTAB 29: PRINT " ";
20190 FOR D = 1 TO L: NEXT D
20195 HTAB 29: PRINT "*";
20200 NEXT J
20210 IF PEEK ( - 16384 ) > 127 THEN 20350
20220 VTAB 12: FOR J = 29 TO 10 STEP - 1
20230 HTAB J: PRINT " ";
20240 FOR D = 1 TO L: NEXT D
20250 HTAB J: PRINT "*";
20260 NEXT J
20270 IF PEEK ( - 16384 ) > 127 THEN 20350
20280 FOR J = 12 TO 9 STEP - 1
20290 HTAB 9: VTAB J: PRINT " ";
20300 FOR D = 1 TO L: NEXT D
20310 HTAB 9: PRINT "*";
20320 NEXT J
20330 IF PEEK ( - 16384 ) > 127 THEN 20350
20340 NEXT I
20350 POKE - 16368,0
20360 PRINT CHR$ (7)
20370 RETURN
20400 REM ** FIX STACK ONERR **
20405 REM FROM PG 136, APPLE REF MANUAL
20410 A$ = "104168104166223154072152072096"
20420 FOR I = 1 TO 10
20430 POKE 767 + I, VAL ( MID$ (A$,I * 3 - 2,3))
20440 NEXT I
20450 RETURN

```



continued from page 87

Apple Checker 3.0 (Cont.)

```

0923:85 02 234 STA BEG+1
0925:4C 3A 09 235 JMP CONT
0928:B1 0A 236 INTCLR LDA (HLD),Y ;GET LENGTH
092A:18 237 CLC
092B:65 01 238 ADC BEG ;CALCULATE NEXT LENGHT ADDR
092D:85 01 239 STA BEG
092F:A5 02 240 LDA BEG+1
0931:69 00 241 ADC #0
0933:85 02 242 STA BEG+1
0935:98 243 STA
0936:91 0A 244 TYA (HLD),Y ;ZERO OUT LENGHT BYTE
0938:F0 CB 245 BEQ INTLP ;DO IT ALL AGAIN
093A: 246 #
093A:38 247 CONT SEC ;COMPUTE END-END-DECVL
093B:A5 03 248 LDA END
093D:E5 08 249 SBC DECVL
093F:85 03 250 STA END
0941:A5 04 251 LDA END+1
0943:E9 00 252 SBC #0
0945:85 04 253 STA END+1
0947:A5 07 254 LDA FTYPE ;IF BINARY-GO DO IT
0949:C9 04 255 CMP #4
094B:F0 0D 256 BEQ DOCKCD
094D:38 257 SEC ;COMPUTE LEN FOR BASIC FILES
094E:A5 03 258 LDA END ;LEN=END-BEG
0950:E5 01 259 SBC BEG
0952:85 05 260 STA LEN
0954:A5 04 261 LDA END+1
0956:E5 02 262 SBC BEG+1
0958:85 06 263 STA LEN+1
095A: 264 #
095A: 265 # DO CKCD ROUTINE
095A: 266 #
095A:A0 00 267 DOCKCD LDY #0 ;0 TO Y REG
095C:B1 01 268 LDA (BYTE),Y ;GET BYTE OF PROGRAM
095E:8D 00 04 269 STA LINE0 ;SHOW ACTIVITY
0961:A6 07 270 LDX FTYPE ;DETERMINE IF MUST IGNORE
0963:F0 0E 271 BEQ CHKINT ;IT'S INTEGER-IGNORE IF NECESSARY
0965:E0 04 272 CPX #4 ;IS IT BINARY?
0967:F0 2B 273 BEQ SUMIT ;YES-SO GO CODE IT
0969: 274 # IT'S APPLESOFT
0969:C9 21 275 CMP #21 ;CHAR<SPACE?
096B:B0 24 276 BGE SUMIT ;YES-SUMIT
096D:C9 04 277 CMP #4 ;CTL-D?
096F:F0 20 278 BEQ SUMIT ;YES-SUMIT
0971:D0 0C 279 BNE IGNCHR ;NO-MUST BE CTL-CHAR OR SPACE-IGNORE I
T
0973:C9 B0 280 CHKINT CMP #B0 ;CHAR<CTL-?
0975:90 1A 281 BLT SUMIT ;YES-SUMIT
0977:C9 A1 282 CMP #A1 ;CHAR>SPACE?

```

```

0979:B0 16 283 BGE SUMIT ;YES-SUMIT
097B:C9 B4 284 CMP #B4 ;CTL-D?
097D:F0 12 285 BEQ SUMIT ;YES-SUMIT
097F: 286 # IGNORE CHARACTER ROUTINE
097F:3B 287 IGNCHR SEC ;DEC LEN FOR EACH CHAR
IGNORED
0980:A5 05 288 LDA LEN
0982:E9 01 289 SBC #1
0984:85 05 290 STA LEN
0986:A5 06 291 LDA LEN+1
0988:F0 04 292 BEQ SPOUT
098A:E9 00 293 SBC #0
098C:85 06 294 STA LEN+1
098E:4C 9B 09 295 SPOUT JMP NXT ;CONTINUE
0991: 296 # BUILD CODE
0991:18 297 SUMIT CLC
0992:45 00 298 EOR CKCD ;DEVELOP CHECK CODE
0994:2A 299 ROL A
0995:65 00 300 ADC CKCD
0997:69 00 301 ADC #0
0999:85 00 302 STA CKCD
099B: 303 #
099B: 304 # ADJUST POINTER FOR NEXT PROGRAM BYTE
099B: 305 #
099B:18 306 NXT CLC ;CLEAR CARRY IND
099C:A5 01 307 LDA BYTE ;GET LO ADDR
099E:69 01 308 ADC #1 ;INCREMENT IT BY 1
09A0:85 01 309 STA BYTE ;STORE IT BACK
09A2:98 310 TYA ;0 TO A REG
09A3:65 02 311 ADC BYTE+1 ;INCREMENT HI ADDR WITH
CARRY (IF ANY)
09A5:85 02 312 STA BYTE+1 ;STORE IT IN HI ADDR
09A7: 313 #
09A7: 314 # CHECK FOR END OF PROGRAM
09A7: 315 #
09A7:18 316 CLC ;CLEAR CARRY IND
09A8:A5 02 317 LDA BYTE+1 ;COMPARE HI ADDR
09AA:C5 04 318 CMP END+1
09AC:90 AC 319 BCC DOCKCD ;BYTE<END
09AE:D0 07 320 BNE ALLDONE ;BYTE>END
09B0:18 321 CLC
09B1:A5 03 322 LDA END ;COMPARE LO ADDR
09B3:C5 01 323 CMP BYTE
09B5:B0 A3 324 BCS DOCKCD ;END<BYTE
09B7: 325 #
09B7: 326 # EXECUTION IS DONE
09B7: 327 #
09B7:A2 09 328 ALLDONE LDX #9 ;OUTPUT CONSTANT "LENGTH: "
09B9:BD 10 0A 329 LP2 LDA LENG,X
09BC:20 ED FD 330 JSR COUT
09BF:CA 331 DEX
09C0:10 F7 332 BPL LP2
09C2:A5 06 333 LDA LEN+1 ;GET HI BYTE OF LEN
09C4:A6 05 334 LDX LEN ;GET LO BYTE OF LEN
09C6:20 41 F9 335 JSR PRINTX ;OUTPUT LENGTH
09C9:A2 0A 336 LDX #0A ;OUTPUT "CHECKSUM: "
09CB:BD 1A 0A 337 LP3 LDA TOT,X
09CE:20 ED FD 338 JSR COUT
09D1:CA 339 DEX
09D2:10 F7 340 BPL LP3
09D4:A5 00 341 LDA CKCD ;GET CKCD TOTAL
09D6:20 DA FD 342 JSR PRBYTE ;OUTPUT CHECKSUM
09D9:20 BE FD 343 JSR CROUT
09DC:A9 A0 344 CLNUP LDA #A0 ;BLANK OUT ACTIVITY IND
09DE:BD 00 04 345 STA LINE0
09E1:A2 04 346 LDX #4
09E3:A9 08 347 LDA #8
09E5:81 E6 348 STA (PBEGH,X) ;RESET BIN LOAD ADDR
09E7:A0 19 349 OUT LDY #25 ;CLEAN UP PAGE ZERO FOR BASIC
09E9:A9 00 350 LDA #0
09EB:99 E0 00 351 ENDP STA PBEGL,Y
09EE:88 352 DEY
09EF:D0 FA 353 BNE ENDP
09F1:BD 01 0B 354 STA BGNASF
09F4:BD 02 0B 355 STA BGNASF+1
09F7:BD 03 0B 356 STA BGNASF+2 ;ZERO ASDFT NXT LINE ADDR
09FA:BD F6 B5 357 STA DOSFT ;0 TO FILE TYPE IN DOS
09FD:A9 01 358 LDA #1
09FF:85 F1 359 STA SPEED ;SET APPLESOFT SPEED TO MAX
0A01:A9 04 360 LDA #4 ;TERMINATE ASDFT PGM
0A03:85 AF 361 STA ASFTH
0A05:A5 4C 362 LDA INTN ;TERMINAT INTEGER PGM
0A07:85 CA 363 STA INTL
0A09:A5 4D 364 LDA INTH+1
0A0B:85 CB 365 STA INTL+1
0A0D:4C D0 03 366 JMP RTNBAS ;RETURN TO BASIC THROUGH DOS
0A10: 367 #
0A10: 368 #
0A10: 369 #
0A10:A0 BA CB 370 LENGT ASC " ;HTGNEL"
0A13:D4 C7 CE
0A16:C5 CC
0A18:BD BD 371 DFB #BD,#BD
0A1A:A0 BA CD 372 TOT ASC " ;MUSKCEHC"
0A1D:D5 D3 CB
0A20:C3 C5 CB
0A23:C3
0A24:BD 373 DFB #BD
0A25:A0 BA C5 374 CTYP ASC " ;EPYT"
0A2B:D0 D9 D4
0A2B:BD 375 DFB #BD
0A2C:A0 BA CE 376 INP ASC " ;NO"
0A2F:CF
0A30:BD BD BD 377 DFB #BD,#BD,#BD
0A33:BD
0A34:B0 AE B3 378 ASC "0.3 EDOC KCEHC"
0A37:A0 C5 C4
0A3A:CF C3 A0
0A3D:CB C3 C5
0A40:CB C3
0A42: 379 #
0A42: 380 # INDIRECT ADDRESSES
0A42: 381 #
0A42:CA 00 382 PDATA DW INTL
0A44:67 00 383 DW ASFTL
0A46:72 AA 384 DW BINL
0A48:CB 00 385 DW INTL+1
0A4A:68 00 386 DW ASFTL+1
0A4C:73 AA 387 DW BINL+1
0A4E:4C 00 388 DW INTN
0A50:AF 00 389 DW ASFTH
0A52:60 AA 390 DW BLEN
0A54:4D 00 391 DW INTN+1
0A56:B0 00 392 DW ASFTH+1
0A58:61 AA 393 DW BLEN+1
0A5A: 394 #

```

*** SUCCESSFUL ASSEMBLY: NO ERRORS



GO — A Greeting Program (Cont.)

```

340 VTAB I: HTAB I: INVERSE : PRINT B40%;: VTAB I: HTAB FN
    T(C): PRINT I%
350 VTAB P: FOR X = P TO V: HTAB Q: FOR C = P TO R: A$(X,C)
    ) = CHR$( FN S(R)): NEXT : PRINT : NEXT
360 VTAB P: FOR X = P TO V: HTAB Q: NORMAL : PRINT B3%;: HTAB
    R: INVERSE : PRINT CHR$( X + Y): NEXT :LL = X + 61:V
    = X + 1
370 REM FILE NAME/TYPE
380 VTAB V: HTAB H: CALL - B6B: POKE - 16368,0: INVERSE
    :N% = "":E = 39
390 HTAB H: PRINT "FILE CHOICE? (OR #=SPACE X=END) ": VTAB
    V: HTAB R: GET X$: IF X$ = R$ THEN 310
    IF X$ = "X" THEN HTAB H: CALL - B6B: PRINT "END": POKE
    44602,12: POKE 44603,253: POKE 44452,22: POKE 44605,2
    1: GOTO 640
410 IF X$ = "0" THEN 660
420 IF X$ < A$ OR ASC (X$) > LL THEN 390
430 LV = ASC (X$) - V: VTAB LV: HTAB I: S = PEEK (F0) + PEEK
    (F1) * F2
440 T = PEEK (S + 1): IF T < > 193 AND T < > 194 AND T <
    > 201 AND T < > 212 THEN VTAB V: GOTO 390
450 FOR X = E TO F STEP - 1: IF PEEK (S + X) < > B THEN
    E = X: GOTO 470
460 NEXT
470 FOR X = F TO E: N% = N% + CHR$( PEEK (S + X)): NEXT
    : VTAB LV: HTAB H: PRINT N%: VTAB V: HTAB H: CALL -
    B6B
480 REM COMMAND
490 HTAB H: IF T = 212 THEN PRINT "EXEC": GOTO 510
500 PRINT "RUN LOAD":
510 PRINT "TEND": IF T = 194 THEN PRINT " ADDRESS":
520 PRINT " SIZE":
530 VTAB V: HTAB R: GET X$: IF X$ = R$ THEN VTAB LV: HTAB
    H: NORMAL : PRINT N%: GOTO 380
540 IF X$ = A$ AND T = 194 THEN HTAB I: CALL - B6B: PRINT
    BL%: GOTO 1140
550 IF X$ = "S" THEN 710
560 IF X$ = "T" THEN 740
570 IF X$ = "E" AND T < > 212 OR X$ < > "E" AND T = 212
    THEN 530
580 IF X$ = "E" THEN D$ = "EXEC": GOTO 620
590 IF X$ = "R" THEN D$ = "RUN": GOTO 620
600 IF X$ < > "L" THEN 530
610 D$ = "LOAD"
620 POKE - 16298,0: POKE 44452,22: POKE 44605,21: POKE 4
    4602,12: POKE 44603,253: IF T = 194 THEN B1$ = "B"
630 HTAB H: CALL - B6B: PRINT B1%: PRINT D$B1%0%N%:
640 POKE - 16298,0: POKE 34,0: NORMAL : VTAB V - (X$ = "
    X" OR T = 193 OR T = 201): END
650 REM DISK SPACE
660 HTAB H: CALL - B6B: PRINT "DISK SPACE":
670 RESTORE : FOR X = 896 TO 960: READ A: POKE X,A: NEXT
    : POKE 905,149: POKE 938,149
680 J = PEEK (43626) * 161K = PEEK (43624): POKE 897,J: POKE
    898,K: POKE 911,J: POKE 912,K: CALL 919
690 FR = PEEK (917) + PEEK (918) * F2: US = 4% - FR: PRINT
    "USED="US" FREE="FR: HTAB R: GET X$: GOTO 310
700 REM SIZE
710 HTAB H: CALL - B6B: PRINT "SIZE": : FOR X = P TO R:
    PRINT A$(LV,X): NEXT : PRINT " SECTORS":
720 HTAB R: GET X$: CALL - B6B: GOTO 490
730 REM FILE TENDING
740 VTAB V: HTAB I: CALL - B6B: IF PEEK (S) = 106 THEN
    POKE S,170
750 HTAB H: PRINT "LOCK UNLOCK RENAME DELETE VERIFY": HTAB
    R: GET X$: IF X$ = R$ THEN CALL - B6B: HTAB H: GOTO
    490
760 IF X$ = "L" THEN 840
770 IF X$ = "U" THEN 870
780 IF X$ = "R" THEN 900
790 IF X$ = "D" THEN 990
800 IF X$ < > "V" THEN 750

```

```

810 REM VERIFY
820 CALL - B6B: HTAB H: PRINT "VERIFY": PRINT D$"VERIFY"
    N%: GOTO 740
830 REM LOCK
840 IF PEEK (S) = 170 THEN 750
850 CALL - B6B: HTAB H: PRINT "LOCK": PRINT D$"LOCK": POKE
    S,170: GOTO 740
860 REM UNLOCK
870 IF PEEK (S) = 160 THEN 750
880 CALL - B6B: HTAB H: PRINT "UNLOCK": PRINT D$"UNLOCK"
    N%: POKE S,160: GOTO 740
890 REM RENAME
900 IF PEEK (S) = 170 THEN POKE S,106
910 HTAB I: CALL - B6B: PRINT "RENAME": NORMAL : PRINT
    " ": GOSUB 50: INVERSE : IF I$ = N% THEN 740
920 HTAB I: PRINT BL$"OK?": NORMAL : PRINT " ": INVERSE
930 HTAB R: GET X$: IF X$ = "N" OR X$ = R$ THEN 740
940 IF X$ < > "Y" THEN 930
950 HTAB I: PRINT "RENAME": NORMAL : PRINT " ": IF PEEK
    (S) = 106 THEN UN = 1: PRINT D$"UNLOCK":N%
960 PRINT D$"RENAME":N%0%: IF UN THEN UN = 0: PRINT D$"L
    OCK":I$
970 N% = I$: VTAB LV: HTAB H: CALL - B6B: INVERSE : PRINT
    N%: GOTO 740
980 REM DELETE
990 HTAB I: CALL - B6B: PRINT "OK?": NORMAL : PRINT "
    ": INVERSE : PRINT BL$"DELETE":
1000 IF PEEK (S) = 170 THEN POKE S,106: PRINT " LOCKED
    FILE="BL%:
1010 HTAB R: GET X$: IF X$ = "N" OR X$ = R$ THEN 740
1020 IF X$ < > "Y" THEN 1010
1030 HTAB I: NORMAL : PRINT " ": IF PEEK (S) = 106 THEN
    PRINT D$"UNLOCK":N%
1040 PRINT D$"DELETE":N%: GOTO 310
1050 REM ERRORS
1060 POKE 768,104: POKE 769,168: POKE 770,104: POKE 771,1
    66: POKE 772,223: POKE 773,154: POKE 774,72
1070 POKE 775,152: POKE 776,72: POKE 777,96: CALL 768: POKE
    216,0: VTAB V: HTAB I: CALL - B6B
1080 ER = PEEK (222): VTAB V: HTAB H: INVERSE : PRINT BL$
    "ERROR "ER: IF ER = 1 THEN PRINT " - LANGUAGE NOT A
    VAILABLE":
1090 IF ER = 6 THEN PRINT " - FILE NOT FOUND":
1100 IF ER = 8 AND X$ = "V" THEN PRINT " - UNABLE TO VER
    IFY": GOTO 1120
1110 IF ER = 8 THEN PRINT " - I/O":
1120 HTAB R: GET X$: RUN
1130 REM ADDRESS/LENGTH
1140 HTAB I: PRINT "OK?": NORMAL : PRINT " ": INVERSE
    : PRINT "ADDRESS/LENGTH (BOOT IF FAILS)": HTAB R: GET
    X$
1150 IF X$ = "N" OR X$ = R$ THEN HTAB I: CALL - B6B: GOTO
    490
1160 IF X$ < > "Y" THEN 1140
1170 ONERR GOTO 1220
1180 HTAB I: NORMAL : PRINT " ": INVERSE : PRINT D$"BLD
    AD":N%
1190 LOMEM: PEEK (115) + PEEK (116) * 256 - 2000: H = 52
    16 + PEEK (115) + PEEK (116) * 256
1200 A = PEEK (43634) + PEEK (43635) * 256: L = PEEK (H)
    + PEEK (H + 1) * 256
1210 VTAB PEEK (37): HTAB B: CALL - B6B: PRINT "BLOAD A
    DDRESS="A" LENGTH="L:
1220 HTAB S: GET X$: CALL - B6B: HTAB B: PRINT "REQUIRED
    BOOT - PLEASE WAIT
1230 NORMAL : VTAB PEEK (37) - 1: PRINT CHR$(4)"PR#6
1240 DATA 1,96,1,0,17,0,145,3,0,149,0,0,1,0,0,96,1,0,1,23
    9,216,0,0,169,3,160,128,32,217,3,162,0,142,149,3,142
1250 DATA 150,3,160,56,185,0,149,162,8,10,144,8,238,149,3
    ,208,3,238,150,3,202,208,242,200,192,196,144,232,96
1260 REM BY CHRIS GLENN

```

Quick Sort (Cont.)

```

83AB-2406 CMP2 BIT RFLAG REALS ?
83AD-30F9 BMI CRLS BRANCH IF SO
83AF-B59B STA SARA
83B1-B49C STY SARA+1
83B3-701A BVS GNYU BRANCH IF STRINGS
83B5-A001 LDY #01
83B7-38 SEC
83B8-B19B LDA (SARA),Y
83BA-E59E SBC IST+1 COMPARE INTEGERS
83BC-AA TAX
83BD-B8 DEY
83BE-B19B LDA (SARA),Y
83C0-E59D SBC IST
83C2-D003 BNE ROA
83C4-8A TXA
83C5-F007 BEQ RTS3
83C7-6A ROA ROR A
83C8-459D EOR IST
83CA-519B EOR (SARA),Y
83CC-0901 ORA #01 CLEAR ZERO FLAG
83CE-60 RTS3 RTS
83CF-A002 GNYU LDY #02
83D1-B19B GNYU1 LDA (SARA),Y
83D3-999F00 STA AGE8-1,Y STORE STRING INFO
83D6-88 DEY IN AGE8
83D7-D0F8 BNE GNYU1
83D9-B19B LDA (SARA),Y CALCULATE MINIMUM
83DB-F019 BEQ CIST LENGTH
83DD-AA TAX
83DE-C59D CMP IST
83E0-9004 BCC CDS

```

```

83E2-A69D LDX IST
83E4-F014 BEQ ROE
83E6-B1A0 COS LDA (AGE8),Y COMPARE STRINGS
83E8-D19E CMP (IST+1),Y
83EA-900E BCC ROE
83EC-D00C BNE ROE
83EE-C8 INY
83EF-CA DEX
83F0-D0F4 BNE COS
83F2-A000 LDY #00
83F4-B19B LDA (SARA),Y
83F6-C59D CIST CMP IST
83F8-F014 BEQ RTS3
83FA-8A ROE ROR A
83FB-0901 ORA #01 CLEAR ZERO FLAG
83FD-60 RTS
*
* END OF PROGRAM
*

```

--- END ASSEMBLY ---

TOTAL ERRORS = 0
PROGRAM LENGTH = 574 BYTES

Applesoft Record Command (A.R.C.) (Cont.)

```

10 REM *****
20 REM $ INITIALIZE $
30 REM $ BY CHUCK KISS $
40 REM $ COPYRIGHT (C) 1982 $
50 REM $ BY MICRO-SPARC INC $
60 REM $ LINCOLN, MA. 01773 $
70 REM *****
80 D$ = CHR$ (4)
90 PRINT D$ "NOMON C,I,0"
100 DIM T$(16),TV(16),T(16),TW(16),C(16)
110 SPEED= 255: TEXT : HOME : X$ = "EXIT"
120 G$ = CHR$ (7): BC$ = CHR$ (2): V1$ = CHR$ (91): V2$ =
    CHR$ (93)
130 FOR I = 0 TO 16: C(I) = 0: NEXT I: V = 16: FOR K = 1 TO
    V: READ T$(K): NEXT
140 DATA "FULL NAME","LAST NAME","FIRST NAME","INITIAL",
    "ADDRESS","CITY","STATE","ZIPCODE","PHONE #","COMMENT",
    "OPTION 1","OPTION 2","OPTION 3","OPTION 4","OPTION 5",
    "OPTION 6"
150 IF DI = 1 THEN 220
160 VTAB 6: HTAB 11: INVERSE : PRINT "INITIALIZE NEW FILE"
    ": NORMAL : HTAB 11: PRINT "-----"
170 POKE - 16368,0
180 FOR D = 1 TO 1000: NEXT D
190 VTAB 13: HTAB 5: PRINT "NAME OF FILE"; G$: INPUT " ":
    : FILE$
200 IF LEN (FILE$) = 0 THEN 190
210 IF ASC ( LEFT$ (FILE$,1)) < 65 THEN VTAB 19: HTAB 1
    2: PRINT "ILLEGAL FILE NAME"; G$: FOR D = 1 TO 150
    0: NEXT D: VTAB 13: HTAB 19: CALL - 958: GOTO 190
220 HOME : INVERSE : HTAB (21 - LEN (FILE$) / 2): PRINT
    FILE$: NORMAL : PRINT : INVERSE
230 HTAB 14: PRINT " INITIALIZER ": NORMAL : PRINT : POKE
    34,4
240 HOME : PRINT " YOU WILL BE ENTERING DATA IN AN"
250 PRINT " ORDERED FORMAT CONSISTING OF FIELDS."
260 PRINT "SELECT A "; INVERSE : PRINT "MAXIMUM OF 12 FI
    ELDS";: NORMAL : PRINT " ": PRINT
270 FOR K = 1 TO V / 2: HTAB 2: PRINT K; ". "; T$(K);:
    (24 + (K - 1)): PRINT K + V / 2". "; T$(K + V / 2): NEXT
    K: PRINT
280 N = 0
290 N = N + 1: IF N > 12 THEN VTAB 20: HTAB 1: CALL - 95
    8: VTAB 23: HTAB 10: FLASH : PRINT " L I M I T = 1
    2 ": NORMAL : FOR D = 1 TO 2000: NEXT D: GOTO 460
300 VTAB 18: CALL - 958: PRINT "ENTER IN THE NUMBER REPR
    ESENTING YOUR CHOICE FOR "; INVERSE : PRINT "FIELD
    #"; N; G$: NORMAL : POKE - 16368,0
310 VTAB 23: HTAB 11: PRINT "PRESS "; INVERSE : PRINT "R
    ETURN";: NORMAL : PRINT " TO "; X$:; POKE 35,22
320 VTAB 19: HTAB (21 + (N - 9)): INPUT "=> "; T$: IF N >
    1 AND LEN (T$) = 0 THEN 460
330 IF N = 1 AND LEN (T$) = 0 THEN PRINT D$ "RUN A.R.C."
340 X$ = "QUIT"
350 IF ASC ( RIGHT$ (T$,1)) < 48 OR ASC ( RIGHT$ (T$,1)
    ) > 58 THEN 300
360 IF LEN (T$) > 2 THEN 300
370 T = VAL (T$): IF T < 1 OR T > V THEN 300
380 IF C(T) = 1 THEN VTAB 21: HTAB 12: INVERSE : PRINT "
    ALREADY SPECIFIED"; G$: NORMAL : FOR D = 1 TO 100
    0: NEXT D: HTAB 12: CALL - 868: VTAB 19: HTAB 23: CALL
    - 868: GOTO 310
390 IF T < 11 AND T > 0 THEN 440
400 IF T > 10 AND T < V + 1 THEN INVERSE : PRINT "ENTER
    NAME FOR THIS FIELD"; G$: HTAB 4: PRINT "(10 LETTERS
    OR LESS)";: NORMAL : VTAB 20: HTAB 27: INPUT " "; Z$: T$
    (T) = Z$
410 IF LEN (Z$) > 10 THEN VTAB 20: HTAB 28: CALL - 958
    : HTAB 29: INVERSE : PRINT "INVALID "; G$: G$: FLASH
420 IF LEN (Z$) > 10 THEN VTAB 21: HTAB 4: PRINT "(10 L
    ETTERS OR LESS)";: NORMAL : FOR D = 1 TO 1500: NEXT D:
    VTAB 20: HTAB 29: CALL - 868: HTAB 1: GOTO 400
430 IF LEN (Z$) = 0 THEN T$ (T) = " "
440 T(N) = T: VTAB (8 + T * (T < 9) + (T - 8) * (T > 8)): HTAB
    (6 + 23 * (T > 8)): PRINT " "; REM 10 SPACE
    S
450 HTAB (6 + 23 * (T > 8)): INVERSE : PRINT T$(T): NORMAL
    : C(T) = 1: GOTO 290
460 POKE 35,23: HOME : N = N - 1
470 IF INT (N / 2) = N / 2 THEN N2% = N / 2
480 IF INT (N / 2) < N / 2 THEN N2% = N / 2 + 1
490 FOR K = 1 TO N2%
500 PRINT : HTAB 4: PRINT K; ". "; T$(T(K));: HTAB (24 - ((
    K + N2%) > 9)): PRINT K + N2%; ". "; T$(T(K + N2%)): NEXT
    K
510 IF INT (N / 2) < N / 2 THEN VTAB (PEEK (37)): HTAB
    23: PRINT " ": REM 13 SPACES
520 VTAB 22: CALL - 868: HTAB 14: PRINT "CORRECT (Y/N)";
    : G$: GET Y$: IF Y$ = "N" THEN X$ = "EXIT": RESTORE
    : DI = 1: GOTO 130
530 IF Y$ < > "Y" THEN 520
540 POKE 34,0: HOME : HTAB 3: PRINT " ENTER LENGTH VALUES
    FOR EACH FIELD:"; HTAB 3
550 PRINT " -----": POKE 34,
    4
560 HTAB 2: VTAB 22: PRINT V1$;: ": INVERSE : PRINT "NOT
    E";: NORMAL : PRINT " "; V2$;: MAXIMUM FIELD LENGTH =
    20"
570 FOR K = 1 TO N
580 VTAB 4 + K: PRINT "FIELD #"; K; ". ....": INVERSE : HTAB
    16: PRINT T$(T(K)); G$: NORMAL : HTAB 31: INPUT " "; T
    V$
590 TV(K) = VAL (TV$)
600 IF K = 1 AND LEN (TV$) = 0 THEN POKE 34,0: RESTORE
    : DI = 1: X$ = "EXIT": GOTO 130
610 IF TV(K) < 1 OR TV(K) > 20 OR TV(K) < > INT (TV(K))
    THEN HTAB 29: VTAB 4 + K: FLASH : PRINT " INVALID "
    : FOR D = 1 TO 1000: NEXT D: VTAB 4 + K: HTAB 29: CALL
    - 868: K = K - 1: NORMAL : HTAB 1
620 NEXT K
630 FOR D = 1 TO 500: NEXT D
640 POKE 34,3: HOME

```

```

650 RL = 0: HTAB 5: PRINT "# FIELD NAME FIELD LENG
    H
660 HTAB 5: PRINT "- - - - -"
670 FOR K = 1 TO N
680 HTAB (5 - (K > 9)): PRINT K;: HTAB 9: INVERSE : PRINT
    " "; HTAB 9: PRINT T$(K);: NORMAL : HTAB
    (28 + (TV(K) < 10)): PRINT " "; TV(K); RL = RL + TV(K);
    NEXT K
690 HTAB 27: PRINT "-----": HTAB (29 - (RL > 99) + (RL <
    10)): INVERSE : PRINT RL;: NORMAL : HTAB 9: PRINT "RE
    CORD LENGTH =>";
700 RL = RL + 1
710 REM : CALC NUMBER OF RECORDS
720 NR = INT (6 + 12000 / (RL + 29))
730 IF NR > 125 THEN NR = 125
740 VTAB 21: HTAB 6: PRINT "FILE CAPACITY => "; INVERSE
    : PRINT NR;: NORMAL : PRINT " RECORDS"
750 VTAB 23: HTAB 14: CALL - 868: PRINT "CORRECT (Y/N) "
    : G$: GET Y$: IF Y$ = "N" THEN 540
760 IF Y$ < > "Y" THEN 750
770 POKE 34,0: HOME : FOR D = 1 TO 500: NEXT D: VTAB 11
780 HTAB 6: INVERSE : PRINT "INITIALIZING FILE ON DISKETT
    E": NORMAL
790 PRINT : HTAB 17: PRINT "STAND-BY"
800 FOR D = 1 TO 500: NEXT D
810 PRINT D$ "OPEN "; FILE$ + BC$
820 PRINT D$ "WRITE "; FILE$ + BC$
830 PRINT N: PRINT RL: PRINT NR: FOR K = 1 TO N: PRINT T$
    (T(K));: PRINT TV(K);: NEXT K
840 PRINT D$ "CLOSE "; FILE$ + BC$
850 PRINT D$ "LOCK "; FILE$ + BC$
860 PRINT D$ "OPEN "; FILE$, L, RL
870 PRINT D$ "WRITE "; FILE$, R0"
880 PRINT 0
890 PRINT D$ "CLOSE"
900 PRINT D$ "RUN A.R.C."
910 END

```

A.R.C. SEARCH

0302-	20 E3 DF	JSR	#DFE3	033B-	F0 1E	BEQ	#035B
0305-	85 06	STA	#06	033D-	C5 19	CMP	#19
0307-	84 07	STY	#07	033F-	90 1A	BCC	#035B
0309-	A0 00	LDY	#000	0341-	A5 19	LDA	#19
030B-	84 1A	STY	#1A	0343-	F0 16	BEQ	#035B
030D-	B1 06	LDA	(#06),Y	0345-	A5 18	LDA	#18
030F-	85 18	STA	#18	0347-	38	SEC	
0311-	C8	INY		034B-	E5 19	SBC	#19
0312-	B1 06	LDA	(#06),Y	034C-	85 18	STA	#18
0314-	48	PHA		034E-	A0 00	LDY	#000
0315-	C8	INY		034F-	B1 06	LDA	(#06),Y
0316-	B1 06	LDA	(#06),Y	0350-	D1 08	CMP	(#08),Y
0318-	85 07	STA	#07	0352-	D0 08	BNE	#035C
031A-	68	PLA		0354-	C8	INY	
031B-	85 06	STA	#06	0355-	C4 19	CPY	#19
031D-	20 BE DE	JSR	#DEBE	0357-	90 F5	BCC	#034E
0320-	20 E3 DF	JSR	#DFE3	0359-	E6 1A	INC	#1A
0323-	85 08	STA	#08	035B-	60	RTS	
0325-	84 09	STY	#09	035C-	E6 1A	INC	#1A
0327-	A0 00	LDY	#000	035E-	A5 18	LDA	#18
0329-	B1 08	LDA	(#08),Y	0360-	C5 1A	CMP	#1A
032B-	85 19	STA	#19	0362-	90 08	BCC	#036C
032D-	C8	INY		0364-	E6 06	INC	#06
032E-	B1 08	LDA	(#08),Y	0366-	D0 02	BNE	#036A
0330-	48	PHA		0368-	E6 07	INC	#07
0331-	C8	INY		036A-	D0 E0	BNE	#034C
0332-	B1 08	LDA	(#08),Y	036C-	A9 00	LDA	#000
0334-	85 09	STA	#09	036E-	85 1A	STA	#1A
0336-	68	PLA		0370-	F0 E9	BEQ	#035B
0337-	85 08	STA	#08				
0339-	A5 18	LDA	#18				

Disk Commander (Cont.)

```

8642: 706 *
8642: 707 * UNABBREVIATED COMMAND LIST
8642: 708 *
8642:CC CF C1 709 WCMD1 ASC "LOAD "
8645:C4 A0
8647:00 710 DFB $00
8648:D5 CE CC 711 WCMD2 ASC "UNLOCK "
8648:CF C3 CB
864E:A0
864F:00 712 DFB $00
8650:C4 C5 CC 713 WCMD3 ASC "DELETE "
8653:C5 D4 C5
8656:A0
8657:00 714 DFB $00
8658:D2 D5 CE 715 WCMD4 ASC "RUN "
8658:A0
865C:00 716 DFB $00
865D:C5 D8 C5 717 WCMD5 ASC "EXEC "
8660:C3 A0
8662:00 718 DFB $00
8663:C2 CC CF 719 WCMD6 ASC "BLOAD "
8666:C1 C4 A0
8669:00 720 DFB $00
866A:C2 D2 D5 721 WCMD7 ASC "BRUN "
866D:CE A0
866F:00 722 DFB $00
8670:CC CF C3 723 WCMD8 ASC "LOCK "
8673:CB A0
8675:00 724 DFB $00
8676:D3 C1 D6 725 WCMD9 ASC "SAVE "
8679:C5 A0
867B:00 726 DFB $00
867C:D2 C5 CE 727 WCMDA ASC "RENAME "
867F:C1 CD C5
8682:A0
8683:00 728 DFB $00
8684:C2 D3 C1 729 WCMD8 ASC "BSAVE "
8687:D6 C5 A0
868A:00 730 DFB $00
868B: 731 *
868B: 732 * CONSTANTS
868B: 733 *

```

```

868B:00 734 COUNT DFB $00
868C:00 735 COUNT1 DFB $00
868D:00 736 ALPHA DFB $00
868E:00 737 BETA DFB $00
868F: 738 *
868F: 739 * THE CONTROLLING SUBROUTINE
868F: 740 * PER THE DOS MANUAL
868F: 741 *
868F:A9 B6 742 CONTRL LDA #<IOB
8691:A0 97 743 LDY #>IOB
8693:20 B5 B7 744 JSR RWTS
8696:60 745 RTS
8697: 746 *
8697: 747 * THE INPUT/OUTPUT CONTROL BLOCK
8697: 748 * (IOB) AS PER THE DOS MANUAL
8697: 749 *
8697:01 750 IOB DFB $01; IOB TYPE INDICATOR
8698:60 751 IBLSLOT DFB $60; SLOT NUMBER TIMES 16
8699:01 752 IBDRVN DFB $01; DISK DRIVE NUMBER
869A:00 753 IBVOL DFB $00; EXPECTED VOLUME NUMBER
869B:00 754 IBTRK DFB $00; TRACK NUMBER
869C:00 755 IBSECT DFB $00; SECTOR NUMBER
869D:A8 B6 756 IBDCPT DW DCT ; ADDRESS OF DCT L.O. BYTE FIRST
869F:00 757 IBBUFP DFB $00; L.O. BYTE OF DATA BUFFER STARTING ADDR
86A0:20 758 DFB $20; H.O. BYTE OF DATA BUFFER STARTING ADDR
86A1:00 759 DFB $00; NOT USED
86A2:00 760 DFB $00; NOT USED
86A3:01 761 IBCMD DFB $01; READ/WRITE COMMAND CODE
86A4:00 762 IBSTAT DFB $00; ERROR CODE
86A5:00 763 IBSPDN DFB $00; ACTUAL VOLUME NUMBER
86A6:60 764 IOBPSN DFB $60; PREVIOUS SLOT ACCESSED
86A7:01 765 IOBPDN DFB $01; PREVIOUS DRIVE ACCESSED
86AB: 766 *
86AB: 767 * THE DEVICE CHARACTERISTICS TABLE
86AB: 768 * (DCT) AS PER THE DOS MANUAL
86AB: 769 *
86AB:00 770 DCT DFB $00; DEVICE TYPE CODE
86A9:01 771 DFB $01; NUMBER OF PHASES PER TRACK
86AA:EF 772 DFB $EF; TIME COUNT
86AB:D8 773 DFB $D8; TIME COUNT
86AC: 774 ADDRA EQU *
8700: 775 BUFP EQU HOOKS+$500

```

*** SUCCESSFUL ASSEMBLY: NO ERRORS



Electronic Message Center (E.M.C.) (Cont.)

```

4200 NEXT
4210 AS = FIS: IF CH = 0 THEN 4240
4220 PRINT CHR$(4)"DELETE "AS: PRINT CHR$(4)"OPEN "AS:
PRINT CHR$(4)"WRITE "AS
4230 FOR I = 1 TO MC - 1: PRINT MES(I): PRINT MD$(I): NEXT
: PRINT CHR$(4)"CLOSE"
4240 VTAB 15: CALL - 958: PRINT "CHANGE ANOTHER": INPUT
YNS: IF LEFT$(YNS,1) = "Y" THEN HOME :CH = 0: GOTO 4000
4250 RETURN
4260 VTAB 15: CALL - 958: PRINT "DATE (MM/DD/YY) :": FOR
J = 1 TO 8: PRINT CHR$(95): NEXT : VTAB 15: HTAB 19: INP
UT ""MD$(I)
4270 IF LEN(MD$(I)) > 8 THEN VTAB 7: CALL - 958: GOTO 4
260
4280 AS = MES(I): GOTO 4110
4290 VTAB 13: CALL - 958: PRINT "MESSAGE :": VTAB 15: FOR
J = 1 TO 224: PRINT CHR$(95): NEXT : VTAB 15: GOSUB 8000
4300 IF LEN(ST$) = 1 THEN AS = MES(I): GOTO 4320
4310 MES(I) = ST$:AS = ST$
4320 GOTO 4110
4330 VTAB 17: CALL - 958: PRINT "ENTER 'DEL' TO CONFIRM :
": INPUT ""YNS: IF YNS < > "DEL" THEN 4110
IF I = MC THEN MC = MC - 1: GOTO 4200
4350 FOR J = 1 TO MC - 1
4360 MES(J) = MES(J + 1):MD$(J) = MD$(J + 1): NEXT
4370 MC = MC - 1
4380 AS = MES(I)
4390 IF I > = MC THEN 4180
4400 GOTO 4110
5000 REM ::DELETE FILE::
5010 HOME : VTAB 12: PRINT "ENTER FILE TO BE DELETED :": P
RINT : HTAB 10: FOR I = 1 TO 10: PRINT CHR$(95): NEXT : P
RINT : VTAB 14: HTAB 10: INPUT ""AS
5020 IF AS = "" THEN RETURN
5030 IF LEN(AS) > 10 THEN 5010
5040 GOSUB 15000
5050 FOR I = 1 TO CZ - 1: IF AS = TIS(I) THEN 5090
5060 NEXT
5070 HOME : VTAB 12: PRINT "UNABLE TO MATCH THAT NAME.": P
RINT : PRINT "TRY AGAIN ?": INPUT ""YNS: IF LEFT$(YNS,1)
= "Y" THEN 5000
5080 RETURN
5090 PRINT : PRINT "ENTER 'DEL' TO DELETE FILE.": PRINT :
HTAB 10: INPUT YNS: IF YNS < > "DEL" THEN RETURN
5100 HOME : VTAB 12: HTAB 5: INVERSE : PRINT "FILE BEING DE
LETED.": NORMAL
5110 IF I = CZ - 1 AND I = 1 THEN PRINT CHR$(4)"DELETE T
ITLES": PRINT CHR$(4)"DELETE "AS: RETURN
5120 FOR J = 1 TO CZ - 1:TIS(J) = TIS(J + 1): NEXT
5130 PRINT CHR$(4)"DELETE TITLES": PRINT CHR$(4)"OPEN T
ITLES": PRINT CHR$(4)"WRITE TITLES": FOR I = 1 TO CZ - 1:
PRINT TIS(I): NEXT : PRINT CHR$(4)"CLOSE"
5140 PRINT CHR$(4)"DELETE "AS
5150 PRINT : PRINT "FILE DELETED, DO ANOTHER ?": INPUT ""
YNS: IF LEFT$(YNS,1) = "Y" THEN 5000
5160 RETURN

```

```

6000 REM ::VIEW NAMES::
6010 GOSUB 15000
6020 B = 1: HOME
6030 FOR I = B TO CZ - 1
6040 IF I < 10 THEN HTAB 2
6050 IF PEEK(37) < 22 THEN PRINT I". "TIS(I): GOTO 6070
6060 GOTO 6090
6070 NEXT
6080 GET AS: PRINT AS: RETURN
6090 J = 1: I = CZ - 1: IF B = 1 THEN 6150
6100 VTAB 2
6110 FOR I = J TO CZ - 1
6120 HTAB 16: IF PEEK(37) < 22 THEN PRINT I". "TIS(I): G
OTO 6140
6130 GOTO 6150
6140 NEXT : HTAB 16: GET AS: PRINT AS: RETURN
6150 HTAB 16: GET AS: PRINT AS
6160 B = 1: I = CZ - 1: IF B = 1 THEN 6140
6170 HOME : GOTO 6030
7000 TEXT : HOME : END
8000 REM ::INPUT ROUTINE::
8010 ST$ = "": CALL - 662
8020 FOR J = 512 TO 736: X = PEEK(J)
8030 IF X = 141 THEN 8060
8040 ST$ = ST$ + CHR$(X - 128)
8050 NEXT : IF LEN(ST$) > 224 THEN PRINT : PRINT CHR$(
7)" TOO LONG!": FOR J = 1 TO 3000: NEXT : HOME : POP : GOTO
3070
8060 IF ST$ = "" THEN POP : HOME : GOTO 3070
8070 RETURN
9000 REM ::READ MESSAGES::
9010 MC = 1
9020 PRINT CHR$(4)"OPEN "AS: PRINT CHR$(4)"READ "AS
9030 ST$ = ""
9040 CALL - 662
9050 FOR I = 512 TO 736: X = PEEK(I): IF X = 141 THEN 9090
9060 IF X > 141 THEN X = X - 128
9070 ST$ = ST$ + CHR$(X)
9080 NEXT
9090 MES(MC) = ST$: INPUT MD$(MC):MC = MC + 1
9100 GOTO 9030
9110 PRINT CHR$(4)"CLOSE"
9120 RETURN
10000 REM ::PRINT FORMATTER
10010 IF LEN(AS) < = RM - LM THEN PRINT AS: RETURN
10020 FOR J = RM - LM TO 1 STEP - 1
10030 CS = MID$(AS,J,1): IF CS = " " OR CS = "." OR CS = "-"
OR CS = "!" OR CS = "?" OR CS = "=" THEN 10050
10040 NEXT : J = RM - LM
10050 IF CS = " " THEN A1$ = LEFT$(AS,J - 1):AS = RIGHT$(
AS, LEN(AS) - J): GOTO 10090
10060 IF CS = "!" OR CS = "." OR CS = "-" OR CS = "?" OR CS =
"=" AND J = RM - LM THEN GOSUB 10130:A1$ = LEFT$(AS,J)
:AS = RIGHT$(AS, LEN(AS) - J - 1): HTAB LM: PRINT A1$: GO
TO 10100
10070 A1$ = LEFT$(AS,J): IF LEN(AS) < > LEN(A1$) THEN
AS = RIGHT$(AS, LEN(AS) - J): GOTO 10090
10080 GOTO 10110
10090 HTAB LM: PRINT A1$
10100 IF LEN(AS) > RM - LM THEN 10020
10110 HTAB LM: PRINT AS
10120 RETURN
10130 IF J = LEN(AS) THEN RETURN

```

continued on next page

Electronic Message Center (E.M.C.) (Cont.)

```

10140 IF MID$(A$,J+1,1) < > " " THEN A1$ = LEFT$(A$,
      J):A$ = RIGHT$(A$, LEN(A$) - J): HTAB LM: PRINT A1$: POP
      : GOTO 10100
10150 RETURN
11000 PRINT TAB((32) - LEN(A$)) / 2;A$: RETURN
12000 DATA "1. LOAD & DISPLAY FILE","2. ADD TO A FILE
      ", "3. ADD A NEW FILE", "4. CHANGE A FILE", "5. DEL
      ETE A FILE", "6. VIEW NAME-FILE", "7. EXIT PROGRAM
      "
15000 REM ::::READ TITLES FILE:::
15010 CZ = 1
15020 PRINT CHR$(4)"OPEN TITLES": PRINT CHR$(4)"READ TI
      TLES"
15030 INPUT T1$(CZ): IF T1$(CZ) = "****" THEN 15050
15040 CZ = CZ + 1: GOTO 15030
15050 PRINT CHR$(4)"CLOSE"
15060 RETURN
32000 REM ::::ERROR ROUTINE:::
32010 ER = PEEK(222):EL = PEEK(218) + PEEK(219) * 256
32020 CALL 768
32030 IF ER = 5 AND EL = 15030 AND A < > 3 THEN PRINT : P
      RINT CHR$(4)"CLOSE": HOME : VTAB 12:A$ = "NO TITLES IN FIL
      E.": GOSUB 11000: FOR I = 1 TO 4000: NEXT : POP : RETURN
32040 IF ER = 5 AND EL = 15030 THEN PRINT : PRINT CHR$(4
      )"CLOSE":ER = 1: GOTO 15060
32050 IF ER = 5 AND EL = 9040 OR EL = 9100 THEN PRINT : GO
      TO 9110
32060 IF ER = 22 AND EL > 10000 THEN RESUME
32070 IF ER = 255 THEN SPEED= 255: GOTO 130
32080 HOME
32090 IF ER = 9 THEN PRINT "DISK FULL!": END
32100 IF ER = 10 THEN PRINT "FILE IS LOCKED!": END
32110 IF ER = 8 THEN PRINT "DISK I/O ERROR!": END
32120 IF ER = 4 THEN PRINT "REMOVE WRITE PROTECT TAB!": EN
      D

```

continued from page 158

The Shape (Cont.)

```

2370 POKE 33,40: HOME
2380 PRINT TAB(33)"ERASE": POKE 33,30
2390 COLOR= 0
2400 HOME : GOTO 480
2410 REM
      CENTER SUB
2420 HTAB 21 - LEN(A$) / 2: PRINT A$: RETURN
2430 REM
      POKE CALCS
2440 VTAB 19: PRINT "START HEX NUMBERS WITH A '$!'"
2450 VTAB 17: INPUT "WHERE WILL YOU PUT THE TABLE ";X$
2460 Y = VAL(X$)
2470 IF LEFT$(X$,1) = "$" THEN X$ = RIGHT$(X$, LEN(X$)
      - 1): GOSUB 2540
2480 X = Y - INT(Y / 256) * 256:Z = INT(Y / 256)
2490 HOME : PRINT "USE THESE POKES IN YOUR PROGRAM BEFORE
      YOU DRAW YOUR FIRST SHAPE"
2500 PRINT : PRINT "POKE 232,";X: PRINT "POKE 233,";Z
2510 PRINT
2520 END
2530 REM
      HEX CONVERTER
2540 Y = 0:Z = 0
2550 FOR I = LEN(X$) - 1 TO 0 STEP - 1
2560 Z = Z + 1
2570 Y1 = ASC(MID$(X$,Z,1)) - 48
2580 IF Y1 > 16 THEN Y1 = Y1 - 7
2590 Y = Y + Y1 * 16 ^ I
2600 NEXT
2610 RETURN
2620 REM
      TURTLE MODE
2630 GR : POKE 34,20: HOME : PRINT "MOVE DOT TO START & PRE
      SS 'P!': POKE 34,21
2640 Z = 1:X = 20:Y = X:XX = Y:YY = Y:TM = 1
2650 COLOR= 0: PLOT XX,YY: COLOR= 15: PLOT X,Y:XX = X:YY =
      Y: HOME : PRINT "X=";X;" Y=";Y: GET A$
2660 IF A$ = "P" THEN SX = X:SY = Y: GOTO 2720
2670 IF A$ = "I" AND Y > 0 THEN Y = Y - 1
2680 IF A$ = "M" AND Y < 39 THEN Y = Y + 1
2690 IF A$ = "K" AND X < 39 THEN X = X + 1
2700 IF A$ = "J" AND X > 0 THEN X = X - 1
2710 GOTO 2650
2720 P = 1:ZC = 0: POKE 34,20
2730 XX = X:YY = Y: HOME : PRINT "X=";X;" Y=";Y
2740 IF P = 1 THEN PRINT "PLOT ON": GOTO 2760
2750 PRINT "PLOT OFF"
2760 GET A$
2770 IF A$ = "M" AND Y < 39 THEN G(Z) = 2:Z = Z + 1:F = 0:Y
      = Y + 1: GOTO 2870
2780 IF A$ = "J" AND X > 0 THEN G(Z) = 3:Z = Z + 1:F = 0:X
      = X - 1: GOTO 2870
2790 IF A$ = "K" AND X < 39 THEN G(Z) = 1:Z = Z + 1:F = 0:X
      = X + 1: GOTO 2870

```

```

2800 IF A$ = "I" AND Y > 0 AND (F = 0 OR P = 1) THEN G(Z) =
      0:F = 1:Z = Z + 1:Y = Y - 1: GOTO 2870
2810 IF A$ = "?" THEN 3110
2820 IF A$ = "E" THEN P = 0: GOTO 2730
2830 IF A$ = "P" THEN P = 1: GOTO 2730
2840 IF A$ = "Q" THEN G(Z) = 10: COLOR= ZC: PLOT X,Y: GOTO
      1650
2850 IF A$ = CHR$(27) THEN 2900: REM ESC KEY
2860 GOTO 2730
2870 CZ = SCRN(X,Y): COLOR= ZC: PLOT XX,YY: COLOR= 15: PLO
      T X,Y:ZC = CZ
2880 IF P = 1 THEN G(Z - 1) = G(Z - 1) + 4: COLOR= 1: PLOT
      XX,YY
2890 GOTO 2730
2900 IF Z > 0 THEN Z = Z - 1
2910 IF Z < = 0 THEN 2730
2920 ON G(Z) + 1 GOSUB 2940,2950,2960,2970,2940,2950,2960,2
      970
2930 COLOR= 15: PLOT X,Y: COLOR= 0: PLOT XX,YY: GOTO 2730
2940 Y = Y + 1: RETURN
2950 X = X - 1: RETURN
2960 Y = Y - 1: RETURN
2970 X = X + 1: RETURN
2980 REM
      LINE MODE HELP
2990 POKE 33,40: HOME
3000 PRINT "V-> DRAW VERTICAL LINES": PRINT "H-> DRAW HOR
      IZONTAL LINES": PRINT "F-> ENTER FREE CURSOR MODE": INPUT "
      PRESS RETURN";A$: HOME
3010 PRINT "DRAW LINES BY GIVING THE ENDPOINTS AND": PRINT
      "LEVEL TO DRAW ON": INPUT "P;PRESS RETURN";A$: HOME
3020 PRINT "P-> TURNS PLOT ON (DRAW LINES)": PRINT "E-> TUR
      NS PLOT OFF (ERASE LINES)": INPUT "PRESS RETURN ";A$: HOME
3030 PRINT "Q-> QUIT DRAWING AND CODE THE SHAPE": INPUT "P
      RESS RETURN";A$: HOME : GOTO 2330
3040 REM
      FREE CUR HELP
3050 HOME
3060 PRINT "I,J,K,M KEYS MOVE THE CURSOR": PRINT "P-> PLOT
      A POINT": PRINT "E-> ERASE A POINT": INPUT "PRESS RETURN";
      A$
3070 HOME
3080 PRINT "P & E ONLY EFFECT THE POINT THE CURSOR ": PRINT
      "I'S RESTING ON, NO OTHERS!": INPUT "PRESS RETURN";A$: HOME
3090 PRINT "Q-> RETURNS YOU TO THE HLIN MODE WITH": PRINT
      "THE PLOT ON": INPUT "PRESS RETURN";A$: HOME
3100 GOTO 1000
3110 REM
      TURTLE MODE HELP
3120 HOME : PRINT "I,J,K,M KEYS MOVE THE CURSOR": PRINT "P-
      > TURN PLOTTING ON": PRINT "E-> TURN PLOTTING OFF": INPUT
      "PRESS RETURN";A$: HOME
3130 PRINT "THE ESC KEY WILL ALLOW YOU TO 'BACK UP!": PRINT
      "AND ERASE ERRORS": INPUT "PRESS RETURN";A$: HOME
3140 PRINT "Q-> QUIT DRAWING AND CODE THE SHAPE": INPUT "P
      RESS RETURN";A$: HOME : GOTO 2730

```

continued from page 176

Apple Recipe Box (Cont.)

```

9820 DIM RD(16),RI(25),L%(23),LS(16)
9830 DIM C$(15),FA$(15),RE$(15)
9840 CS = - 958:BL = - 1054
9850 D$ = CHR$(4):CL$ = D$ + "CLOSE":OP$ = D$ + "OPEN"
9860 IX$ = "INDEX.FILE.L60":RD$ = D$ + "READ":RF$ = "REC
      IPE.FILE"
9870 RO$ = "OPEN.RECORDS":LO$ = ",L4"
9880 WR$ = D$ + "WRITE"
9890 LF$ = ",L45":L$ = ",L60"
9100 N = 0:PP% = 1:PI% = 1:NH = 9:NDC% = 1
9110 DT$ = "....."
9120 ONRR GOTO 7000
9130 DATA APPETIZERS/PICKLES/RELISH,SOUPS/SALADS/SA
      UCES/DRESSINGS,MAIN DISHES (MEAT/FISH/POULTRY),MAIN
      DISHES (DAIRY/CASSEROLES)
9140 DATA VEGETABLES,BREAD/ROLLS/PIES/PASTRY,DESSERTS,J
      ELLY/JAM/PRESERVES,BEVERAGES
9150 FOR X = 1 TO NH: READ HD$(X): NEXT
9160 DATA 162,0,32,117,253,160,2,138,145,105,200,169,0,
      145,105,200,169,2,145,105,189,0,2,41,127,157,0,2,20
      2,224,255,208,243,96
9170 FOR J = 800 TO 833: READ I: POKE J,I: NEXT
9180 DATA 2/8,4/8,6/8,2/4,2/6,4/6,2/16,4/16,6/16,8/16,
      10/16,12/16,14/16
9190 DATA 1/4,1/2,3/4,1/2,1/3,2/3,1/8,1/4,3/8,1/2,5/8,
      3/4,7/8

```



```

9200 FOR X = 1 TO 13: READ RE$(X): NEXT
9210 FOR X = 1 TO 13: READ FA$(X): NEXT
9220 PRINT D$;"RENAME";IX$;",";IX$: PRINT
9230 PRINT OP$;IX$;L$
9240 PRINT RD$;IX$;",";R0"
9250 INPUT N: INPUT NC%
9260 IF N = 0 THEN GOTO 9280
9270 FOR X = 1 TO N: PRINT RD$;IX$;",";R":X: INPUT TR$(X)
: CALL 800:NR$(X) = MID$(CV$,1): NEXT
9280 PRINT CL$;IX$
9290 X = PEEK (37): REM SET ADDRESSES FOR SCREEN LINES

9300 FOR I = 0 TO 23: POKE 37,I: CALL - 990
9310 LX(I) = 256 * ( PEEK (41)) + PEEK (40): NEXT
9320 POKE 37,X: CALL - 990: GOTO 30

```

continued from page 179

Apple Disk Doctor (Cont.)

```

3120 IF XU = 1 THEN POKE (BUF + XY), PEEK (BUF +
XY + 32): POKE (BUF + XY + 32),160: POKE ICMD,WR:
CALL RWTS,XY = 221:XX = 1: IF PEEK (IERR) < > 0
THEN GOSUB 11000: RETURN
3130 IF PEEK (BUF + XY) + PEEK (BUF + XY + 1) =
0 THEN XY = 221:XX = 1
3140 NEXT : NEXT
3150 VTAB 18: HTAB 5
3160 IF XU = 1 THEN PRINT "FILE RECOVERED IN VTO
C": VTAB 20: HTAB 5: PRINT "REMEMBER TO LOAD AND S
AVE IT"
3170 IF XU = 0 THEN PRINT " FILE NOT FOUND"
3180 FOR XX = 1 TO 1500: NEXT
3190 RETURN
3500 IF LEN (S$) < 30 THEN S$ = S$ + MID$ ("
",1,30 - LEN (S$)): RE
M 30 SPACES
3510 XU = 1
3520 FOR XZ = 1 TO 29
3530 IF PEEK (BUF + XY + XZ + 2) < > ASC ( MID
$(S$,XZ,1)) + 128 THEN XU = 0:XZ = 30
3540 NEXT
3550 RETURN
4000 REM REMOVE DOS
4010 HOME : VTAB 7
4020 PRINT " IF DOS IS REMOVED YOU CAN NOT BOOT T
HE": PRINT
4030 PRINT " DISK. DO YOU WANT TO CONTINUE? (Y/N
)"
4040 FOR XX = 1 TO 2 STEP 0
4050 VTAB 9: HTAB 39: INPUT "N";S$
4060 IF S$ = "N" OR S$ = "Y" THEN XX = 2
4070 NEXT
4080 IF S$ = "N" THEN RETURN
4090 POKE ITRK,17: POKE ISEC,0
4100 POKE ICMD,RD: CALL RWTS: REM READ THE VTOC
4110 IF PEEK (IERR) < > 0 THEN GOSUB 11000: RE
TURN
4120 POKE BUF + 56,0: POKE BUF + 57,0: REM TRK
0 NOT AVAILABLE
4130 FOR XX = 60 TO 64 STEP 4: POKE BUF + XX,255:
POKE BUF + XX + 1,255: NEXT
4140 POKE ICMD,WR: CALL RWTS
4150 IF PEEK (IERR) < > 0 THEN GOSUB 11000: RE
TURN
4160 FOR XX = 1 TO 118: POKE BUF + XX - 1,NDB(XX)
: NEXT
4170 FOR XX = 118 TO 255: POKE BUF + XX,0: NEXT
4180 POKE ITRK,0: POKE ISEC,0
4190 CALL RWTS: REM WRITE BOOT SECTOR
4200 IF PEEK (IERR) < > 0 THEN GOSUB 11000
4210 RETURN
5000 HOME : VTAB 6
5010 HTAB 10: PRINT "CURRENT SLOT:"; PEEK (IBSL)
/ 16: PRINT
5020 HTAB 9: PRINT "CURRENT DRIVE:"; PEEK (IDRV):
PRINT
5030 FOR XX = 1 TO 2 STEP 0: VTAB 12: HTAB 14: IN
PUT "NEW SLOT:";S$: IF S$ > "0" AND S$ < "8" THEN
XX = 2
5040 NEXT
5050 POKE IBSL, VAL (S$) * 16
5060 FOR XX = 1 TO 2 STEP 0: VTAB 14: HTAB 13: IN
PUT "NEW DRIVE:";S$: IF S$ > "0" AND S$ < "3" THEN
XX = 2

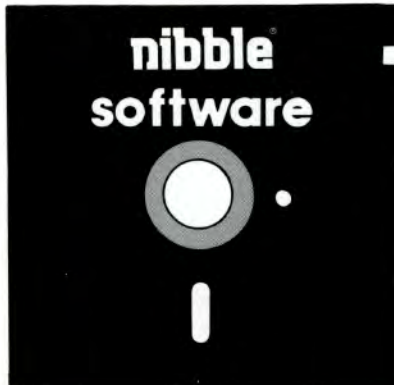
```

```

5070 NEXT
5080 POKE IDRV, VAL (S$)
5090 RETURN
6000 I = 2: RETURN
10000 REM ***** SETUP *****
10010 REM INSTALL INTERFACES
10020 FOR XX = 770 TO 833: READ XY: POKE XX,XY: N
EXT
10030 CALL 770: REM GET IOB ADDR
10040 IOB = PEEK (6) + ( PEEK (7) * 256)
10050 IBSL = IOB + 1
10060 IDRV = IOB + 2
10070 IVOL = IOB + 3
10080 ITRK = IOB + 4
10090 ISEC = IOB + 5
10100 ICMD = IOB + 12
10110 IERR = IOB + 13
10120 IAVOL = IOB + 14
10130 BUF = PEEK (IOB + 8) + ( PEEK (IOB + 9) * 2
56)
10140 POKE IVOL,0: REM CLEAR VOLUME ID
10150 RWTS = 780
10160 RD = 1:WR = 2
10170 TEXT : HOME
10180 PRINT " * * * * * DISK DOCTOR * * * * *
": POKE 34,2
10190 PRBYTE = 800
10200 ESCF = 64578
10210 BYTE = 8
10220 ZAP = 810:BDISP = 29: REM ZAP VARIABLES
10230 POKE 30, PEEK (IOB + 8): POKE 31, PEEK (IOB
+ 9)
10240 DIM NDB(118)
10250 FOR XX = 1 TO 118: READ NDB(XX): NEXT
10260 DIM ZTBL(19)
10270 FOR I = 0 TO 19: READ ZTBL(I): NEXT
10280 RETURN
10290 REM RWTS INTERFACE
10300 DATA 32,227,3,133,7,132,6,96,234,234
10310 DATA 165,7,164,6,32,217,3,176,6
10320 DATA 169,0,160,13,145,6,96
10330 DATA 234,234,234,234
10340 REM MONITOR PRBYTE INTERFACE
10350 DATA 165,8,32,218,253,96
10360 REM NIBBLE ZAP ROUTINE
10370 DATA 234,234,234,234,162,240,165,8,16
10380 DATA 8,10,10,10,10,133,8,162,15,138
10390 DATA 164,29,49,30,5,8,145,30,96
10400 REM BOOT ROUTINE WHEN DOS REMOVED
10410 DATA 1,157,136,192,142,244,3,138,74
10420 DATA 74,74,74,9,192,141,45,8,165
10430 DATA 0,141,44,8,32,96,251,162,0
10440 DATA 189,46,8,240,6,32,240,253,232
10450 DATA 208,245,32,111,253,108,44,8,0
10460 DATA 0,141,141,160,160,160,160,160,160
10470 DATA 160,160,160,206,207,160,196,207,211
10480 DATA 160,207,206,160,212,200,201,211,160
10490 DATA 196,201,211,203,141,141,160,160,201
10500 DATA 206,211,197,210,212,160,196,201,211
10510 DATA 203,160,215,201,212,200,160,196,207
10520 DATA 211,160,193,206,196,160,208,210,197
10530 DATA 211,211,160,210,197,212,213,210,206
10540 DATA 0
10550 REM
10560 REM TAB TABLE FOR ZAP
10570 REM
10580 DATA 30,30,31,31,0,32,32,33,33,0,34,34,35,
35,0,36,36,37,37,0
10970 REM
10980 REM ---- DISK ERROR HANDLER ----
10990 REM
11000 HOME :XX = PEEK (IERR)
11010 IF XX = 16 THEN S$ = " DISK IS WRITE PROTEC
TED "
11020 IF XX = 32 THEN S$ = " VOLUME MISMATCH
"
11030 IF XX = 64 THEN S$ = " I/O ERROR
"
11040 IF XX = 128 THEN S$ = " READ ERROR
"
11050 INVERSE : VTAB 7: HTAB 8: PRINT SPC( 25):
VTAB 8: HTAB 8: PRINT S$: VTAB 9: HTAB 8: PRINT S
PC( 25)
11060 FOR XX = 1 TO 2500: NEXT
11070 NORMAL
11080 RETURN

```


nibble® software



APPLE TRAC PLUS/SPACE MAZE...
\$24.95 + Shipping
Featured in Vol. #1, Issue #1

Apple TRAC Plus is an easy-to-use, adaptable system for analyzing "Where your Money Goes." TRAC Plus accepts information about your credit cards, checks, and cash transactions and prints more than 10 different reports for Budget Spending Analysis. Examine Year-To-Date Spending, Trends, Comparisons to Average Spending. (TRAC Plus is the core program for a complete Personal Finance system with its companion programs TRAC Budget, TRAC Graphics, and Income TRAC (offered as separate disks in this ad).
(48K AS)

Space Maze is a Hi-Res Graphics game in which the player pilots his ship through a Maze using the Game Paddle controls (or a joystick). Includes a number of sound options and two levels of play. A challenge for all! **(16K AS)**

STAR ATTACK/AIRSEA BATTLE/LOW RES SHAPEWRITER...
\$19.95 + Shipping
Featured in Vol. #1, Issue #2

Star Attack is a very fast, Hi-Resolution Graphics Gallery game. It combines Assembly Language and either Applesoft or Integer Basic for fast challenging action. Seven ships scan across the screen making random appearances, sometimes at random speeds. Some are Hostile Klingon and Tie Battle ships. Others are friendly... Hospital ship, Communications Satellite and more. The trick is to zap the bad guys and avoid hitting your friends... and it's not easy! Runs under game paddle control.
(32K AS/IB)

Airsea Battle combines Assembly Language and Integer Basic to produce a Low-Resolution Graphics battle between the Navy and the Air Force. You can choose sides (in either the Two-Player game or the One-Player vs. the Apple). Shapes, Explosions, and Splashes heighten the Graphics effect. **(16K IB)**

Low Resolution Shapewriter is a combined Assembly-Language and Integer Basic program for creating and running high speed Low Resolution Shapes in your programs. You create shapes with Up, Down, Right, Left, and Plot/No Plot commands from the keyboard. Instructions are given for saving shapes and then integrating them into your own programs. **(16K IB)**

APPLE TEXT PROCESSOR (T.O.U.G.H.)/APPLE SIMON/TIGER GRAPHICS... \$24.95 + Shipping
Featured in Vol. #1, Issue #3

Apple TOUGH (Text Outputter, Updater, and Generalized Handler) transforms your Apple into a line-oriented Text Processor. It is very easy to use and allows: • Upper/Lower Case characters; • Right Margin Justification; • Tab/Indent; • Variable Line Length; • Horizontal Page Centering.

Editing Features include: • Review; • Erase/Retype Lines; • Insert Lines; • Shift Paragraphs; • Find/Replace words and Print Text. Operates with Parallel or Serial Printer (and incorporates features of Epson MX-80, IDS series, and Centronics 737-739). Requires 1 disk drive. **(48K AS)**

Apple Simon is a Low Resolution Graphics version of the popular game. Combines Color and Sound to test your coordination and memory! It's fun! **(16K IB)**

Tiger Graphics is an Applesoft and Machine Language program for printing the High Resolution Graphics screen to the IDS 440/460 Paper Tiger Printers. Program controls allow printing with Black-on-White or White-on-Black. Requires the printer to be equipped with the Graphics Option. **(32K AS)**

FAST/SUPER WEAVER/FOOTBALL...
\$24.95 + Shipping
Featured in Vol. #1, Issue #4

F.A.S.T. (Forecasting and Sales Trending) is a Simulation and Modeling Tool for testing different strategies in selling multiple products and services. Prompts the user to build a productivity model which is then used to test different Hiring Rates, Pricing, Product Emphasis and Product Introduction Timing. Four different reports project Unit and Dollar Sales. **(16K AS)**

Super Weaver simulates an 8 Harness Loom in High Resolution Graphics. Simple keyboard commands allow the weaver to enter the threading, tie-up, and treadling of the Loom to produce a woven pattern on the screen. The finalized pattern can then be printed as a Draw-Down on a Centronics compatible parallel printer (it is written for the IDS 440 printer). **(16K AS)**

Football is a Low Resolution Graphics simulation of the popular hand-held game. Two players alternately choose Running, Passing, or Kicking plays in order to score touchdowns and field goals. **(16K IB)**

PIP I/I AM BUDDHA/TAPE LOAD AIDS... \$24.95 + Shipping
Featured in Vol. #1, Issue #5

PIP I is a Tape-based Personal Inventory Program which provides data base management for inventorying and tracking your personal assets. PIP I provides reports on Depreciation Costs, decreasing Replacement Values, Overall Asset analysis, and more. Optional Serial/Parallel printer. **(32K AS)**

I Am Buddha is an interactive game which produces surprising answers to your What, Who, Where, Why, When, and How questions. A little oracle! **(16K AS)**

Tape Loading Aids are two utilities which assist an accurate loading of programs and data from tape cassettes. Provides audible confirmation of beginning and end

of Load. The second provides Auto Re-Try if errors are encountered. **(16K ML)**

PIP II/AMPER-INTERPRETER/FOUR-IN-A-ROW... \$29.95 + Shipping
Featured in Vol. #1, Issue #6

PIP II (Personal Inventory Program) provides a complete **disk-based** data management system for managing and tracking your personal assets. It is a disk version of PIP I (described above). Uses optional Serial/Parallel printer. **(48K AS)**

Amper-Interpreter Utilities provides an extremely powerful Utility Library for your Apple. It uses the Ampersand Key to emulate the PRINT, USING command for formatting Decimal, Binary, Floating Point, and Hexadecimal Results. Provides Hooks to also: • Clear Screen; • Clear Page; • Bell; • Clear Error; and • execute ONERR patch. **(48K ML)**

Four-In-A-Row is an interactive Low Resolution Graphics game of skill and strategy. You'll try to beat your opponent to a Four-In-A-Row line of blocks to win. Watch the Force of Gravity drop your blocks through the grid as each play struggles for a Win! **(48K IB)**

APPLE A.I.M./CONCORDANCE/LOW SCORE II... \$29.95 + Shipping
Featured in Vol. #1, Issue #7

Apple A.I.M. (Automated Intelligent Mailing) is a highly modular system for managing Mail Lists and other Filing/Retrieval/Listing applications. With A.I.M., you can define up to 32 different Fields of Information in each record. Rearrange and Select special Fields for formatting/printing. Includes options for: • Formatting; • Adding; • Deleting; • Editing; • Finding; • Sorting; and • Printing the stored information. You can print Mailing Labels directly from your disk file! Uses a serial or parallel printer. **(48K AS)**

Apple Concordance is a powerful Utility which aids your programming! Automatically prepares lists of Where your Program Variables are Used and how they're Referenced. Also prepares a Cross-Reference of Program Line Numbers and where they are used. **(32K ML)**

Low Score II is a Hi-Res graphics game of Skill and Strategy. Try to Beat the Apple to the Lowest Score to Win! Uses a library of shapes which can be applied to your own programs! **(32K AS)**

WILL 'O THE WISP/BLAST AWAY LOAN REDUCTION ANALYSIS...
\$19.95 + Shipping
Featured in Vol. #1, Issue #8

Will 'O The Wisp is a complete Adventure-type game that takes you through forests, caves, and castles to the den of Ralph, the Magician. You follow the Will 'O the Wisp (which sometimes leads you correctly and other times leads you into dangerous turf). Illustrates the principles involved with writing your own Adventure games. **(48K AS)**

Blast Away! is a fast action shoot 'em up in Low Resolution graphics. As each window of the old ghost house lights up you have to quickly aim and fire before the ghost moves on. Illustrates a number of game writing principles. Uses joystick or game paddles. **(16K AS)**



Loan Reduction Analysis is a useful application for testing alternate financing methods for your major purchases. Simple screen prompts let you test Interest, Changes in Interest, Term of Loan, etc. Produces a screen full of resulting payments (while varying the principal and interest). (16K AS)

APPLE CHAMP/AMPER READER/ APPLE COMMUNICATOR...

\$29.95 + Shipping
Featured in Vol. #2, Issue #1

Apple CHAMP (Checkbook Advanced Management Program) is a versatile data entry and checkbook reconciliation system. CHAMP allows for record Addition, Editing, Search/Change, Posting, and Bank Reconciliation. It maintains automatic running bank balances and facilitates monthly checkbook balancing. Requires 1 disk. (48K AS)

Amper-Reader is an Assembly Language program which speeds disk file access of sequential text files by 5-10 Times Normal Speed. Dramatically Reduces DOS Overhead for reading large files. Optional Automatic Record Blocking for large files, i.e. you can read in 10 or 20 or 100 records at a time from files containing many more records (in order to conserve memory space). Disk contains Amper-Reader and demonstration program. Requires 1 Disk. (48K ML)

Apple Communicator is a basic demonstration program for assembling words and sentences using the Game I/O Connector. Under Game Paddle Control, the cursor scans across the Screen-displayed Alphabet. Pressing the button picks a letter to add to words being formed at the bottom of the screen. Adaptable to a variety of input/control methods for communication with people who may be handicapped and unable to speak or type. (16K IB)

MUSIC RETRIEVAL SYSTEM (M.R.S.)/LE MANS/MORSE CODE SYSTEM...

\$19.95 + Shipping

Featured in Vol. #2, Issue #2

Music Retrieval System (M.R.S.) is a fast, flexible file management system for storing, cataloging, and retrieving information

on your record collection. Sort by title, composer, etc. M.R.S. is readily adaptable to other storage/retrieval tasks by changing the data names. (48K AS)

Le Mans is a very fast racing game in Low Resolution graphics. One or Two Players compete by guiding the race car(s) along twisting, narrowing roads. Watch out for crashes and make it safely through the passes to win. (16K IB)

Morse Code System generates Morse Code sight and sound for learning Digits, Alpha, and Special Characters. Demonstrates the use of the DOS EXEC Commands for file control and automatic Apple operation. (16K IB)

APPLE PAINTBOX/MAD MAD CUBE/LINEFINDER...

\$24.95 + Shipping

Featured in Vol. #2, Issue #3

Apple Paintbox is a High Resolution Graphics program which combines the functions of Etch-A-Sketch, Spirograph, and a Function-Driven Drawing Board. Automatic Color Selection, Lines, Rectangles, Curves, and Auto-Color-Fill-In make it an unusual graphics drawing aid. Optional Disk for saving Pictures. (16K AS)

The Mad Mad Mad Mad Cube is a Low Resolution Graphics version of the popular 3x3x3 plastic Cube Game. After setting the colors of the faces of the cube, the blocks of 9 cubes can be rotated Horizontally and Vertically. After several rotations, the challenge is to return to the original color combination. Optional Disk for saving Cube configurations. (24K AS)

Applesoft Linefinder is a powerful Utility for locating individual Applesoft program lines in memory. Use for Appending Forbidden Line Numbers, Repairing Programs, Improving Program Efficiency and more. Specifying the Target Line searches for the Start/End memory position. Also produces information on Program Start/End, Hi-Mem, Strings, Free-Space, Variables and Arrays. (48K AS)

APPLE STAR SYSTEM/TRAP'EM/ BOOT 3.2/3.3...

\$29.95 + Shipping

Featured in Vol. #2, Issue #4

Apple Star System is an extremely powerful and flexible system for data file management. The program is generally self-prompting, and allows the user to enter and delete data, as well as to display entries and modify them as needed. Deleted records may be restored quickly and easily. Other features include data output to a printer and single field sorting. Requires 1 disk drive. (48K AS)

Trap'em is an addictive character graphics game that combines quick reflexes with quick thinking. You may choose to play against up to 3 deadly beasts which stalk you as you attempt to trap them while staying out of their fatal reach. Trap'em is written entirely in Applesoft. (16K AS)

Boot 3.2/3.3 is a machine language utility which allows you to boot thirteen sector diskettes under DOS 3.3 without the inconvenience of using the Basics disk or running "BOOT 13". The program modifies the data write and sector search routines of the RWTS. Boot 3.2/3.3 does not destroy the 3.3 sector header, so the diskette may be easily restored. (48K ML)

APPLE DISK MASTER/APPLE TRAP- STEP/BIORHYTHMS...

\$29.95 + Shipping

Featured in Vol. #2, Issue #5

Apple Disk Master is a multi-featured DOS utility program for the Apple. It allows: Recovery of Deleted Files; Direct Track/Sector Examination; Catalog Sorting; and a variety of other functions. It is adaptable between DOS 3.2 and 3.3 with the change of a single memory location. (48K ML)

Apple Trap/Step is a powerful Applesoft Program Development Aid. It allows single stepping through the execution of an Applesoft program to examine program logic. It also allows the placement and removal of "Traps" throughout the program to examine the values of specific variables without disrupting the program flow. (48K ML)

Apple Biorhythms is a versatile implementation of the popular Biorhythm principle. It produces Printed Bio-Charts as well as Low Resolution Graphics display. Now you can take your Bio-Charts with you to watch for critical days. For IDS or EPSON printers. (48K, Printer, AS)



APPLE M.A.P.S./T-SORT/ARTIST...

\$29.95 + Shipping

Featured in Vol. #2, Issue #6

Apple MAPS is a Comprehensive Marketing and Personal Shopping system. It helps keep track of your stock of Groceries and other Items; Flags the Need for Buying Stocks; and even Prints a pre-sorted Shopping List! Learn What You Have and What to Get! Keeps running totals on all products. Easy Editing of Grocery files. (48K AS)

Amper-Jump and T-Sort is a fast Machine Language Utility for Sorting Primary and Index Fields of arrays simultaneously! This gives the sequence of a Sorted Field and the Index to the Physical Location of each Record in the File! The utility uses less than 2K of memory and sorts arrays of any size. (48K ML)

Apple Artist is a Hi-Res Graphics Picture Maker which puts you in Command of the Screen. Your Keyboard controls the direction and includes functions for drawing Boxes, Fill-Ins, Lines, and More. Pictures may be saved to disk. (16K AS)

nibble® software

APPLE DART/PRINT-USE/LASER BLASTER... \$29.95 + Shipping Featured in Vol. #2, Issue #7

Apple D.A.R.T. is a Home Finance System which allows you to compare Monthly Expenditures. It projects Future Expenditures and Debts for 15 Debt Accounts, 10 Asset Accounts, and 5 Miscellaneous Accounts. It allows you to evaluate and plan your Financial Position. Project your Net Worth! Test new Investment and Debt Strategies. Show the past year's history and the coming year's projections for any combination of accounts. (48K AS)

Amper-Print-Use is a powerful Machine Language Utility for Fast Formatting of Alphabetic and Numeric Information. It simulates the PRINT...USING instruction found on other larger computers and handles Column Headers, Decimal, Scientific notation, Left and Right Justification of Strings... and more. Numbers may be rounded and aligned with dollar signs, commas and/or exponents. Instantly accessible from your Applesoft Programs. (48K ML)

Laser Blaster is a Hi-Resolution Graphics Action Game in outer space. Asteroids and Aliens are on a Collision Course with your Space Station. Rotate your Laser Turret and Fire before they HIT! (16K AS)



FINANCIER III/POOR BOY'S LE/CATSUP/NFL RANKER... \$19.95 + Shipping

Featured in Vol. #2, Issue #7

Financier III is a complete Apple Financier package, with direct reduction loan calculation, amortization, two types of depreciation, simple interest calculation and interest through compounding. (32K AS)

Poor Boy's LE is a program editing aid that is transparent to the user's program. Features include start and stop list, fast cursor advance, screen clear, shorten line length, and last BLOAD address and length. (48K ML)

Apple CATSUP is a HELLO program which will catalog a disk and provide a shorthand method for issuing DOS commands. All commands are presented in menu format and may be invoked by using one or two keystrokes. (48K AS)

NFL Ratings System is a quantitative program that measures the relative strengths of all the NFL teams and updates itself as the season progresses. The program will predict the correct winner of pro football games with 70 to 75 percent accuracy. (48K AS)



TRAC "B"/TRAC GRAPH/ARCHIVES/TIC-TAC-TOE... \$29.95 + Shipping Featured in Vol. #2, Issue #8

TRAC "B" (Budget TRAC) is a Flexible, Fast Budget account Preparation and Reporting System for your Personal Finances. Working with Apple TRAC (offered elsewhere in this ad) TRAC "B" allows Manual or Automatic Budget preparation using information from your original TRAC files. Allows Budget/Actual comparisons, Individual Account % of Total, and Over/Under Calculation. Uses the screen and/or optional Serial or Parallel Printer. (48K AS. Recommended: TRAC Plus)

TRAC GRAPH plots your Finances in Hi-Resolution Graphics for each Finances Category. Takes information directly from the TRAC Files and Charts it for Review. A Picture is worth Many Numbers! (48K AS. Required: TRAC Plus)

Apple Archives is a super-fast Machine Language program for Automatic Transfer of Disk Files to and from Tape Cassette. Allows entry of Header Information Record to Tape. Backup your Disk Files automatically and Inexpensively with Tape copies! (48K ML)

Three-D Tic-Tac-Toe is a Lo-Res Graphics version of the popular multi-level board game. Fast Action and Automatic Scoring! (16K AS)

HEXPAD/DOS REMOVER/CMD COMMAND CHANGER/FORMAT/X-Y PLOT... \$19.95 + Shipping

Featured in Vol. #2, Issue #8

Apple Hexpad is a new "Phantom" Keypad for Fast, Easy Entry of Machine Language programs into your Apple. It transforms a portion of your keyboard into a true HEX-PAD. Features automatic Forward/Backward 8 position spacing! (16K AS)

DOS Remover physically expunges DOS from Tracks 1 and 2 of your Diskettes.

Frees an additional 8K of disk storage for those Large Files. (48K AS)

Command Changer lets you physically Change and Shorten your DOS commands to your own taste. Great for Customized programs! (48K AS)

Formatter is a Routine for Controlling the Format of Data Entry for your screen applications! Professional looking screens are a snap. (16K AS)

X-Y Plot provides simple parameters for Plotting Data with the Apple Tool Kit. (32K AS + Toolkit)

TURTLE GRAPHICS/TRAC "I"/CRUNCHER... \$29.95 + Shipping Featured in Vol. #3, Issue #1

Apple Turtle Graphics is a pseudo LOGO-type language for Guiding your Hi-Resolution Turtle around the Screen! You write a series of simple Instructions for Directing the Turtle to Move, Turn, and Draw... and it follows your every command! The program is also a study in the development of new pseudo-languages. (16K AS)

TRAC "I" (Income TRAC) adds a Third Major Module to the TRAC Home Finance System. It is an easy-to-use system for monitoring your Income from various sources. Generates Reports for Year-to-Date Income Profiles (and Monthly Income Statements... using data from TRAC Budget and TRAC Files shown elsewhere in this ad). Shows Current vs. Average Months and Income Summaries. (48K AS. Recommended: TRAC Plus)

Applesoft Line Cruncher is a powerful Utility for compressing Multiple Applesoft Program Lines into Single, Multi-Statement Programs. Speeds up your programs and reduces the amount of Memory required to contain your Applesoft programs. (48K ML)

SCRAMBLE/CHECKER/SCROLL/X-REF/FREE SPACE... \$19.95 + Shipping Featured in Vol. #3, Issue #1

Apple Scramble is a Hi-Res version of the Popular \$.49 Letter/Tiles Game. Arrange the Letters in Alphabetic Order by sliding the Tile this way and that. Great for learning game logic and control while you test your manipulative skills. (48K AS)

Apple Checker is an important Utility for comparing your typing of Nibble Programs with Checksum totals which are published in Nibble. Checker Checksums are published for the major programs in each issue. (48K ML)

Lo-Res Hyper-Scroll provides very Fast Four-Way Scrolling of your Screens. A very handy technique for Spectacular Screen Displays. (48K ML)

Variable Cross Referencer allows you to Extract, Sort and Alphabetize all of your Applesoft Variables along with the Program Line Numbers where they are used. (48K AS)

Amper Free Space uses the Ampersand to trigger a useful Disk Free Space indicator. Now you can find out how much disk space is left, Any Time... without loading other programs! (48K ML)

APPLE MLE/LAMP/APPLE CAL...

\$29.95 + Shipping
Featured in Vol. #3, Issue #2

Apple MLE (Machine Language Editor) is a powerful and unique Utility for Entering and Editing Machine Language Programs. It allows dynamic Insert, Delete, and Xchanging of Machine Code directly in Memory. Includes full Disk Command set for retrieving and saving files. NO MORE RETYPING of Machine Language programs to correct minor errors. A Must in your Library! **(48K ML)**

LAMP is a useful Utility program which contains the following features: • Decimal-to-Hex Conversion; • List parameters of Last BLOADED or BRUN program; • Find the location of an Applesoft Line in Memory; • ASCII Conversion; • Locate Specified Program spaces in Memory; and • Display Free Sectors on Disk Drive 1 or 2. **(48K ML)**

Apple CAL allows you to schedule Events, Holidays, and Appointment Days, Months or even Years in advance. Automatically prints Month Calendars for Any Month you Specify for any year between 1981 and the year 2000. Retrieves key Holidays through 1987! Easy and Handy! **(48K AS)**

APPLE SLUGGER/PIECHART/AMPER-FIND/SORT...

\$19.95 + Shipping
Featured in Vol. #3, Issue #2

Apple Slugger is a Hi-Res Action Baseball Game. Pitch, Bat, and Field in your own home! The Routines can be used in your own game development! **(16K AS)**

Apple Piechart is a set of routines which use the Apple Toolkit for Hi-Res Pie Charting with Color, Labels, and Highlighting! **(Toolkit, 16K AS)**

Amper-Find is a Fast Machine Language Utility for Searching Out Variables, Strings, and Commands in your Applesoft Programs. Great for checking out Variable Name usage! **(16K ML)**

Speed Sort will Speed Up your String Sorting by a factor of 60. Handles One and Two Dimension Arrays in a Flash. Includes instructions for appending the machine language to your Basic program for automatic loading. **(48K ML)**

APPLE BOND MANAGER/DOS 3+2/DARTS...

\$29.95 + Shipping
Featured in Vol. #3, Issue #3

Apple Bond Manager is a comprehensive program for Analyzing Alternate Bond Investments. You enter information about Bonds and your own Financial position and Bond Manager produces Comparative Gross Yields, Capital Gains Yields, Current Yields, and After Tax Yields. Bond Records can be stored, retrieved, and edited from disk as needed. **(48K AS)**

DOS 3+2 is the perfect Utility for fast Switching between Apple's DOS 3.2 and 3.3. It provides for Co-Resident 3.2 and 3.3 and is selectable from within any program with a single POKE. Any Disk Read Error will automatically Switch to the "Other" DOS version to try reading again! Read and Write on 13 or 16 Sector Disks interchangeably! **(48K ML)**

Apple Darts is a challenging Hi-Res Graphics Dart Game. Two Levels of Play and spectacular Hi-Res displays heighten the fun. It takes skill and quick timing to Win at Darts! **(16K AS)**



AMPERSPEED/LAMP II/PROTECTOR/3-D MAP...

\$22.95 + Shipping
Featured in Vol. #3, Issue #3

Amperspeed is a very fast Utility program for Reading and Writing Sequential Text Files. Improves the disk access speed by nearly 3 to 1! **(48K ML)**

LAMP-II works with LAMP-I (offered elsewhere in this ad) to add more utility functions to the LAMP Library. It includes a utility for placing itself or Any Program between DOS and its Buffers. Adds 3 Letter Commands for Cataloging and displaying Free Sectors on Disk Drive 1 or 2. **(48K ML)**

Program List Protector is a Utility Processor for preventing your Applesoft programs from being Listed or Run by unauthorized people. Simple to Use. Keys the program access to your own Code-Word. **(48K AS/ML)**

3-D MAP is a demonstration program for displaying 3 Dimensional Landscapes using the Apple Toolkit. Describes the methods used for converting Two to Three Dimension Displays. **(16K AS + Toolkit)**

MICRO-CALC/VARIABLE CRUNCHER/LIFE...

\$29.95 + Shipping
Featured in Vol. #3, Issue #4

Micro-Calc generates your own, customized Finance programs for Home and Business. You set up your own Accounting Spreadsheet on the Screen. You specify Data, Comments, and Formulas for up to 20 Columns and 20 Rows. You can examine the results with Left and Right screen Scrolling. Then Micro-Calc takes over and literally Creates your own Applesoft program to do the job! **(48K AS)**

Applesoft Variable Cruncher is a powerful Utility for Compressing your Variable Names to One or Two letters. Don't be afraid to waste memory writing Long,

Descriptive Variable Names. Cruncher will shrink them down to size. Your programs will save memory and run faster. **(48K ML)**

Apple LIFE is a fast, Hi-Resolution Graphics program which allows you to set up your own Colony of Life Cells. Then the cells follow simple but powerful rules for Birth, Death, and Survival. The Colony begins to Live... Grow... Decline... and Create other Colonies. Fascinating to Watch. **(48K AS)**

ABACUS/GO/QUICKSORT/INITIA...

\$22.95 + Shipping
Featured in Vol. #3, Issue #4

Electric Abacus is a Lo-Resolution Graphics version of the timeless oriental calculator. Demonstrates the logic and operation of your very own Graphic Abacus. **(16K AS)**

Apple GO is a flexible Utility which exercises All of your DOS Keyboard Commands with only a Keystroke or two for each command. Use a Single Letter for the Command and a Single Letter-designator for each File Name. Also displays Disk Free Space, File Sizes and Binary File Addresses and Lengths. **(48K AS)**

QuickSort is a fast Machine Language Utility for Sorting your Arrays 50-170 Times faster than with Applesoft. A Simple Ampersand-Command specifies your One or Two Dimensional Array and puts the array in order in a Flash. **(48K ML)**

Apple INITIA is a unique Utility for Setting and Resetting Individual Arrays within your Applesoft Programs. With normal Applesoft, you have to use FOR...NEXT Loops to initialize and clear arrays. It takes 47 seconds to initialize an A\$(1000,10) Array. INITIA does the job in 2.5 seconds! **(48K ML)**

A.L.E. (APPLESOFT LINE EDITOR)/SPRINT/OTHELLO...

Featured in Vol. #3, Issue #5
New Price \$29.95 + Shipping)

A.L.E. (Applesoft Line Editor) is a powerful utility for helping you write and edit Applesoft programs. Auto Line Numbering, Insert, Delete, and other commands allow you to directly edit program lines without having to retype or recopy entire lines. Watch your Applesoft lines Open Up Magically to Make Room for Additions! A Single Control Character saves the Entire Line when you've finished Editing it. A.L.E. is a MUST for your Library. **(48K ML)**

SPRINT (SPEED READING IMPROVEMENT TECHNIQUE) is an automatic Text/Number Generator which can help you increase your Reading Speed by up to 100%. It has options for Pre-schoolers up through the Adult level of Reading. **(48K AS)**

OTHELLO is a Lo-Res Graphics version of the popular board game. You Flip and Flank your Opponent and capture More and More Pieces to Win. You can Play against another Player or against your Apple! And your Apple is an excellent Player! **(48K AS)**

DDT/HOURLASS/DISK DUMP/INPUT-SCREEN/SUPER-WRITER...

Featured in Vol. #3, Issue #5...
New Price \$22.95 + Shipping)

nibble[®] software

DDT is a set of Fast Utilities for Relocating and Imbedding programs above DOS Buffers and Recapturing Tracks 1 and 2 from your Disk. (48K AS)

ELECTRIC HOURGLASS is a Lo-Res Graphics Hour Glass with Sound and Multi-Colored Sand (from Different Deserts). Demonstrates Graphics and Game Methods. (16K AS)

DISK DUMPER provides a fast simple way to Dump Selected or All Disk Files to Your Printer Automatically! (48K AS)

INPUT SCREEN BUILDING providing samples for Professional Data Entry Screens using simple DATA Statements. (16K AS)

SUPER WRITER lets you create formatted text files using your Applewriter Program. Formats Text in Pages for Forward and Backward Page Review. (48K AS)

A.R.C. (APPLESOFT RECORD COMMAND)/DOS COMMANDER/ART GALLERY ...
Featured in Vol. #3, Issue #6 ...
\$29.95 + Shipping

A.R.C. (Applesoft Record Command System) is a comprehensive Data Filing, Editing, Printing, and Communications program. Build your own Records and Fields... Search Records... Merge Files... Send data over communications Lines. Fast, In-Memory Operation! (48K AS)

DISK Commander is a very handy utility that Shortens your DOS Commands to 3 Characters and your Catalog Names to 1-2 Digits. Saves Wear and Tear in Catalog Control. A few Keystrokes Do the Job! (48K ML)

Apple Art Gallery is a comprehensive collection of 33 Different Art Forms! Each Art Form has Multi-Variations for Hours of Entertainment and it's all in Hi-Res Graphics! (48K AS)

BINARY CLOCK/DUMP/SHOP LIST/PRINT USR/COMPARE ...
Featured in Vol. #3, Issue #6 ...
\$22.95 + Shipping

Binary Clock is an accurate Machine Language Timepiece in Lo-Res Graphics. See the current Time in 8-4-2-1 Code! (16K ML)

Hi-Res Dump combines the Epson MX-80 and Graftrax to produce Giant 2X Prints of the Hi-Res Graphics Screen! (16K ML)

Shopping List is a comprehensive Personal Shopping and Inventory Program which demonstrates the principles of Structured Programming! Performs Editing, Filing and Display of your Personal Shopping items. (48K AS)

Print USRing is a fast accurate new Utility for formatting Big and Little Numbers in your programs. Automatic Decimal Alignment, Scientific Notation, and more. (48K ML)

Compare Applesoft is a valuable Utility for Comparing different versions of your Programs for Changes, Additions, and Deletions. A Must for Keeping Track of new versions! (48 AS)

AMP-L-SOFT/MESSAGE CENTER/FOOTBALL ...
Featured in Vol. #3, Issue #7 ...
\$29.95 + Shipping

AMP-L-SOFT is a whole Library of Utilities under Ampersand Control. TONE, SYSCALL, HEXCONVERT, BIT CONTROL, TAGSORT, and SUBSTRING SEARCH are directly accessible by your programs to improve the speed and power of Applesoft! (48K ML)

Electronic Message Center is a versatile Message Handler for Entering, Editing, and Automatically Displaying Sequences of Messages. Great for Passing the Word at Home or the Office. (48K AS)

Apple Bowl Football is a Fast Moving Text Game that uses Actual Football Statistics to produce a Realistic Simulation of the game! A complete selection of Plays! Pass, Run, and Kick your way to Glory! (48K AS)

SHAPE/LIGHTSABRE/PRICE/SCROLL/LOWER CASE WRITER ...
Featured in Vol. #3, Issue #7 ...
\$23.95 + Shipping

The Shape is a Powerful Free-Form Shape Table Creation program that lets you create Graphics Shapes on the Lo-Res Screen. Then The Shape scans your shape and translates it into a Hi-Res Table for your own programs. (48K AS)

LightSaber Battle pits Darth and Luke to a Slug-Fest in Low Resolution Graphics. Learn the Basics for your own Game Development. (16K AS)

Price Demo uses a Product Pricing problem to illustrate the use of Indexing to access very large files! (48K AS)

Apple Scroller puts Right, Left, and Downward Text Scrolling at your fingertips with the fast Machine Language program! A spectacular addition to your Library. (48 ML)

Lower Case Writer lets you type Upper and Lower Case Letters to your Hi-Res Screen. Experiment with Text Processing or add titles to your Games! (48K ML)

Includes Key Perfect/Checker Tables for Major Programs!

Key to System Requirements: 16K Memory Size Required. (48K etc.)
AS Applesoft IB Integer Basic ML Machine Language
*Software prices above plus \$1.50 Shipping/Handling. (\$2.50 Outside the U.S.)

For **Software** ordering information, see the coupon on the Inside Back Page of this Issue. **Other Products** may be ordered by addressing your order directly to:

NIBBLE
Box 325
Lincoln, MA 01773
(617) 259-9710

Master Charge and Visa Cards Accepted

NIBBLE INDEXES — 1980-1981
\$5.95

A New, 30-page, 4-Way Index to all of the Articles, Letters, and Programs in the first two volumes of Nibble Magazine. The Index is sorted and printed:

By Title
By Author
By Subject
By Issue

Types include: Letters, Articles, Errata, Enhancements, etc.

Subjects include: Personal Finance, Graphics, Errata, Enhancements, Games, Utilities, Reviews, Tips, Pascal, and More.

Cross-Indexed to the Nibble Express.

Never again wonder about where to go to find what you want! The Index will guide you there!

APPLE TURTLE GRAPHICS TUTOR ... \$23.95 + Shipping*

A complete Turtle package for Young and Old! Dynamic Demonstration Library of all Turtle functions shows you what the Turtle can do and how to put it through its paces! Create your own Pictures! Combine them! Even create 3-D Effects! The Turtle Tutor is simplicity itself to operate! (48K AS)

RECIPE BOX/DISK DOCTOR/QUASAR ...
\$29.95 + Shipping

Apple Recipe Box makes Cook-Booking Simple! Retrieve, Edit, Display and/or Print your favorite Recipes! Automatic portioning for 1/2X, 2X, and 4X Servings! You'll love it as a kitchen aid! (48K AS)

Apple Disk Doctor is a comprehensive "Zap" Utility for Direct Reading, Writing, and Editing of your diskettes. See Deleted Files... Hide your Catalog Names... See how DOS stores your Data! (48K ML)

Quasar II puts you in command of a Starship charged with clearing distant galaxies of Meteors and Asteroids. Dodge the debris and Fire before you collide! Fast, flicker-free animation! (48K A/S and M/L)

FAT GRAPH/DRAW/SAV-DEL/FINANCIER ...
\$22.95 + Shipping

Fat Graph gives you Hi-Res Graphics tracking of your Dieting Objectives and Actual Weight Loss. Watch the pounds roll off in color! (16K AS)

Southern Draw gives you a new, precision "Pencil" cursor for drawing Hi-Res Graphics Pictures. Fast, Easy Sketching! (16K AS)

SAV-DEL is a sophisticated and easy to use utility for Saving/Deleting Selected Arrays in Memory. You can pick what you want Without having to CLEAR the whole Memory. (48K M/L)

Financier - Tax Depreciation adds new Tax Rules and Financier updates for the Economic Recovery Act. Use it with Apple Financier or as a Stand-alone Program. (48K A/S)

nibble Programs On Diskette!

nibble

Box 325, Lincoln, MA. 01773



☐ APPLE TURTLE GRAPHICS TUTOR \$23.95 + Shipping*

☐ RECIPE BOX/DISK DOCTOR/QUASAR \$29.95 + Shipping*

☐ FAT GRAPH/DRAW/SAV-DEL/ FINANCIER \$23.95 + Shipping*

☐ AMP-L-SOFT/E.M.C./FOOTBALL \$29.95 + Shipping*

☐ SHAPE/LIGHT SABRE/PRICE/SCROLL/LOWER CASE
\$23.95 + Shipping*

☐ A.R.C./DISK COMMANDER/GALLERY \$29.95 + Shipping*

☐ BINARY CLOCK/DUMP/SHOP LIST/PRINT USR/COMPARE
\$22.95 + Shipping*

☐ A.L.E./SPRING/OTHELLO \$29.95*

☐ DDT/HOURLASS/DUMP/INPUT SCREEN/WRITER \$22.95*

☐ APPLE MICROCALC/VARIABLE CRUNCH/LIFE
\$29.95 + Shipping*

☐ ABACUS/GO/QUICKSORT/INITIA \$22.95 + Shipping*

☐ APPLE BOND MANAGER/DOS 3+2/DARTS
\$29.95 + Shipping*

☐ AMPERSPEED/LAMP II/PROTECTOR/3-D MAP
\$22.95 + Shipping*

☐ APPLE MLE/LAMP-1/APPLE CAL \$29.95*

☐ APPLE SLUGGER/PIECHART/AMPER-FIND/SORT \$19.95*

☐ TURTLE GRAPHICS/TRAC "I"/CRUNCHER \$29.95*

☐ SCRAMBLE/CHECKER/SCROLL/X-REF/FREE SPACE \$19.95*

☐ TRAC "B"/TRAC GRAPH/APPLE ARCHIVES/THREE-D
TIC TAC TOE \$29.95*

☐ HEXPAD /DOS REMOVER/COMMAND CHANGER/FORMAT/XY
PLOT \$19.95*

☐ APPLE FINANCIER III/POOR BOY LE/CATSUP/
NFL RANKER \$19.95*

☐ APPLE D.A.R.T./PRINT-USE/LASER BLASTER \$29.95*

☐ APPLE MAPS/AMPER JUMP & SORT/ARTIST \$29.95*

☐ DISK MASTER/TRAP STEP/BIORHYTHM \$29.95*

☐ APPLE STAR SYSTEM, TRAP 'EM, AND BOOT 3.2/3.3 \$29.95*

☐ APPLE PAINT-BOX, MAD MAD CUBE, LINE-FINDER \$24.95*

☐ MUSIC RETRIEVAL SYSTEM (MRS), LE MANS, and MORSE CODE
SYSTEM \$19.95*

☐ APPLE CH.A.M.P., AMPER-READER, and APPLE
COMMUNICATOR \$28.95*

☐ WILL 'O THE WISP, BLAST AWAY! And LOAN REDUCTION
\$19.95*

☐ APPLE AIM, CONCORDANCE, LOW SCORE \$28.95*

☐ PERSONAL INVENTORY PROGRAM (PIP II), AMPER-UTILITY, and
FOUR-IN-A-ROW \$29.95* (Amper Update: \$8.95 below)

☐ FORECASTING AND SALES TRENDS (F.A.S.T.), SUPERWEAVER,
and FOOTBALL \$24.95*

☐ APPLE TEXT PROCESSOR (T.O.U.G.H.), APPLE SIMON, Includes
T.O.U.G.H. Enhancements for Tabbing, Fast Disk Read-Write, and
Expanded Format control. (Works with Epson MX-80 and
Centronics 737). \$24.95*

☐ STAR ATTACK, AIRSEA BATTLE, and LOW-RES SHAPEWRITER
Includes Enhancements \$19.95*

☐ APPLE TRAC (Home Finance and Expense Analysis System) and
SPACE MAZE. Includes TRAC PLUS!
\$24.95*

NOTE: Nibble disks are now supplied in 16 sector (DOS 3.3) format
All Nibble Software runs on the Apple IIe, Franklin Ace,
and other Apple compatible computers.

Enclosed is my

☐ check ☐ money order
for \$ _____

(All prices plus shipping)

*U.S. Shipments add \$1.50 for shipping and handling. Outside U.S. add \$2.50
for shipping and handling. Mass. residents add 5% Sales Tax.

Note: Disk updates require return of original NIBBLE diskette. Add shipping
and handling.



We accept MasterCard and VISA



Card # _____ Exp. _____
PLEASE PRINT CLEARLY

Telephone _____

Name _____

Address _____

City _____

State _____ Zip _____

NIBBLE EXPRESS

TABLE OF CONTENTS

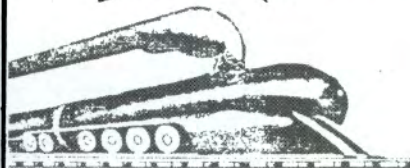
EDITORIAL	3
APPLE TRAC — Personal Finance Management by Mike Harvey	7
SORT 'EM OUT — Principles of Sorting by NIBBLE Staff	17
SEUDO UTO-START — Reset with CTRL Y by Rick Con	17
INITIALIZE NEW FILES WITH ONERR GOTO by NIBBLE	18
MACHINE LANGUAGE SCREEN DUMP by R.M. Mottola	18
FREE? DISK SECTORS by Chuck Hartley	19
HI-RES SPACE MAZE — Graphics Game by NIBBLE Staff	22
UN-GRAPHIC GRAPHIC PRINTING by NIBBLE Staff	23
TABLE PRINTING MADE SIMPLE! by NIBBLE Staff	24
DYNAMIC ARRAY DIMENSIONING by NIBBLE Staff	26
BLOCKING VERY LARGE FILES by NIBBLE Staff	26
LOW RESOLUTION SHAPEWRITER — High Speed Actio	31
SPACE ANIMATION — Add ZIP to your Games by NIBBL	35
STAR ATTACK — Fast Hi-Res Conflict Game by Mike Har	37
PADDLE READING IN ASSEMBLY LANGUAGE by NIBBL	38
FIRING HI-RES MISSILES — Aiming and Control by NIB	39
AIRSEA — Join the Navy/Air Force Maneuvers b	47
AUTO-INTERTEXT PAGE CONTROL by NIBBLE Staff	50
WATCH OUT FOR GRAPHICS OVERFLOW by Mike Harv	50
T.O.U.— The Tower of Babel by Mike Harvey	53
TOUCHY LOCKE HANGEMENTS by William Reynolds III	58
TOUCH FIND AND REPLACE OPTION by William Reyno	59
DOUBLE/TRIPLE/AND MORE OVERPRINTING by NIBBL	59
ARROWS AND CONTROL CODES by NIBBLE Staff	60
APPLE TRICKS — Fast DOS/Spcl Chars/Unlistables by C	65
APPLESOFT VS. INTEGER BASIC PERFORMANCE by A	69
APPLE II — Paper Tiger Graphics by Mike Harvey	73
APPLE II SOUND — Fun Music and Fun by Mike Harvey	77
APPLE II FILE CONTROL by Alexander Laird	78
APPLESOFT REM REMOVER by Alan D. Floeter	83
FAST FORECASTING AND SALES TRENDING by Mike H	89
SUPERWEAVER — Hi-Res Weaving Design by Alexander	93
FOOTBALL — Lo-Res Grid-Iron Action by Lou Haehn	97
BUILD DUAL JOYSTICKS FOR UNDER \$15.00 by NIBBL	98
BUILD YOUR OWN PERSONAL UNIT by NIBBLE Staff	99
DISK II — WITH THE EIGHTY COUNT by NIBBLE	101
APPLE TRACKS — CAR RACING in Living FAST by Cra	103
PIPI — Personal Inventory Program on Tape by Rick Con	106
FUN WITH ASSEMBLY — The Fun by by by Craig Cr	113
PIPI II — PIP Goes Discs by Mark Capella	119
PASSING VARIABLES IN APPLESOFT BASIC by R.M. M	120
MANAGING AND MOVING DISK BUFFERS by William Re	121
MONITOR EXECUTION — Basically by William Reynolds	123
AMPER-INTERPRETER — Add Print-Using and Much Mo	133
FUN WITH ASSEMBLER — Graphics by Alexander Laird	135
STRING FUNCTION FOR INTEGER BASIC by William Re	135
BASIC ASSEMBLY SUBROUTINE CREATOR	136
CHRS FUNCTION FOR INTEGER BASIC by William Reyr	139
FUN WITH ASSEMBLER — Apple II by by by Craig Cross	147
APPLE A.I.M. — Animated Intelligent Mailing by Michael	153
APPLE CONCORDANCE — Track Variable and Line #'s b	157
LOW SCORE II — Strategy Game by Rudy A. Guy	158
HOW TO WRITE GAMES THAT LAST by Mike Harvey	159
IMPROVING THE MULTIPLE SCREEN SORT by Rick Con	161
APPLE UPPER/LOWER CASE PRINTING by Mike Harvey	169
WILL O' THE WISP — High Adventure by Mark Capella	171
NIFFUM — DOS 3.3 to 3.2 Conversion by C.J. Thompson	174
BLAST AWAY! — Lo-Res Shooting Gallery by Andrew Be	
FUN WITH MONITOR — How to Enter Assembly Language	

SECOND PRINTING!

BRAND NEW PROGRAM LISTINGS!

Includes Key Perfect and Apple Checker Tables for all Major Programs!

nibble
EXPRESS!
(for your Apple*)



ORDER NOW

All programs and Articles are centered on the Apple Computer family.

NIBBLE

P.O. Box 325
Lincoln, MA 01773

Yes! I want NIBBLE EXPRESS Vol. 1 in my library!
Here's my ☐ Check ☐ Money order
for \$12.95 plus \$1.75 postage/handling. (Outside U.S.
add \$2.75 Surface Mail or \$5.00 Airmail.)

☐ Also send me NIBBLE EXPRESS Vol. 2 at \$14.95 plus
\$1.75 postage/handling. (Outside U.S. add \$2.75 Surface
Mail or \$5.00 Airmail.)



Master Card & Visa Accepted

Card # _____ Expires _____

PLEASE PRINT CLEARLY

Signature _____

Telephone _____

Name _____

Street _____

City _____ State _____ Zip _____

Your check or money order must accompany your order to qualify.

Outside U.S.: Checks must be drawn on a U.S. Bank.
*Apple is a registered trademark of Apple Computer Company.

©1983 by MICRO-SPARC, INC.
P.O. BOX 325
LINCOLN, MA 01773

All rights reserved.

*Apple is a registered trademark
of Apple Computer Company.

ISBN 0-912341-00-9