COMPUTE!'s

TI Collection

VOLUME TWO

More than two dozen never-before-published programs and articles. An outstanding collection of applications, utilities, games, and tutorials for your TI-99/4A.

A **COMPUTE! Books** Publication $14.95

COMPUTE!'s
TI
COLLECTION

VOLUME TWO

COMPUTE!™ Publications, Inc. abc
One of the ABC Publishing Companies
Greensboro, North Carolina

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-87455-036-X

COMPUTE! Publications, Inc., Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is one of the ABC Publishing Companies and is not associated with any manufacturer of personal computers. TI and TI-99/4A are trademarks of Texas Instruments, Inc.

Contents

Appendix

Foreword

What do you do when you have a computer that's no longer made? If you have a TI-99/4A, you've probably asked yourself that very question.

COMPUTE!'s TI Collection, Volume Two is the perfect answer. You may not be able to buy commercial software for your computer, but with this book you can add many impressive programs to your software library.

All the programs and articles you find here are appearing for the first time anywhere. Games that entertain, or entertain *and* educate, sophisticated applications, and illuminating tutorials on a wide range of TI topics—all are ready to use on your TI-99/4A.

How about a simple memo processor, one that doesn't need a disk drive or take up an entire tape for storage? It's easy to use and perfect for short letters and memos. Ever wanted a spreadsheet for your TI? "MitiCalc" is the answer. It's not as fast as a commercial package, but it's close. You'll find it just right for those small personal accounting and financial tasks.

Tutorials and information about your computer put you inside the machine. You'll learn about everything from the TI's "heart," its microprocessor, to the way it communicates with the outside world through its joystick port. Articles on memory management, and how the TI's memory is arranged, give you clear and concise information on this important aspect of the machine.

Have you thought about programming in assembly language, but haven't had the courage to try? *COMPUTE!'s TI Collection, Volume Two* gives you two plans of attack. Type in and use such assembly language programs as "TI FastSearch" or "TI File Management." Or learn some of the basics of assembly language programming—with either the *Editor/Assembler* or the Mini Memory command module.

And we haven't forgotten how much fun the TI-99/4A can be. Games, from shoot-em-ups like "Spitfire" to the mind-boggling "BOG'L," will give you hours of entertainment. There are even children's educational games included, like "Happy Face Arithmetic" and "Spelling Tutor," which says the words for your child to spell.

With such a range of programs and information, you'll find *COMPUTE!'s TI Collection, Volume Two* an extraordinary value.

All the programs in *COMPUTE!'s TI Collection, Volume Two* are ready to type in and run. If you prefer not to type in the programs, however, you can order a 5-1/4-inch disk which includes all the programs in the book. Call toll-free 1-800-346-6767 (in New York, call 212-265-8360) or use the coupon found in the back of this book.

1

Applications and Techniques

MitiCalc

Milo Tsukroff

A spreadsheet on the TI-99/4A? Impossible, some might say. Fortunately, not *impossible. "MitiCalc" for the TI-99/4A, though not as sophisticated as* commercially *available spreadsheets which can have thousands of "cells," or entry points for numbers and other information, offers enough for most home users. For TI BASIC.*

"MitiCalc" allows you to enter titles, numbers, or equations on a table with 18 columns, each containing 18 rows. This table is often called a *spreadsheet.* Any entry which involves numbers or equations can use other numeric entries as numbers.

Because of the ability of one entry to call upon the value of other entries, a change made to a single entry can affect many others. When a spreadsheet is properly set up, this enables you to see the results of a change quickly.

When you want to save a spreadsheet, the program lets you save it on either disk or cassette tape. You can also load spreadsheets from disk or cassette, making the work of setting up a spreadsheet largely automated once you've created the basic overlay. All the options available to you while using MitiCalc are right in front of you—either on the keyboard overlay you've made or on the menu screen.

Note: When using MitiCalc on an unexpanded TI-99/4A, you'll have space for about 3000 characters. This does *not* include the numbers which will be displayed on the spreadsheet for the numeric entries. It should be possible to fill the spreadsheet without seeing a *MEMORY FULL IN LINE... message. This message indicates a program crash, which wipes out all your entries. If you feel that you've used a great deal of memory, save what you've done before continuing. Of course, if you have a TI-99/4A with more than 16K of memory, you'll be able to fill an entire spreadsheet without worrying.

Running with MitiCalc

After loading MitiCalc into memory, type RUN and press ENTER. The computer takes about 15 seconds to check the program for errors. As soon as the menu screen appears, you're ready to begin.

Place the MitiCalc keyboard overlay into the holder above the number keys.

MitiCalc Keyboard Overlay

							◀ Page	Page ▶		Clear Screen	O
Cancel Entry	Edit Entry	Erase	•BREAK•	Go to Menu	Go to Home	Title Entry	Numeric Entry	Compute Sheet		•QUIT•	O

Menu Screen Functions

When the menu screen is displayed, access the following options by simply pressing the appropriate key. All other keys are locked out.

1. Lets you load previously saved spreadsheets from either the disk drive or the cassette recorder. When loading in a stored spreadsheet, the program first clears the memory to avoid a MEMORY FULL condition. A message prompting you to enter a filename appears. If you're loading a file to disk, type

   DSK1.*filename*

   where *filename* is the name of this particular spreadsheet. Press ENTER.

   If you're loading a file to the cassette recorder, type

   CS1.*filename*

   and press ENTER. (If you want, you can omit the filename when loading a file from tape. Just make sure that the tape is positioned at the correct location for that file. Pressing ENTER simply loads the next file found on the tape.)

   After the spreadsheet has been entered, all numeric entries are calculated once. Every time a numeric entry starts to be calculated, a brief *bip* sound indicates that processing has begun. A unique *hoot* sound announces that the computer has finished processing all entries.
2. Allows you to save your present spreadsheet to either disk or tape. As in the previous option, you'll be prompted for a filename. Again, type DSK1.*filename* or CS1.*filename* and press ENTER.

   All entries in a column are saved, whether or not they're used. All columns are saved, up to the last column with entries in it.

Note: About one minute of tape is required for each column saved. Unless a great deal of room is needed, it's best to set up your spreadsheet in the leftmost columns.

3. Clears out all entries on the spreadsheet, both title entries and numeric entries. This takes about 15 seconds.
4. Pressing this key ends the program. Using it is easier than pressing FCTN-4, and should be the only way that you end with MitiCalc.
5. Displays the spreadsheet. Before displaying, the computer asks you to enter the starting entry position. Enter the location as a column letter, A–O (uppercase only), and a number, 1–18. Some possible entries are A12, Q6, and J13. Press ENTER. If you simply wish to start in the Home position—A1—press ENTER only. Note: If you enter an incorrect location, the program repeats the prompt. Though the range of columns runs from A to R, you cannot use columns P, Q, and R in an entry location when going to the spreadsheet.
6. All numeric entries are calculated. Processing starts at column A and proceeds to the last column with a numeric entry. Every time a numeric entry starts to be calculated, a brief *bip* sound indicates that processing has begun. When all entries have been processed (which takes some time for larger sheets) the *hoot* sound announces that the computer's done.

Special-Function Keys

When the MitiCalc spreadsheet is displayed, you'll see four columns of entries on the screen. Each column is only seven characters wide—title entries show only the first seven characters, and numbers longer than seven characters cannot be displayed. The column letters appear at the top of the spreadsheet, row numbers at the extreme left and right. The current cell is indicated by both a pair of flashing cursors on the spreadsheet and by an entry location shown near the upper-left corner.

Running along the top of the screen is a space, called the *display line,* in which the full entry is displayed. It shows all the characters of the entry at which you're located.

The following Function and Control keys are active and should be indicated on your keyboard overlay (see the figure above). No other keys are active. Pressing an active key results in a keytone, accompanied by the appropriate action. Pressing an invalid key results in a keytone, immediately followed by a bad-key *boop* sound.

Key	**Function**
FCTN-2	Edit Entry
FCTN-4	BREAK
FCTN-5	Go to Menu
FCTN-6	Go to Home (A1)
FCTN-7	Title Entry
FCTN-8	Numeric Entry
FCTN-9	Compute Sheet
FCTN-=	QUIT
CTRL-=	Clear Screen
CTRL-8	Page Left
CTRL-9	Page Right
FCTN-E	Up
FCTN-X	Down
FCTN-S	Left
FCTN-D	Right

BREAK (FCTN-4) and QUIT (FCTN-=) are active at all times. *Do not press them by mistake.* To end the program, it's best to return to the menu screen and use the 4 option. Hitting either FCTN-4 or FCTN-= could ruin hours of hard work.

If you do accidentally press BREAK by mistake, don't panic. You actually haven't lost anything—yet. Immediately type in CONTINUE or CON and press ENTER. The blinking cursors will reappear, but in the wrong place, because the screen has scrolled. Press FCTN-5 (Go to Menu) to access the menu screen, and use the 5 option to return to the spreadsheet. The sheet will be redrawn correctly when you do this, and no data will be lost.

What the Keys Do

FCTN-E, X, S, D Up, Down, Left, Right

Each time one of these keys is pressed, the dual cursors move one row or column in the desired direction. The display line is updated with the contents of the new position, and the location indicator is reset. If you move beyond the edge of the displayed part of the spreadsheet, it's redrawn with the columns shifted over one.

CTRL-8, 9 Page Left, Page Right

These keys redraw the entire spreadsheet, shifted over four columns in the appropriate direction.

FCTN-5 **Go to Menu**
This returns you to the menu screen.

FCTN-6 **Go to Home**
Redraws the spreadsheet, with the first four columns shown and the cursor at the Home—A1—position.

CTRL-= **Clear Screen**
Clears all entries in the four displayed columns. Caution: *Do not press FCTN instead of CTRL.* FCTN-= is QUIT.

FCTN-7 **Title Entry**
Erases the display line and puts a blinking cursor at the beginning of the display line. Any characters put onto the display line after you use this key are entered as a title entry. The cursors on the entry location in the spreadsheet will freeze there while you work on the display line.

FCTN-8 **Numeric Entry**
Erases the display line and puts a blinking cursor on the second space of the display line. (The first space is reserved for a special character.) You'll be able to enter only certain characters and numbers onto the display line. The number value of the numeric entry will be displayed on the spreadsheet if it contains seven characters or fewer. The characters of the actual entry can be seen only on the display line. As with Title Entry, the cursors on the entry location freeze while you work on the display line.

FCTN-2 **Edit Entry**
Allows you to change or modify an existing entry, whether title or numeric. The cursor is put on the display line, and the blinking cursors freeze.

FCTN-9 **Compute Sheet**
All numeric entries are calculated, from the top of each column to the bottom, proceeding from Column A to the last column which includes a numeric entry. This may take some time, so the program indicates when it's started evaluating each numeric entry with a brief *bip* sound. When all entries have been evaluated, the program announces the fact with a multitonal *hoot.*

The Display Line—Special Keys

The display line can hold a total of 30 characters. Only when the cursor is on the display line can you enter information. It's necessary to type fewer than 25 words per minute to allow the program to pick up every character. Because a tone sounds every time a key is struck, you can tell if you're typing too fast. If you do, the computer will begin to skip tones. If you hit a key which is not allowed, such as inactive Function keys, you'll also hear a bad-key *boop,* and no character will be entered.

The following Function keys are active when in the display line.

Key	**Function**
FCTN-1	Cancel Entry
FCTN-3	Erase
FCTN-4	BREAK
FCTN-=	QUIT
FCTN-S	Left
FCTN-D	Right
ENTER	Enter

Remember that BREAK (FCTN-4) and QUIT (FCTN-=) are always active. *Do not press them by mistake.* Follow the instructions previously given if you accidentally press BREAK.

Display Line Key Features

FCTN-S, D Left, Right
Use these keys to move the cursor left or right along the display line.

FCTN-3 Erase
Erases the entire display line. It also changes a numeric entry to a title entry by eliminating the numeric entry's special first character.

FCTN-1 Cancel Entry
Pressing this key eliminates all changes made to the display line. The old line is displayed, and you're returned to the blinking cursors of the spreadsheet.

ENTER Enter
Makes the program read the string of characters on the display line into memory. If the entry is a title entry, it goes right in. If the entry is a numeric entry, it is processed and checked for errors before it's entered into the spreadsheet's memory.

The Display Line—The Title Entry

When you're typing a title entry, any characters can be entered on the display line—numbers, letters, or symbols. Only the first seven characters of your entry are displayed. All other characters will be shown only on the display line. If there are fewer than seven characters, they're left-justified—flush with the left edge of the entry block. Spaces are counted as characters only if they're between characters or to the left of characters which are visible on the display line.

For example, a display line could show

IT IS FINISHED.

The initial space and the spaces between IT and IS and between IS and FINISHED all count as characters. The entry is 16 characters long in memory, which counts toward the 3000-character limit. What's shown on the spreadsheet, however, is

IT IS

To enter a new title entry into the spreadsheet after typing it onto the display line, press ENTER. Any previous entry will be erased, and the numeric value of that entry is set to zero.

The Display Line—The Numeric Entry

When typing in a numeric entry, you can enter only certain characters. All other characters are bad keys and will not appear in the display line when you type them. None of the characters on the display line will be displayed directly. Instead, the entry shows the numeric value of the entry.

The first character of a numeric entry has an ASCII value of 127, and appears as a blank. Because eliminating it turns a numeric entry into a title entry, it cannot be deleted except with the FCTN-3 (Erase) command.

To enter a numeric entry, press ENTER. First, the brief tone sounds to indicate that the entry has entered the subroutine where it's to be processed. Then, if there's an error which can be detected, a sharp beeping tone sounds. The cursor will reappear on the display line at the location of the mistake. When a numeric entry doesn't have a detectable error, it's evaluated and its value is displayed on the spreadsheet. The value is also entered internally for use by other entries at this point. All values are rounded to the second decimal place, whether shown on the spreadsheet or stored internally.

If the character length of the number is longer than seven characters, including decimal points and negative signs, the number *will not* be displayed. Instead, a series of exclamation points will appear. All numbers are displayed right-justified.

Mathematics of the Numeric Entry

The numeric entry is the most important part of MitiCalc. It's evaluated by a large subroutine which reads the characters of the entry and comes up with the entry's value.

Numbers. Numbers can be entered normally, including decimals, but negative numbers cannot be entered directly—the minus sign is recognized only as indicating subtraction. Also, scientific notation cannot be entered directly. On the spreadsheet, negative numbers and, occasionally, numbers in scientific notation will be displayed. One convenient way to enter a negative amount is to subtract a number from zero. Examples of acceptable numbers are 32899, 7.2333, 4, 6783, and 0−50.

Operators. You can add, subtract, multiply, and divide numbers or items which are processed as numbers. Here are some examples of acceptable entries:

14+6.7	Addition
14−6.7	Subtraction
14*6.7	Multiplication
14/6.7	Division
23.*2+48.6	Equations can be entered, up to the 29-character limit.

Some examples of unacceptable entries include:

23.53.+46	Extra period in first number
−16+2	First number is missing
45*+47.88	Two adjacent operators; middle number is missing
56/	No number at end
45E16+1000000	Scientific notation not allowed

Entry References. Instead of a number, you can refer to the value of another numeric entry. You must enclose the location of the desired entry in parentheses. Because of this requirement, *parentheses may not be used for any other reason in a numeric entry.* The column letter must be an uppercase letter from *A* to *R*, and the row number, which follows immediately without a space, has to be in the range of 1 to 18. Thus, three characters is the maximum within the parentheses. A few acceptable entry references follow:

```
(A18)+3
43*(Q6)-17.34
0-(A09)*2+403.79
```

Unacceptable entry references include:

(AA2)	Not an entry location
14+C5/2	No parentheses
(E7-3	Missing parentheses
(M19)	Row location is beyond 18
(D 2)	Space in entry reference
(G014)	Entry reference too long

Column Sums. A convenient feature, the column sum, counts as a number in an equation. It's indicated by a dollar sign ($), followed immediately by the column letter. The column sum is taken of an entire column—except when the column is the same one that the numeric entry is in. When that's the case, only the numeric entries above the row in which the numeric entry is located are added. Examples of acceptable column sums in numeric entries are

```
$B
17+$F-493.45
(C14)/7.5+$E-14000
```

Order of Processing. Equations are processed from left to right. Algebraic hierarchy is *not* followed, and parentheses are only to be used for entry references. This can cause problems, and it often requires ingenuity to write an expression that will work as desired. If necessary, an unused entry location may be used to hold intermediate calculations. (If correct algebra was allowed, the program would be *much* larger and, even worse, *much* slower.) The process of coming up with a solution to an equation is much like an inexpensive pocket calculator, and looks like this:

```
3*14+4/2   Equation
  42+4/2
     46/2
       23   Answer
```

If the situation was such that $B=3, (C3)=14, and (D17)=2, the equation would look like this:

```
$B*(C3)+4.00/(D17)   Equation
     42+4.00/(D17)
           46/(D17)
                 23   Answer
```

Mistakes. Some mistakes won't trigger the error warning and may in fact produce strange results. These mistakes are missing operators before or after a column sum or an entry reference; division by zero or by a number that's extremely close to zero; and multiplication of two extremely high value numbers.

If no characters or numbers are entered, an error will result. To eliminate the entry entirely, including the special first character, use the FCTN-3 (Erase) key.

Processing of Entire Sheet

Because of the direction of processing, when the entire sheet is computed, it's best to make entry references or column sums refer to entries that are either

- in columns to the left of the numeric entry's location, or
- above and in the same column.

Referencing below or to the right, even if the entries are already in place, should be avoided. If this is not avoided, it will be necessary to compute the entire sheet one or more *extra* times when the spreadsheet is loaded from tape or disk. References to entries or columns which are not computed at the time of referencing return a value of zero, as do references to title entries.

Editing the Display Line—Helpful Hints

Making modifications to existing entries is easier with the following tips.

- If you decide that you don't want to make any modifications after all, you can use FCTN-1 (Cancel) to eliminate any changes that you've made. This is especially useful if you don't remember exactly what was in the old display line, or if a numeric entry has errors you can't easily fix. The old display line will be redisplayed, and you'll be returned to the blinking cursors of the spreadsheet.
- To erase an entry quickly and easily, press FCTN-7 (Title Entry) from the spreadsheet. Press ENTER, and the entry will be completely cleared, including the internal numeric value.
- You cannot insert or delete characters. This worked too slowly in TI BASIC. To eliminate characters, use the space bar. Sometimes it may be faster to use FCTN-3 (Erase), but be careful if you're working on a numeric entry. If you erase the special first character, it's changed to a title entry.

- Because the only characters needed for a numeric entry are numbers, operators, and the capital letters *A* to *R*, only these are allowed. It's helpful to use ALPHA LOCK when typing a numeric entry to avoid making mistakes on entry references or column sums.
- If you want to process a single numeric entry already in the spreadsheet without computing the entire sheet, use FCTN-2 (Edit Entry). Press ENTER without making any modifications, and the entry will be reevaluated.
- Remember while typing on the display line (and elsewhere in the program) that a tone sounds every time you strike a key, *and it registers in the program.* It is possible to press a key and have the program *not* pick it up. If you don't hear a tone, the computer didn't pick up your keystroke.

Check Balancing

Here's a sample spreadsheet I've developed. It demonstrates how MitiCalc works. To type it in, go to the spreadsheet display and move to the locations indicated. In each location, put in either a title or numeric entry—whichever is indicated.

Location	**Title Entry**	**Numeric Entry**
A1	OrigBal	
A2	Chk#+Dt	
B1		(Original balance of the account)
B2	Amount	
A18	Balance	
B18		2*(B1)−$B
C1	Balance	
C2	Chk#+Dt	
D1		(B18)
D2	Amount	
C18	Balance	
D18		2*(D1)−$D
A3–A17, C3–C17		(Number of check, date, and other information)
B3–B17, D3–D17		(Amount of check whose number is to the left of the entry)
A3–A17, C3–C17		(Deposits—date and other information)
B3–B17, D3–D17		(Deposits: Subtract the amount of the deposit from zero to get a negative amount)

After putting in all checks, press FCTN-9 (Compute Sheet) to get the new balance.

MitiCalc

```
100 CALL CHAR(150,"000000FFFF")
110 CALL CHAR(151,"4040404040404040")
120 CALL CHAR(152,"0202020202020202")
130 CALL CHAR(153,"FFFFFFFFFFFFFFFF")
140 DEF B(A$)=SGN(ASC(A$)-127)+1
150 OPTION BASE 1
160 DIM U(16),O$(16),N(18,18),S$(18,18)
170 CALL CLEAR
180 PRINT "      ** MitiCalc **"
190 PRINT ::TAB(8);"** MENU **"::"1 Load old data f
    rom":"  cassette or disk."
200 PRINT "2 Save the program's data":"  on cassett
    e or disk.":"3 Clear all entries.":"4 End this
    program."
210 PRINT "5 Go to the MitiCalc table.":"6 Recalcul
    ate entire sheet.":::" PRESS THE APPROPRIATE KE
    Y."
220 CALL KEY(5,K,S)
230 IF S=0 THEN 220
240 CALL SOUND(100,320,12)
250 IF (K<49)+(K>54)THEN 220
260 ON (K-48)GOSUB 3230,3370,3140,3670,3470,3330
270 GOTO 170
280 P=1
290 Y=1
300 X=1
310 FOR E=1 TO 24
320 CALL HCHAR(E,1,153,32)
330 NEXT E
340 CALL HCHAR(5,4,150,28)
350 CALL VCHAR(6,3,150,18)
360 FOR E=1 TO 9
370 CALL HCHAR(E+5,2,E+48)
380 CALL HCHAR(E+5,32,E+48)
390 NEXT E
400 FOR E=0 TO 8
410 CALL HCHAR(E+15,2,E+48)
420 CALL HCHAR(E+15,32,E+48)
430 CALL HCHAR(E+15,1,49)
440 NEXT E
450 CALL HCHAR(4,4,32,28)
460 FOR E=1 TO 4
470 CALL HCHAR(4,E*7-3,151)
480 CALL HCHAR(4,E*7+3,152)
490 CALL HCHAR(4,E*7,P+E+63)
500 NEXT E
```

```
510 FOR G=P TO P+3
520 FOR Y1=1 TO 18
530 IF S$(G,Y1)="" THEN 960
540 IF B(S$(G,Y1))THEN 980
550 D$=S$(G,Y1)
560 TL=1
570 X1=G-P+1
580 GOSUB 2870
590 NEXT Y1
600 NEXT G
610 CALL HCHAR(2,2,32,30)
620 Z=P+X-1
630 FOR E=1 TO LEN(S$(Z,Y))
640 CALL HCHAR (2,E+1,ASC(SEG$(S$(Z,Y),E,1)))
650 NEXT E
660 CALL HCHAR(4,1,30,3)
670 CALL HCHAR(4,1,Z+64)
680 FOR G=1 TO LEN(STR$(Y))
690 CALL HCHAR(4,G+1,ASC(SEG$(STR$(Y),G,1)))
700 NEXT G
710 J=Y+5
720 CALL GCHAR(J,X*7+3,W)
730 CALL GCHAR(J,X*7-3,M)
740 CALL HCHAR(J,X*7-3,30)
750 CALL HCHAR(J,X*7+3,30)
760 CALL KEY(5,K,S)
770 IF S THEN 810
780 CALL HCHAR(J,X*7-3,M)
790 CALL HCHAR(J,X*7+3,W)
800 GOTO 720
810 CALL SOUND(-100,200,15,400,17)
820 IF K=6 THEN 1750
830 IF K=1 THEN 1300
840 IF (K=4)*(S$(Z,Y)<>"")THEN 3130
850 CALL HCHAR(J,X*7-3,M)
860 CALL HCHAR(J,X*7+3,W)
870 IF INT(K/4)=2 THEN 1000
880 IF (K=158)*(P<>1)THEN 3030
890 IF K=157 THEN 1190
900 IF K=15 THEN 2690
910 IF (K=159)*(P<15)THEN 3080
920 IF K=14 THEN 3220
930 IF (K=12)*(P<>1)THEN 280
940 CALL SOUND(-450,110,10)
950 GOTO 720
960 CALL HCHAR((Y1+5),((G-P+1)*7-3),32,7)
970 GOTO 590
980 D$=STR$(N(G,Y1))
990 GOTO 570
1000 ON (K-7)GOTO 1010,1070,1130,1160
1010 IF X=1 THEN 1040
```

```
1020 X=X-1
1030 GOTO 610
1040 IF P=1 THEN 610
1050 P=P-1
1060 GOTO 450
1070 IF X=4 THEN 1100
1080 X=X+1
1090 GOTO 610
1100 IF P=15 THEN 610
1110 P=P+1
1120 GOTO 450
1130 IF Y=18 THEN 610
1140 Y=Y+1
1150 GOTO 610
1160 IF Y=1 THEN 610
1170 Y=Y-1
1180 GOTO 610
1190 FOR D=P TO P+3
1200 FOR E=1 TO 18
1210 S$(D,E)=""
1220 N(D,E)=0
1230 NEXT E
1240 NEXT D
1250 IF (LC>P+3)+(LC<P)THEN 1270
1260 LC=P-1
1270 IF (LL>P+3)+(LL<P)THEN 1290
1280 LL=P-1
1290 GOTO 290
1300 CALL HCHAR(2,2,32,30)
1310 FG=0
1320 P2=2
1330 CALL GCHAR(2,P2,P1)
1340 CALL HCHAR(2,P2,30)
1350 CALL HCHAR(2,P2,P1)
1360 CALL KEY(5,K,S)
1370 IF S=0 THEN 1330
1380 CALL SOUND(-100,550,15)
1390 IF K=7 THEN 1300
1400 IF K=3 THEN 3570
1410 IF K=13 THEN 1540
1420 IF (K=8)*(P2<>2)*(FG=0)+(K=8)*(P2<>3)*(FG=1)TH
     EN 1520
1430 IF (K=9)*(P2<>31)THEN 1480
1440 IF (K>126)+(K<32)THEN 1500
1450 IF (FG)*((K>82)+(K=44)+(K>57)*(K<65)+(K<40)*(K
     <>36)*(K<>32))THEN 1500
1460 CALL HCHAR(2,P2,K)
1470 IF P2=31 THEN 1500
1480 P2=P2+1
1490 GOTO 1330
```

```
1500 CALL SOUND(-450,120,5)
1510 GOTO 1330
1520 P2=P2-1
1530 GOTO 1330
1540 IF LL>Z THEN 1560
1550 LL=Z
1560 FOR D=31 TO 2 STEP -1
1570 CALL GCHAR(2,D,G)
1580 IF G<>32 THEN 1600
1590 NEXT D
1600 Q$=""
1610 FOR E=1 TO D-1
1620 CALL GCHAR(2,E+1,G)
1630 Q$=Q$&CHR$(G)
1640 NEXT E
1650 IF Q$="" THEN 1670
1660 IF B(Q$)THEN 2580
1670 S$(Z,Y)=Q$
1680 D$=Q$
1690 N(Z,Y)=0
1700 TL=1
1710 X1=X
1720 Y1=Y
1730 GOSUB 2870
1740 GOTO 720
1750 CALL HCHAR(2,2,127)
1760 CALL HCHAR(2,3,32,29)
1770 P2=3
1780 FG=1
1790 I=0
1800 GOTO 1330
1810 CALL SOUND(20,1000,12)
1820 FOR F=1 TO 16
1830 U(F)=0
1840 O$(F)=""
1850 NEXT F
1860 I$=""
1870 T=1
1880 FOR F=2 TO (LEN(Q$))
1890 V$=SEG$(Q$,F,1)
1900 IF (V$=".")*(POS(I$,".",1)<>0)THEN 2070
1910 IF V$="." THEN 1940
1920 IF V$<"0" THEN 2240
1930 IF V$>"9" THEN 2070
1940 I$=I$&V$
1950 NEXT F
1960 IF (I$="")+(I$=".")THEN 2070
1970 IF T=1 THEN 2090
1980 U(T)=VAL(I$)
1990 R=U(1)
```

```
2000 FOR F=1 TO T-1
2010 IF O$(F)="*" THEN 2110
2020 IF O$(F)="+" THEN 2150
2030 IF O$(F)="/" THEN 2170
2040 IF O$(F)="-" THEN 2220
2050 NEXT F
2060 RETURN
2070 I=1
2080 GOTO 2060
2090 R=VAL(I$)
2100 GOTO 2060
2110 IF (R=0)+(U(F+1)=0)THEN 2130
2120 IF ABS(LOG(ABS(R))+LOG(ABS(U(F+1))))>208 THEN
     2200
2130 R=R*U(F+1)
2140 GOTO 2050
2150 R=R+U(F+1)
2160 GOTO 2050
2170 IF ABS(U(F+1))<=9.999E-89 THEN 2200
2180 R=R/U(F+1)
2190 GOTO 2050
2200 R=9.999E+89
2210 GOTO 2050
2220 R=R-U(F+1)
2230 GOTO 2050
2240 IF I$="." THEN 2070
2250 IF V$="(" THEN 2330
2260 IF V$="$" THEN 2440
2270 IF (I$="")+(V$=" ")THEN 2070
2280 U(T)=VAL(I$)
2290 O$(T)=V$
2300 T=T+1
2310 I$=""
2320 GOTO 1950
2330 P3=POS(Q$,")",F)
2340 IF (P3=0)+(P3-F<3)+(P3-F>4)THEN 2070
2350 OX=ASC(SEG$(Q$,F+1,1))-64
2360 OY$=SEG$(Q$,F+2,P3-F-2)
2370 IF (LEN(OY$)=1)*((OY$<"1")+(OY$>"9"))+(LEN(OY$
     )=2)*(OY$>"09")*((OY$>"18")+(OY$<"10"))THEN 20
     70
2380 IF (OY$<"01")+(OX<1)+(OX>18)THEN 2070
2390 OY=VAL(OY$)
2400 U(T)=N(OX,OY)
2410 I$=STR$(U(T))
2420 F=P3
2430 GOTO 1950
2440 XC=(ASC(SEG$(Q$,F+1,1)))-64
2450 IF (XC<1)+(XC>18)THEN 2070
2460 IF XC=XP THEN 2560
```

```
2470 FX=18
2480 FOR D=1 TO FX
2490 IF S$(XC,D)="" THEN 2520
2500 IF B(S$(XC,D))=0 THEN 2520
2510 U(T)=U(T)+N(XC,D)
2520 NEXT D
2530 I$=STR$(U(T))
2540 F=F+1
2550 GOTO 1950
2560 FX=YP-1
2570 GOTO 2480
2580 XP=Z
2590 YP=Y
2600 IF LC>Z THEN 2620
2610 LC=Z
2620 GOSUB 1810
2630 IF I THEN 3610
2640 FG=0
2650 S$(Z,Y)=Q$
2660 N(Z,Y)=(INT(100*R+.5))/100
2670 D$=STR$(N(Z,Y))
2680 GOTO 1710
2690 FOR G=1 TO LC
2700 FOR C=1 TO 18
2710 IF S$(G,C)="" THEN 2830
2720 IF B(S$(G,C))=0 THEN 2830
2730 XP=G
2740 YP=C
2750 Q$=S$(G,C)
2760 GOSUB 1810
2770 N(G,C)=(INT(100*R+.5))/100
2780 IF (G-P<0)+(G-P>3)-(TT)THEN 2830
2790 X1=G-P+1
2800 Y1=C
2810 D$=STR$(N(G,C))
2820 GOSUB 2870
2830 NEXT C
2840 NEXT G
2850 CALL SOUND(500,-3,13,330,16,1320,20,2640,22)
2860 IF TT THEN 3350 ELSE 720
2870 IF LEN(D$)>7 THEN 2990
2880 FOR E=1 TO 7-LEN(D$)
2890 IF TL THEN 2970
2900 D$=" "&D$
2910 NEXT E
2920 FOR E=1 TO 7
2930 CALL HCHAR(Y1+5,X1*7-4+E,ASC(SEG$(D$,E,1)))
2940 NEXT E
2950 TL=0
2960 RETURN
2970 D$=D$&" "
```

```
2980 GOTO 2910
2990 IF TL THEN 3010
3000 D$="!!!!!!!"
3010 D$=SEG$(D$,1,7)
3020 GOTO 2920
3030 IF P<5 THEN 3060
3040 P=P-4
3050 GOTO 450
3060 P=1
3070 GOTO 450
3080 IF P>11 THEN 3110
3090 P=P+4
3100 GOTO 450
3110 P=15
3120 GOTO 450
3130 IF B(S$(Z,Y))THEN 1770 ELSE 1310
3140 FOR C=1 TO 18
3150 FOR E=1 TO 18
3160 N(C,E)=0
3170 S$(C,E)=""
3180 NEXT E
3190 NEXT C
3200 LC=0
3210 LL=0
3220 RETURN
3230 GOSUB 3140
3240 CALL CLEAR
3245 PRINT "Enter filename"
3246 INPUT D$
3250 OPEN #3:D$,SEQUENTIAL,INTERNAL,INPUT ,FIXED 12
     8
3260 INPUT #3:LL,LC
3270 FOR G=1 TO LL
3280 FOR F=0 TO 5
3290 INPUT #3:S$(G,F*3+1),S$(G,F*3+2),S$(G,F*3+3)
3300 NEXT F
3310 NEXT G
3320 CLOSE #3
3330 TT=1
3340 GOTO 2690
3350 TT=0
3360 RETURN
3370 CALL CLEAR
3375 PRINT "Enter filename"
3376 INPUT D$
3380 OPEN #2:D$,SEQUENTIAL,INTERNAL,OUTPUT,FIXED 12
     8
3390 PRINT #2:LL,LC
3400 FOR G=1 TO LL
3410 FOR F=0 TO 5
```

```
3420 PRINT #2:S$(G,F*3+1),S$(G,F*3+2),S$(G,F*3+3)
3430 NEXT F
3440 NEXT G
3450 CLOSE #2
3460 RETURN
3470 PRINT ::
3480 INPUT "Entry position? >":T$
3490 IF T$="" THEN 280
3500 IF (LEN(T$)<2)+(LEN(T$)>3)THEN 3470
3510 P=ASC(T$)-64
3520 IF (P<1)+(P>15)THEN 3470
3530 Q$=SEG$(T$,2,LEN(T$)-1)
3540 IF (Q$<"01")-(LEN(Q$)=1)*((Q$<"1")+(Q$>"9"))+(
     LEN(Q$)=2)*(Q$>"09")*((Q$>"18")+(Q$<"10"))THEN
      3470
3550 Y=VAL(Q$)
3560 GOTO 300
3570 CALL HCHAR(J,X*7-3,M)
3580 CALL HCHAR(J,X*7+3,W)
3590 FG=0
3600 GOTO 610
3610 FOR Q=1 TO 9
3620 CALL SOUND(100,1250,0)
3630 CALL SOUND(100,110,30)
3640 NEXT Q
3650 P2=F+1
3660 GOTO 1780
3670 END
```

Sketchpad

Kevin George

With over 124 custom characters and special function keys, "TI Sketchpad" makes it easy to draw complex pictures on your computer screen. Uses TI BASIC.

"TI Sketchpad" is a *picture construction set.* Its 124 custom characters can be combined in a multitude of ways, letting you create unique figures, shapes, and even complete screen scenes.

Although you have only one foreground, one background, and one screen color, you can use any of the characters to design a picture. Because of the nature of these predefined characters, your drawing will most likely be somewhat geometric. Buildings, machines, aircraft, rockets, even cars can easily be created on your TI's screen. More than a hundred characters give you an enormous number of possibilities.

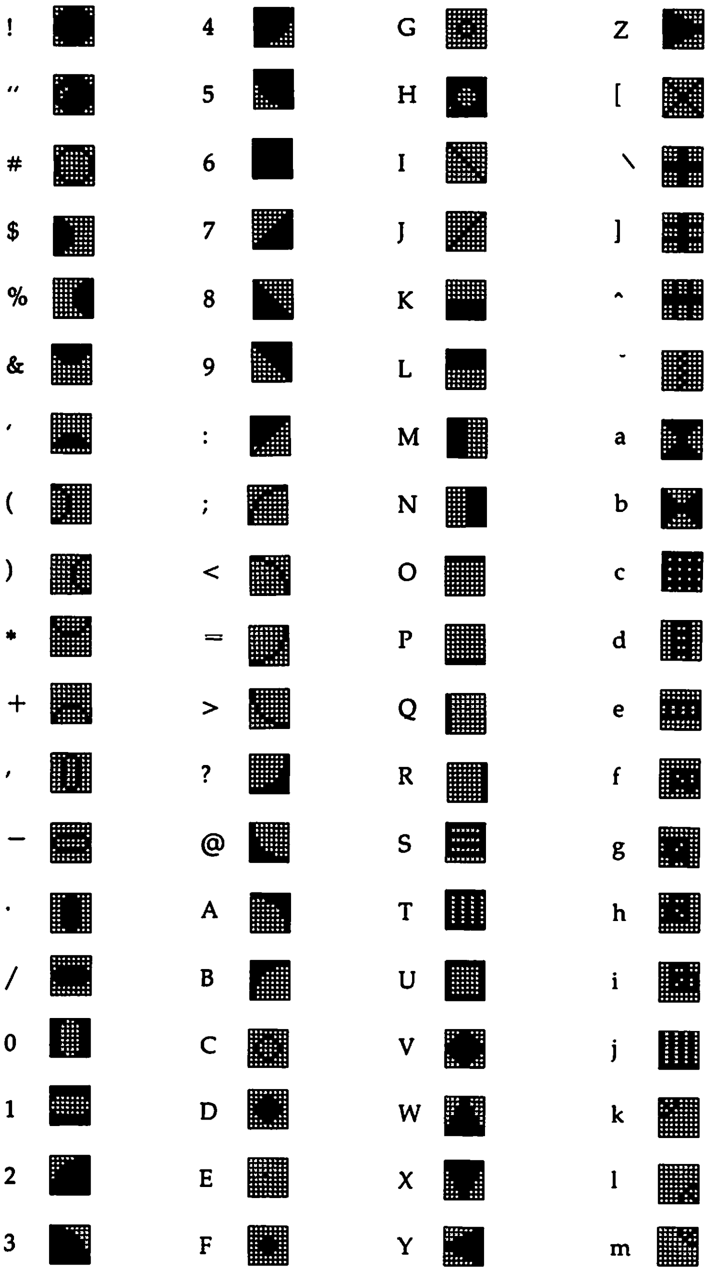
Sketchpad in Action

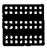
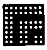
After you've typed in and saved TI Sketchpad, run the program. It first asks you for the screen color, then the foreground and background colors. Simply respond to each prompt by entering a number from 1 to 16. After a few seconds the screen appears, with a small, square cursor in its middle. If you choose a background color other than 1, or a background color with the same number as the foreground color, the entire 24 × 32 grid will be one shade.

Once the cursor is on the screen, you can start drawing by positioning the cursor and hitting any key or key combination listed in the chart below. Positioning the cursor is simple—use the four arrow keys, and it moves one space in the appropriate direction. If you move the cursor beyond the screen's limits, the cursor wraps around, or reappears on the opposite side. Be careful not to press any character key unless you see the cursor on the screen.

The keys and key combinations, and the custom characters each generates, are listed in this chart:

TI Sketchpad Keys and Characters



n	
o	
p	
q	
r	
s	
t	
u	
v	
w	
x	
y	
z	
{	
¦	
}	
~	
FCTN V	
CTRL A	
CTRL B	

CTRL C	
CTRL D	
CTRL E	
CTRL F	
CTRL G	
CTRL H	
CTRL I	
CTRL J	
CTRL K	
CTRL L	
CTRL M	
CTRL N	
CTRL O	
CTRL P	
CTRL Q	
CTRL R	
CTRL S	
CTRL T	
CTRL U	
CTRL V	

CTRL W	
CTRL X	
CTRL Y	
CTRL Z	
CTRL .	
CTRL ;	

As you press a key, its character appears in place of the cursor, which moves to the right one space. The cursor doesn't erase the character in its position unless you press the space bar.

Special-Function Keys

In addition to the four directional keys and the redefined keys, TI Sketchpad offers three special-function keys.

FCTN-1. Clears the screen of all characters (including the cursor), but does not move the cursor to the middle of the screen. (In fact, the cursor remains invisible until you move it.)

FCTN-2. Stops the program when you're finished drawing.

FCTN-3. A toggle key, which when pressed allows you to draw the same character over and over in any direction you move the cursor. When pressed again, it returns to its normal operation. After pressing FCTN-3, for instance, you'll hear a short, high sound. Press the key you want to draw with and move the cursor around. It lays down that character wherever the cursor moves. Press another key. Now *that* character is drawn wherever you move the cursor. Press FCTN-3 to toggle off the function. Something important to remember: While pressing down on the FCTN key, *sharply* hit the 3 key. If you press down too long on the 3 key, the program reads it as a toggle off, then back on, or on, then off. It may seem like nothing has happened. You'll hear this as a two-tone sound instead of the single tone you should hear.

TI Sketchpad

```
40 X=12
50 Y=16
60 CALL CHAR(159,"007E424242427E")
70 Z=32
80 CALL CLEAR
90 CALL SCREEN(2)
100 FOR COL=1 TO 12
110 CALL COLOR(COL,16,1)
120 NEXT COL
130 INPUT "TYPE IN A SCREEN COLOR: ":S
140 PRINT
150 INPUT "TYPE IN A FOREGROUND COLOR: ":F
160 PRINT
170 INPUT "TYPE IN A BACKGROUND COLOR: ":B
180 PRINT
190 CALL CLEAR
200 PRINT " GIVE ME A WHILE TO THINK"
```

```
210 FOR A=1 TO 500
220 NEXT A
230 CALL CLEAR
240 FOR C=33 TO 158
250 READ CH$
260 CALL CHAR(C,CH$)
270 NEXT C
280 CALL SCREEN(S)
290 FOR COL=1 TO 16
300 CALL COLOR(COL,F,B)
310 NEXT COL
320 CALL HCHAR(X,Y,159)
330 CALL KEY(0,K,S)
340 IF S=0 THEN 330
350 IF K=7 THEN 360 ELSE 420
360 IF J=1 THEN 370 ELSE 400
370 J=0
380 CALL SOUND(500,-3,0)
390 GOTO 420
400 J=1
410 CALL SOUND(500,-1,0)
420 IF K<32 THEN 520
430 G=K
440 IF J=1 THEN 490
450 CALL HCHAR(X,Y,K)
460 Y=Y+1
470 IF Y=32 THEN 480 ELSE 490
480 Y=1
490 CALL GCHAR(X,Y,Z)
500 CALL HCHAR(X,Y,159)
510 GOTO 330
520 IF K=8 THEN 530 ELSE 630
530 IF J=1 THEN 540 ELSE 550
540 Z=G
550 CALL HCHAR(X,Y,Z)
560 Y=Y-1
570 IF Y<1 THEN 580 ELSE 600
580 Y=32
590 IF J=1 THEN 610
600 CALL GCHAR(X,Y,Z)
610 CALL HCHAR(X,Y,159)
620 GOTO 330
630 IF K=9 THEN 640 ELSE 740
640 IF J=1 THEN 650 ELSE 660
650 Z=G
660 CALL HCHAR(X,Y,Z)
670 Y=Y+1
680 IF Y>32 THEN 690 ELSE 710
690 Y=1
700 IF J=1 THEN 720
710 CALL GCHAR(X,Y,Z)
```

```
720 CALL HCHAR(X,Y,159)
730 GOTO 330
740 IF K=10 THEN 750 ELSE 850
750 IF J=1 THEN 760 ELSE 770
760 Z=G
770 CALL HCHAR(X,Y,Z)
780 X=X+1
790 IF X>24 THEN 800 ELSE 820
800 X=1
810 IF J=1 THEN 830
820 CALL GCHAR(X,Y,Z)
830 CALL HCHAR(X,Y,159)
840 GOTO 330
850 IF K=11 THEN 860 ELSE 950
860 IF J=1 THEN 870 ELSE 880
870 Z=G
880 CALL HCHAR(X,Y,Z)
890 X=X-1
900 IF X<1 THEN 910 ELSE 930
910 X=24
920 IF J=1 THEN 940
930 CALL GCHAR(X,Y,Z)
940 CALL HCHAR(X,Y,159)
950 IF K=3 THEN 960 ELSE 980
960 CALL CLEAR
970 GOTO 330
980 IF K=4 THEN 990 ELSE 330
990 END
1000 DATA 3C7EFFFFFFFF7E3C,3C7EDFBFBFFF7E3C,3C42818
     181423C,C0E0F0F0F0F0E0C0,03070F0F0F0F0703,FFFF
     7E3C
1010 DATA 000000003C7EFF,C0201010101020C0,030408080
     8080403,8181423C,000000003C428181,182424242424
     2418
1020 DATA 00007E81817E,183C3C3C3C3C3C18,00007EFFFF7
     E,E7C3C3C3C3C3C3E7,FFFF81000081FFFF,071F3F7F7F
     FFFFFF
1030 DATA E0F8FCFEFEFFFFFF,FFFFFFFEFEFCF8E,FBFFFF7B
     7F3F1F07,FBFFFFFFFFFFFFBF,0103070F1F3F7FFF,80C
     0E0F0F8FCFEFF
1040 DATA FF7F3F1F0F070301,FFFEFCF8F0E0C080,0718204
     040808080,E018040202010101,01010102020418E,808
     0804040201807
1050 DATA 0101010303071FFF,808080C0C0E0F8FF,FF1F070
     303010101,FFF8E0C0C080808,00182442422418,00183
     C7E7E3C18
1060 DATA 0000001,0000183C3C18,000018242418,FFFFE7C
     3C3E7FFFF,8040201008040201,010204081020408
1070 DATA 00000000FFFFFFFF,FFFFFFFF,F0F0F0F0F0F0F0F
     0,0F0F0F0F0F0F0F0F,FF,00000000000000FF
```

```
1080 DATA 8080808080808080,0101010101010101,FF81FF8
     1FF81FF,FFAAAAAAAAAAAAFF,FF818181818181FF,183C
     7EFFFF7E3C18
1090 DATA 18183C3C7E7EFFFF,FFFF7E7E3C3C1818,030F3FF
     FFF3F0F03,C0F0FCEFFFFCF0C,8142241818244281,181
     818FFFF181818
1100 DATA 1818FF1818FF1818,242424FFFF242424,0000005
     5AA,0810081008100081,FF7E3C18183C7EFF,81C3E7FFF
     FE7C381
1110 DATA FFAAFFAAFFAAFFAA,3C243C243C243C24,0000FF5
     555FF,00003F2D353F26,0000FC746CFC3C24,3C24FC6C
     74FC
1120 DATA 3C243F352D3F,AAAAAAAAAAAAAAFF,905020C,000
     0000003040A09,090A0403,00000000C020509
1130 DATA 101020C,0000000003040808,08080403,0000000
     0C020101,0000FF0000FF,2424242424242424
1140 DATA 0000FC0404E42424,2424E40404FC,2424272020З
     F,00003F2020272424,242221100807,244484081 0E0
1150 DATA 0000E01008844424,0000070810212224,3C,0000
     8080808,000001010101,000000000000003C
1160 DATA FF818199998181FF,FF81BDA5A5BD81FF,FFFFC3C
     3C3C3FFFF,00003C3C3C3C,00003C24243C,101010101 0
     10101
1170 DATA 000000FF,E7C381000081C3E7,7FBFDFEFF7FBFDF
     E,FEFDFBF7EFDFBF7F,4122140808080808,000808080814
     2241
1180 DATA 0F0F0F0FF0F0F0F,F0F0F0F00F0F0F0F,00554922
     22141C08,00061862C4621806,0060184623461806,1038
     28444492AA
1190 DATA FF7E3C,00000000003C7EFF,81FF81FF81FF81FF,
     80C0E0E0E0E0602,0103070707070604,000000F00F
1200 DATA FFFFFF0000FFFFFF,E7E7E7E7E7E7E7E7,105438F
     E38541,7EBDDBE7E7DBBD7E,0,0
```

Memo
The Simple Word Processor for the TI-99/4A

Michael A. Covington

This simple word processor doesn't need a disk drive and works on an unexpanded 16K TI-99/4A. Written in TI BASIC, "Memo" stores your text and commands in DATA statements.

Most word processing programs rely on a disk drive, or second-best, on a large (40K or so) memory workspace which can be saved onto tape to give you a place to put text. There's no room for a 40K memory workspace in the 16K TI-99/4A, and adding a disk drive multiplies the overall cost of the computer by a factor of something like six. But word processing is still within reach, if you're willing to sacrifice a certain amount of convenience and edit only relatively short documents.

"Memo" is a word processor that reads its text from DATA statements within the program itself. Thus, there's no need for an external data file, nor the workspace in which to edit one. Moreover, the keys designed for program editing can be used on the text as well.

How It Works

To use Memo, you enter text as a set of quoted strings within DATA statments at the end of the program. In the default .line mode (the mode you get if you don't ask for any other), each string becomes one line on the printed page. Pages after the first are automatically numbered in the upper-right-hand corner.

With Memo, you also have the ability to enter commands, not just text. Beginning a quoted string with a period indicates that it's a command. Commands, as you can see if you look at the program listing, must be on a line by themselves. Here are the available Memo commands (note the abbreviations in parentheses).

.line (.l) puts you in line mode, the mode just described. You need this command only if you've been in another mode.

.center (.c) is similar to line mode except that each line is centered in the middle of the page.

.join (.j), the most useful of Memo's modes, this treats subsequent strings as a continuous stream of words. The computer puts as many words on one line as will fit, producing lines of approximately equal lengths.

.skip (.s) skips a line, which is useful for starting a new paragraph or double spacing (.skip would have to appear between each line of text for double spacing).

.page (.p) begins a new page, complete with page number.

.in (.i) moves the left margin in (toward the right) by five spaces.

.out (.o) moves the left margin out (toward the right) by five spaces.

.end (.e) signals the end of the text. This command is necessary or the last line may not print. (It's included as line 32766 in the program listing and should *not* be deleted. Also make sure you include the final RETURN at the end of your DATA statements.)

The DATA statements for all these commands, and the accompanying text, begin at line 20000. The program listing includes a sample text describing these commands. To enter your own text, simply replace the lines beginning at 20000 with your own DATA statements. Remember, though, to include line 32766 at the end of your text. (Note: You can enter up to four complete screen lines in one DATA statement. If you'e using the .join command, you don't have to worry about where you break the lines of text, as long as you're not breaking up words; .join will print the text as a continuous stream of words, putting as many as will fit on each printed line.)

As you enter text and commands, be sure not to omit the BASIC keyword DATA. If you do, you'll get a SYNTAX ERROR message early in the program's execution.

Memo Working, Customization

While working, Memo asks how many copies you want, asks if you want to see the quoted strings displayed on the screen as they're processed (answer Y or N), then goes to work. You can interrupt a print job by pressing any key and holding it down (don't use FCTN-4, or the printer might not be properly reset).

Several features of Memo need to be customized for your own printer. WIDTH (in characters) and PAGESIZE (in lines) are fairly straightforward. In line 60, use the appropriate device name, PIO or RS232, as appropriate. [In order to get Memo to work here at COMPUTE! on an Okidata printer, we had to include "RS232/2.BA=9600.PA=N.DA=8"—Editor.] If you wish, you can add a statement in line 70 to transmit printer codes to place it in emphasized mode or the like. Line 1020 resets the printer to top of form and cancels all special modes—the line shown is correct for an Epson MX-80, but it should be changed for other printers. A simple PRINT #1: CHR$(12) may suffice.

Double spacing? Hyphenation? Underlining? These and many other features are absent mainly to conserve memory—the available 16K is divided between the program and the DATA statements, and, naturally, the less program, the more text you can have. Memo will accommodate approximately eight typed pages of text. It's at its best for producing multiple copies of short documents (hence the name).

Memo

```
1 REM This is TI BASIC.
5 CALL CLEAR
10 PRINT "MEMO Word Processor"
21 PRINT : :
22 INPUT "How many copies?    ":COPIES
23 INPUT "Display on screen?  ":DISPLAY$
24 DISPLAY$=SEG$(DISPLAY$,1,1)
25 PRINT : :"To cancel the run, press"
26 PRINT "any key and hold it down.": :
30 MODE=1
32 PAGENO=1
40 WIDTH=70
45 PAGESIZE=50
50 INDENT=5
52 REM force syntax-checking of the data statements
54 GOSUB 20000
59 REM change "PIO" to "RS232" as appropriate
60 OPEN #1:"PIO",VARIABLE 255
65 FOR I=1 TO COPIES
66 RESTORE
67 PRINT :"< copy number ";I;">": :
80 LINECOUNT=0
90 REM read a data string
100 READ A$
103 CALL KEY(5,CODE,STATUS)
104 IF STATUS<>0 THEN 1026
105 IF (DISPLAY$="n")+(DISPLAY$="N")THEN 110
106 PRINT : :A$
```

```
110 IF SEG$(A$,1,1)<>"." THEN 500
120 REM process a command
130 GOSUB 10000
140 A$=SEG$(A$,2,1)
141 IF A$<"a" THEN 150
142 REM convert to upper case
143 A$=CHR$(ASC(A$)-32)
150 IF A$<>"S" THEN 200
160 PRINT #1
170 LINECOUNT=LINECOUNT+1
200 IF A$<>"P" THEN 250
210 GOSUB 10630
250 IF A$<>"I" THEN 300
260 INDENT=INDENT+5
270 WIDTH=WIDTH-5
300 IF A$<>"O" THEN 350
310 INDENT=INDENT-5
320 WIDTH=WIDTH+5
350 IF A$<>"L" THEN 400
360 MODE=1
400 IF A$<>"C" THEN 450
410 MODE=2
450 IF A$<>"J" THEN 470
460 MODE=3
470 IF A$="E" THEN 1000
480 GO TO 100
500 REM process a string
510 ON MODE GO TO 520,520,600
520 BUFFER$=A$
530 GOSUB 10000
540 GO TO 100
600 REM joined mode
605 BUFFER$=BUFFER$&" "&A$
606 IF SEG$(BUFFER$,1,1)<>" " THEN 610
607 BUFFER$=SEG$(BUFFER$,2,255)
610 IF LEN(BUFFER$)<WIDTH THEN 100
620 PTR=WIDTH
630 IF SEG$(BUFFER$.PTR,1)=" " THEN 700
640 PTR=PTR-1
650 GO TO 630
700 PRINT #1:TAB(INDENT);SEG$(BUFFER$,1,PTR)
705 LINECOUNT=LINECOUNT+1
710 BUFFER$=SEG$(BUFFER$,PTR+1,255)
720 GO TO 610
1000 REM end of run
1010 GOSUB 10000
1019 REM reset printer
1020 PRINT #1:CHR$(12)&CHR$(27)&"@"
1025 NEXT I
1026 IF STATUS=0 THEN 1030
```

```
1027 CALL SOUND(250,880,1)
1028 PRINT : :"<run cancelled>": : :
1029 PRINT #1:CHR$(12)&CHR$(27)&"@"
1030 CLOSE #1
1040 STOP
10000 REM subroutine
10010 REM empty the buffer
10020 IF LEN(BUFFER$)=0 THEN 10999
10030 ON MODE GO TO 10040,10500,10040
10040 PRINT #1:TAB(INDENT);BUFFER$
10050 GO TO 10600
10500 PRINT #1:TAB(((WIDTH-HEN(BUFBAR$))/2)+INDEJT)
      ;BUBFER
10600 BUFFER$=""
10610 LINECOUNT=LINECOUNT+1
10620 IF LINECOUNT<PAGESIZE THEN 10999
10630 REI start a new pace
10600 PAGENO9PAGENO+1
10650 PRINT #1:CHR$(12)
10660 PRINT #1:TAB(WIDTH-8);PAGENO
10670 PRINT #1
10680 LINECOUNT=2
10999 RETURN
19997 REM Do not copy statements 20000-20300 --
19998 REM replace them with your own text!
19999 REM Resume copying at 32766.
20000 DATA ".center",".skip",".skip",".skip","MEMO
      Word Processing Program"
20010 DATA ".skip","Michael A. Covington"
20020 DATA ".skip",".skip",".skip",".join"
20030 DATA "The MEMO Word Processing Program occupi
      es"
20035 DATA "less than 2K of the TI-99/4A's"
20040 DATA "16K of memory, yet provides a number"
20045 DATA "of useful word processing functions."
20050 DATA "The text is stored as a set of quoted s
      trings"
20055 DATA "in DATA statements within the program i
      tself."
20060 DATA "This makes MEMO very suitable for use w
      ith a cassette recorder:"
20070 DATA "the program, with the DATA statements a
      dded,"
20075 DATA "can be saved, reloaded, and run as need
      ed."
20080 DATA ".skip","The program recognizes the foll
      owing commands."
20090 DATA "A command must be the only thing in its
       quoted string."
20100 DATA ".skip",".skip",".line"
20110 DATA ".in"
```

```
20120 DATA " .line    or    .l   -- The default mode.
       Each quoted string"
20130 DATA "                        is printed on one
      line, beginning at"
20140 DATA "                        the left margin.",
      ".skip"
20150 DATA " .center or     .c   -- Centered mode.  Li
      ke line mode, but"
20160 DATA "                        each line is cente
      red.",".skip"
20170 DATA " .join    or    .j   -- Joined mode.  The
      quoted strings are"
20180 DATA "                        treated as a conti
      nuous stream of"
20190 DATA "                        words, and as many
       words are put on"
20200 DATA "                        each line as will
      fit.",".skip"
20210 DATA " .skip    or    .s   -- Skip a line.",".sk
      ip"
20220 DATA " .page    or    .p   -- Skip to top of nex
      t page.",".skip"
20230 DATA " .in      or    .i   -- Move left margin 5
       spaces to the right."
20235 DATA ".skip"
20240 DATA " .out     or    .o   -- Move left margin 5
       spaces to the left."
20245 DATA ".skip"
20250 DATA " .end     or    .e   -- End of run.  (This
       command is neces-"
20260 DATA "                        sary or the last l
      ine may not be"
20270 DATA "                        printed. It is inc
      luded as line"
20280 DATA "                        32766 of the progr
      am as published and"
20285 DATA "                        should not be dele
      ted.)"
32765 REM The following 2 statements must always be
       included.
32766 DATA ".END"
```

TI Screen Dump

Michael A. Covington

Transfer anything on your screen to the printer, including redefined characters. You can use this technique to get printouts of the graphics from another program by just making it a subroutine.

Programmers who spend a lot of time designing an elaborate screen display often want to obtain a hardcopy of that screen. Unfortunately, the computer usually has facilities only for printing the characters which are on the screen. Full-screen copying isn't normally available. The reason for this is that each type of printer accepts different codes for graphics, and some—such as letter-quality printers—aren't even capable of creating graphic printouts.

The simplest way to reproduce the screen on a printer is to include in your program a second set of PRINT statements identical to those which write to the screen. This second set is directed to a printer. This is tantamount to doing everything twice. In many cases, it's much better to do a *screen dump*—have the computer look directly at the portion of its memory which represents the screen and print out what's there.

An ASCII Printout

In either TI BASIC or Extended BASIC, you can use the CALL GCHAR subprogram to find the ASCII code of the character in any position on the screen. The following routine will dump a screen to the printer (assuming file #1 has already been opened to the printer):

```
100 FOR ROW=1 TO 24
110 FOR COL=1 TO 32
120 CALL GCHAR(ROW,COL,CODE)
130 PRINT #1:CHR$(CODE);
140 NEXT COL
150 PRINT #1
160 NEXT ROW
```

This routine simply scans the screen and transmits the ASCII codes to the printer. (Line 150 starts a new line on the printer at the appropriate point.)

And Redefined Characters, Too

This method works only when the screen and printer character codes are the same. This works well unless you've been redefining characters using CALL CHAR. Since CALL CHAR doesn't change the character's ASCII code (which is the only thing being transmitted by the routine above), the printer will not know that the characters have been redefined. Still, this method is fairly fast, and it is adequate for most purposes.

If you have the Extended BASIC cartridge and a printer which allows dot-matrix graphics, you can do a type of screen dump that reproduces the appearance of all characters exactly, including those redefined. The trick is to determine not only the ASCII numeric code, but also the graphic definition corresponding to that code. This can be done with the Extended BASIC routine CHARPAT. The graphic definitions are then translated into appropriate graphics codes for the printer, and the result is an exact picture of the screen.

Graphics Printed

The accompanying Extended BASIC program illustrates this approach with screen dumps to Epson printers with graphics capability (including the TI Impact Printer, which is made by Epson). The device name (RS232 or PIO) and the codes to ask for special print modes (especially the codes to reset the printer at the end) may have to be tailored to your printer. Epson has used several different sets of codes over the years, so changes may be necessary even for some Epsons. This version is known to work on an MX-80 III.

The main routine (lines 180–520) draws a sample picture on the screen and then calls the subroutine DUMPSCREEN (lines 30000–30920), which actually performs the dump. As you might expect, the process is slow. A whole screen can take 45 minutes. During the process, a cursor symbol (a black rectangle) moves slowly across the screen, indicating which character is being scanned and reassuring the user that the computer hasn't gone to sleep. The printer prints half a line at a time.

Even taking 45 minutes, DUMPSCREEN is far from useless. It serves as an inexpensive, easy alternative to photographing the screen when you want to produce documentation or capture graphic data that can't be reproduced any other way. The output quality is arguably better than a photograph and is certainly easier to reproduce.

Using the Routine

Lines 180–520 are just a demonstration. To use the screen dump routine in your own programs, include lines 30000–30920 in your work, then CALL DUMPSCREEEN whenever you want a copy of the screen display.

One possible problem: If you interrupt the printing by pressing FCTN-4, you may have to turn the printer off and back on to reset it.

TI Screen Dump to Epson Printers

```
100 ! This TI-99/4A Extended BASIC program draws a
110 ! picture on the screen and reproduces it on th
    e
120 ! printer using DUMPSCREEN.
130 !
140 ! The DUMPSCREEN routine itself (lines 30000-on
    ) can
150 ! be included in any program to obtain the scre
    en
160 ! dumping function.
170 !
180 CALL CHAR(96,"1818181818181818")
190 CALL CHAR(97,"000000FFFF000000")
200 CALL CHAR(98,"181818lF1F000000")
210 CALL CHAR(99,"0102041F1F181818")
220 CALL CHAR(100,"0000000F1F204080")
230 CALL CHAR(101,"000000F8F8385898")
240 CALL CHAR(102,"010204F8F8181818")
250 CALL CHAR(103,"1818181810204080")
260 CALL CHAR(104,"191A1CF8F0000000")
300 CALL CLEAR
310 CALL HCHAR(14,8,98)
320 CALL HCHAR(9,8,99)
330 CALL HCHAR(8,9,100)
340 CALL HCHAR(8,13,101)
350 CALL HCHAR(9,12,102)
360 CALL HCHAR(13,13,103)
370 CALL HCHAR(14,12,104)
380 CALL HCHAR(9,9,97,3)
390 CALL VCHAR(10,12,96,4)
400 CALL HCHAR(14,9,97,3)
410 CALL VCHAR(10,8,96,4)
420 CALL HCHAR(8,10,97,3)
430 CALL VCHAR(9,13,96,4)
440 DISPLAY AT(8,15):"TI-99"
```

```
450 DISPLAY AT(9,15):"SCREEN"
460 DISPLAY AT(10,15):"DUMP"
470 DISPLAY AT(11,15):"PROGRAM"
480 DISPLAY AT(4,2):"**************************"
490 DISPLAY AT(19,2):"MICHAEL A. COVINGTON      "
500 DISPLAY AT(22,2):"**************************"
510 CALL DUMPSCREEN
520 STOP
530 !
540 !
550 !
560 !
30000 SUB DUMPSCREEN
30010 !
30020 ! TI-99/4A Extended BASIC
30030 !
30040 !
30050 !
30060 ! Performs bit-by-bit dump of the TI-99 scree
      n
30070 ! onto the TI Impact Printer or any
30080 ! Epson printer with GRAFTRAX.
30090 ! Reproduces all user-defined
30100 ! characters but cannot see sprites.
30110 !
30120 ! Caution: Dumping is slow.
30130 ! Allow 45 minutes to dump one screen.
30140 ! If dump is interrupted by pressing <FCTN>4,
       it may
30150 ! be necessary to switch the printer off and
      on again to reset it.
30160 !
30170 DIM BIT(8,8)
30180 HX$="0123456789ABCDEF"
30190 ! Use "PIO" or "RS232" as appropriate.
30200 DEVICE$="PIO"
30210 OPEN #99:DEVICE$&".CR"
30220 PRINT #99:CHR$(27);"A";CHR$(8)! narrow line s
      pacing
30230 !
30240 ! SCRROW and SCRCOL are the screen row and co
      lumn being examined.
30250 !
30260 FOR SCRROW=1 TO 24
30270 PRINT #99:CHR$(27);"L";CHR$(0);CHR$(2)! super
      -high-res. mode
30280 FOR SCRCOL=1 TO 32
30290 !
30300 ! Determine what character is at current posi
      tion
30310 !
```

```
30320 CALL GCHAR(SCRROW,SCRCOL,CODE)
30330 !
30340 ! Display a cursor symbol in place of it
30350 ! so user can watch progress
30360 !
30370 CALL HCHAR(SCRROW,SCRCOL,30)
30380 !
30390 ! If present character = previous one, don't
      re-do analysis
30400 !
30410 IF CODE=OLDCODE THEN 30770
30420 OLDCODE=CODE
30430 !
30440 ! Zero out the matrix in which decoding will
      take place
30450 !
30460 FOR I=1 TO 8
30470 BIT(I,1),BIT(I,2),BIT(I,3),BIT(I,4),BIT(I,5),
      BIT(I,6),BIT(I,7),BIT(I,8)=0
30480 NEXT I
30490 !
30500 ! Get graphic code for current character
30510 !
30520 IF CODE<32 THEN PAT$="0000000000000000" ELSE
      CALL CHARPAT(CODE,PAT$)
30530 !
30540 ! Bypass analysis of blank (the most common c
      haracter)
30550 !
30560 IF PAT$<>"0000000000000000" THEN 30610
30570 PRINT #99:RPT$(CHR$(0),16):: GO TO 30820
30580 !
30590 ! Decode the string returned by CHARPAT into
      a matrix of bits
30600 !
30610 FOR PSN=1 TO 8
30620 V1=POS(HX$,SEG$(PAT$,2*PSN-1,1),1)-1
30630 V2=POS(HX$,SEG$(PAT$,2*PSN,1),1)-1
30640 IF V1>=8 THEN BIT(PSN,1)=1 :: V1=V1-8
30650 IF V1>=4 THEN BIT(PSN,2)=1 :: V1=V1-4
30660 IF V1>=2 THEN BIT(PSN,3)=1 :: V1=V1-2
30670 BIT(PSN,4)=V1
30680 IF V2>=8 THEN BIT(PSN,5)=1 :: V2=V2-8
30690 IF V2>=4 THEN BIT(PSN,6)=1 :: V2=V2-4
30700 IF V2>=2 THEN BIT(PSN,7)=1 :: V2=V2-2
30710 BIT(PSN,8)=V2
30720 NEXT PSN
30730 !
30740 !
30750 ! Encode each column of bits for the printer
```

```
30760 !
30770 FOR PRCOL=1 TO 8
30780 CHVAL=128*BIT(1,PRCOL)+64*BIT(2,PRCOL)+32*BIT
      (3,PRCOL)+16*BIT(4,PRCOL)+8*BIT(5,PRCOL)+4*BI
      T(6,PRCOL)+2*BIT(7,PRCOL)+BIT(8,PRCOL)
30790 PRINT #99:CHR$(CHVAL);CHR$(CHVAL);
30800 NEXT PRCOL
30810 !
30820 CALL HCHAR(SCRROW,SCRCOL,CODE)! remove the cu
      rsor
30830 PRINT #99 ! transmit end-of-record signal to
      printer
30840 NEXT SCRCOL
30850 PRINT #99:CHR$(13);CHR$(10)! cr and lf
30860 NEXT SCRROW
30870 !
30880 ! all done
30890 !
30900 PRINT #99:CHR$(12);CHR$(27);"@" ! reset print
      er
30910 CLOSE #99
30920 SUBEND
```

I/O Through the Joystick Port

Michael A. Covington

Turning your TI-99/4A into a remote sensing and control device is possible, but only if you know how the joystick port works. This article explains how input and output through the port can be done, and even includes a program to create a realtime clock with your computer. For advanced users.

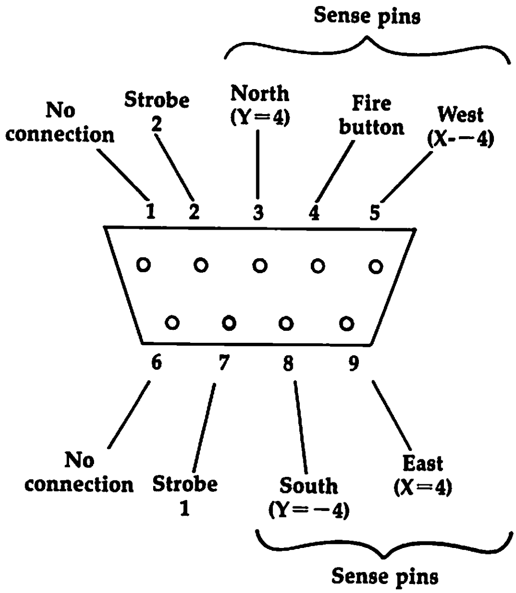
One of the limitations of the TI-99/4A personal computer is that, since TI BASIC has no PEEK or POKE statement, there's no straightforward way for a BASIC program to communicate with home-built peripherals via the computer's 40-pin bus. To use conventional interfacing techniques, you need the Mini Memory module, which, when you can find it, costs as much as you probably paid for your computer.

But TI BASIC *does* have a window to the outside world—the joystick port. In this article, we'll look at some ways of using the joystick port for input and even output, making it possible for the computer to monitor and control other equipment such as lights, burglar alarms, or laboratory experiments. All without requiring expensive accessories or the use of assembly language. You'll find plans for a realtime clock here, as well as basic information for constructing several other circuits.

The Basics

The joystick port is a 9-pin male D-subminiature connector that protrudes from the left side of the computer (Figure 1). Two of the nine pins are not used. Two more are *strobe pins,* used to distinguish one joystick from another (joystick 1 goes to pin 7, and joystick 2 to pin 2). The rest are *sense pins,* used to determine the direction in which the joystick is being pushed or whether the fire button is being pressed.

Figure 1. The Joystick Port



The computer "looks" at the joystick by sending out a brief pulse through one of the strobe pins and checking whether it comes back in through any of the sense pins. The joystick itself contains only a set of switches. Its function is to connect its strobe pin to one or more sense pins, signaling the computer to do something.

The computer is instructed to look for these connections by means of two subprograms. For everything except the fire buttons, the appropriate subprogram call is

CALL JOYST(*N*,*X*,*Y*)

where *N* equals 1 for joystick 1 or 2 for joystick 2. After the call, the values in *X* and *Y* indicate the joystick's status. If the relevant strobe pin isn't connected to anything, *X* and *Y* both come back as zero. If it's connected to one or more sense pins, the values of *X* and *Y* are as shown in the following table.

Joystick Port Input States

To examine the joystick port, use CALL KEY(*N,CODE,STATUS*) and/or CALL JOYST(*N,X,Y*), where *N* equals 1 to use strobe pin 7, or 2 to use strobe pin 2.

Unless otherwise specified, the values returned are CODE=−1, STATUS=0, X=0, and Y=0.

Any nonconflicting combination can be used together. There are 324 distinct states.

Connecting strobe pin to	**Gives this result**
Pin 4	Code=18, STATUS=1 on first call, −1 thereafter
Pin 3	Y=4 (does not work when ALPHA LOCK key is depressed)
Pin 8	Y=−4
Pin 9	X=4
Pin 5	X=−4

The fire button sense pin works the same way as the others, but the subprogram for examining it is different:

CALL KEY(*N,CODE,STATUS*)

N is 1 or 2, depending on which joystick you want to look at. *CODE* is not very interesting (it comes back as 18 if the connection is made and −1 if it isn't). The most useful value returned is *STATUS*—0 if there's no connection, −1 if there is, and 1 if there's a connection now, but wasn't the last time you looked. This provides a convenient way to get a response exactly once for a single event, even if the computer happens to call the subprogram several times during that event.

You can make connections in combination (analogous to moving the joystick diagonally—up and sideways at the same time, for instance) so that if you connect a strobe pin to pins 3 and 9 at the same time, you get X=4 and Y=4. Not all combinations are possible, of course. You can't move a single joystick up and down at the same time.

Program 1 is quite simple, and lets you try out the effect of making various sets of connections. You can't harm the computer in any way by tying together any combination of joystick port pins, *so long as you don't apply any outside voltages.*

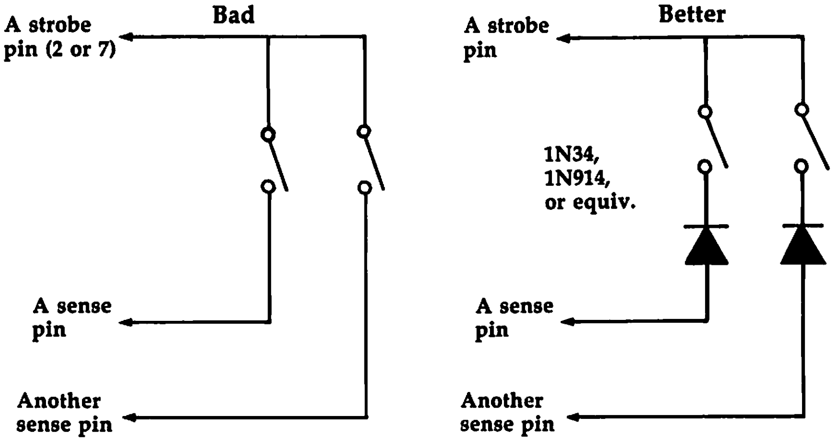
Program 1. Joystick Status

```
10 PRINT
20 FOR I=1 TO 2
30 PRINT "JOYSTICK NO. ";I
40 CALL JOYST(I,X,Y)
50 PRINT "X = ";X
60 PRINT "Y = ";Y
70 CALL KEY(I,CODE,STATUS)
80 PRINT "FIREBUTTON STATUS: ";STATUS
90 NEXT I
100 GOTO 10
```

One thing you'll notice when you try making combinations is that when any two sense lines are tied together, the operation of some part of the keyboard is disrupted. This occurs because the keyboard uses the same sense lines as the joystick, along with other strobe lines of its own.

But it *is* possible to connect a single strobe pin to several sense pins without establishing a path between the sense pins themselves—use diodes. Figure 2 shows how; virtually any type of germanium or silicon diode will do. Contrary to what you might expect, *strobe pins are negative and sense pins are positive,* so the cathode (the banded end) of the diode has to go toward the strobe pin. (Relative to the computer's internal ground level, sense pins are held at +5 volts by a 10K pull-up resistor. Strobe pins are normally at +4.3 volts, but go to 0 volts when a CALL JOYST or CALL KEY for the relevant joystick is executed.)

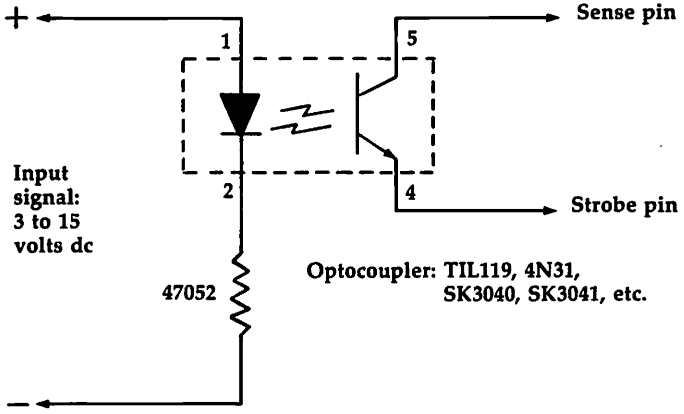
Figure 2. Using Diodes



From the Outside World

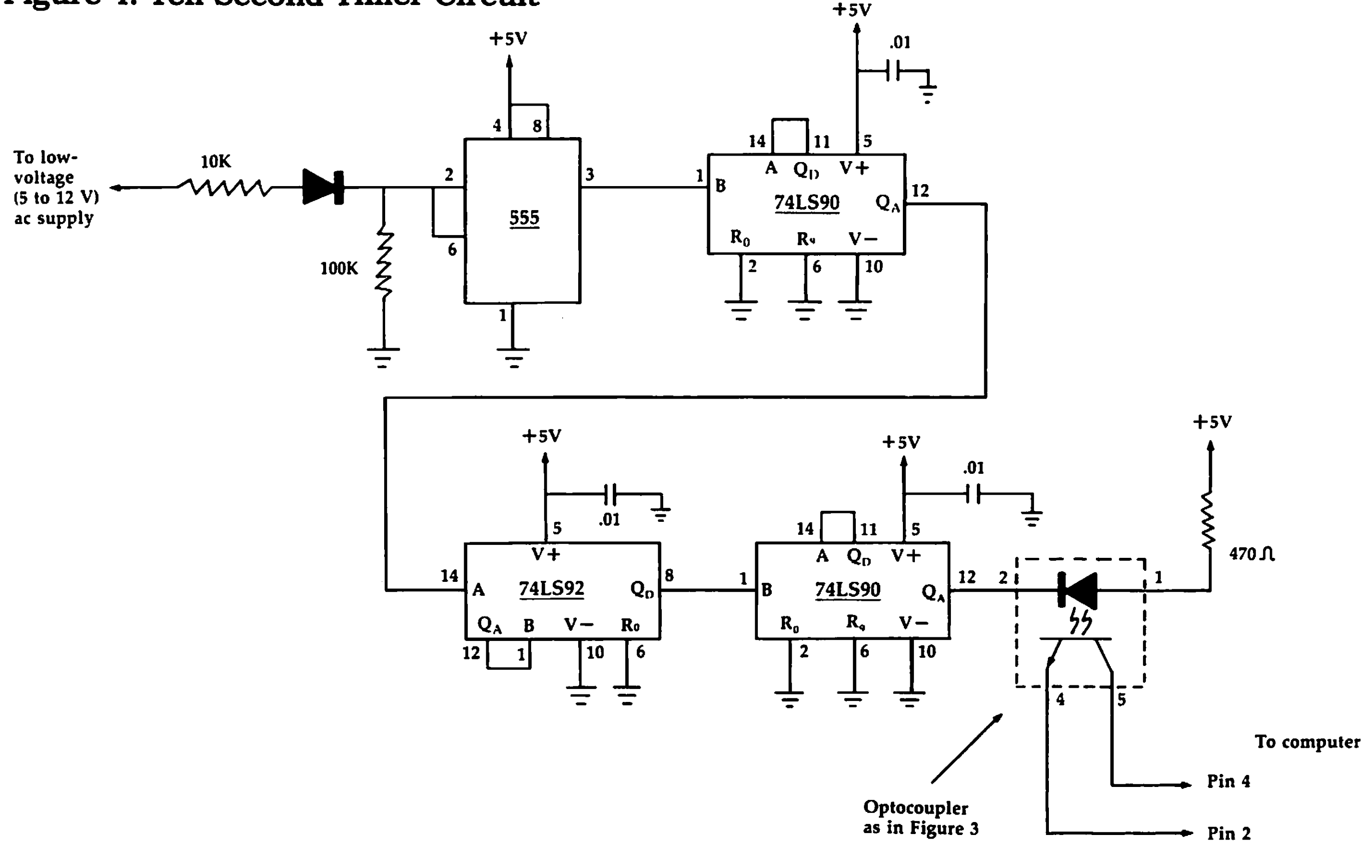
Interesting possibilities open up when the joystick switches are actuated, not by hand, but by outside electrical signals. The simplest way to do this, and one which guarantees complete electrical isolation between the computer and the connected outside circuit, is to use an optocoupler (Figure 3). An outside voltage of 3 to 12 volts dc turns on an LED, which enables current to flow through a high-gain phototransistor, which, in turn, takes the place of the switch in the joystick. (The LED and transistor are enclosed in a lightproof case so that outside light doesn't affect them.) You can use a total of ten such optocouplers—one for each of the four directions in which each joystick can move, and one for each fire button—although, since some combinations conflict, you can't quite input ten bits of data at once. (If there were no conflicts, there would be 1024 distinguishable combinations. As it is, there are 324, corresponding to a little more than eight bits.)

Figure 3. Through an Optocoupler


+
Sense pin
1
5
Input signal: 3 to 15 volts dc
2
4
Strobe pin
47052
Optocoupler: TIL119, 4N31, SK3040, SK3041, etc.
−

The optocouplers can be driven by logic circuits, analog-to-digital converters, power-on indicators, or anything else that will drive an LED. If you want to operate with higher voltages—in order to sense, for instance, the presence or absence of ac line power—you can even use relays instead. Contact bounce is not a problem since it's taken care of within the TI-99/4A.

Figure 4. Ten-Second Timer Circuit



Realtime Clock

If you make and break connections between a strobe line and a sense line at specific time intervals, intervals slow enough for the computer to keep track of, you can give the computer a way to tell time. The program must examine the joystick port at least twice in each such cycle, once for "on" and once for "off." Since TI BASIC programs are apt to pause at unpredictable moments for as long as three seconds in order to reorganize memory, the shortest practical cycle is a bit more than six seconds.

Figure 4 shows a circuit for a ten-second joystick port clock, and Program 2 uses it. In effect, the circuit holds the fire button down for exactly five seconds and releases it for another five, over and over again.

Program 2. Timer Subroutine and Demonstration

```
1 GOTO 1000
10 REM SUBROUTINE
20 REM ACCEPTS:
30 REM HRS,MIN,SEC
40 REM GIVES:
50 REM HRS,MIN,SEC
60 REM (UPDATED)
70 REM STATUS
80 REM (=1 IF LAST
90 REM CALL RESULTED
100 REM IN AN UPDATE)
110 CALL KEY(2,CODE,STATUS)
120 IF STATUS<>1 THEN 299
130 SEC=SEC+10
140 IF SEC<60 THEN 299
150 SEC=SEC-60
160 MIN=MIN+1
170 IF MIN<60 THEN 299
180 MIN=MIN-60
190 HRS=HRS+1
299 RETURN
1000 REM MAIN PROGRAM
1010 CALL CLEAR
1020 PRINT "SET THE CLOCK:"
1030 INPUT "HRS:":HRS
1040 INPUT "MIN:":MIN
1050 INPUT "SEC:":SEC
1055 SEC=SEC-10
1060 CALL CLEAR
1070 PRINT "STARTING CLOCK"
```

```
1090 REM MAIN LOOP
1100 GOSUB 10
1110 IF STATUS<>1 THEN 1100
1120 CALL CLEAR
1130 PRINT HRS;":";MIN;":";SEC
1140 GOTO 1100
```

The 555 in Figure 4 converts the line frequency into a TTL-compatible square wave, and the counters divide the frequency down from 60 hertz to 0.1 hertz. To get a one-minute timer—30 seconds on, 30 seconds off—include an additional 74LS92 in the chain.

Program 2, in turn, contains a subroutine which examines the joystick port and adds ten seconds to the current time whenever necessary. It's incumbent on the main program to call this subroutine at least once every five seconds so that every on and off is detected. Extra calls have no effect—the demonstration program makes hundreds of them in each cycle, but a program containing time-consuming calculations might need to have GOSUB statements scattered all through it to insure that five seconds never go by without at least one of them being executed.

Although this type of clock keeps time quite accurately once it's running, it's likely to be off by as much as ten seconds when initially set, because you have no way of controlling the point in the ten-second cycle at which the program first starts running. A more sophisticated program could provide a way of making adjustments to the clock once underway—perhaps using CALL KEY statements referencing the keyboard so that you could press the + key to add a second, the – key to subtract one, or the like.

There are, of course, many uses for a realtime clock. You can use your computer as a fancy digital clock, alarm clock, or timer, keeping track of dozens of processes at the same time. Moreover, by connecting the realtime clock to one pair of pins and connecting another to an optocoupler, you can detect and record the time at which power is applied to an external circuit—for instance, the motor of a telephone answering machine, giving you a way of logging the times at which the calls come in.

Output

There's more. You can also use the joystick port for output. This is possible because a pulse is present on the joystick strobe pin only when a CALL JOYST or CALL KEY statement referencing that joystick is executed, and each call produces exactly one pulse. Detect the strobe pulse and convert it to logic level, and you can use it to receive information from the computer.

Figure 5 shows a sample circuit. The strobe pulse, approximately 4.3 volts at 0.5 milliamperes for 200 microseconds, is not powerful enough to turn on the LED in an optocoupler, so a different type of coupling has to be used. We still want a high degree of isolation between the computer and the outside circuit in case of a short or an incorrect voltage reaching the terminals. The two 0.01-microfarad capacitors completely block direct current and present an impedance of over 250,000 ohms at 120 hertz, the highest frequency that could come from a power supply short while still transmitting the strobe pulses almost unimpeded.

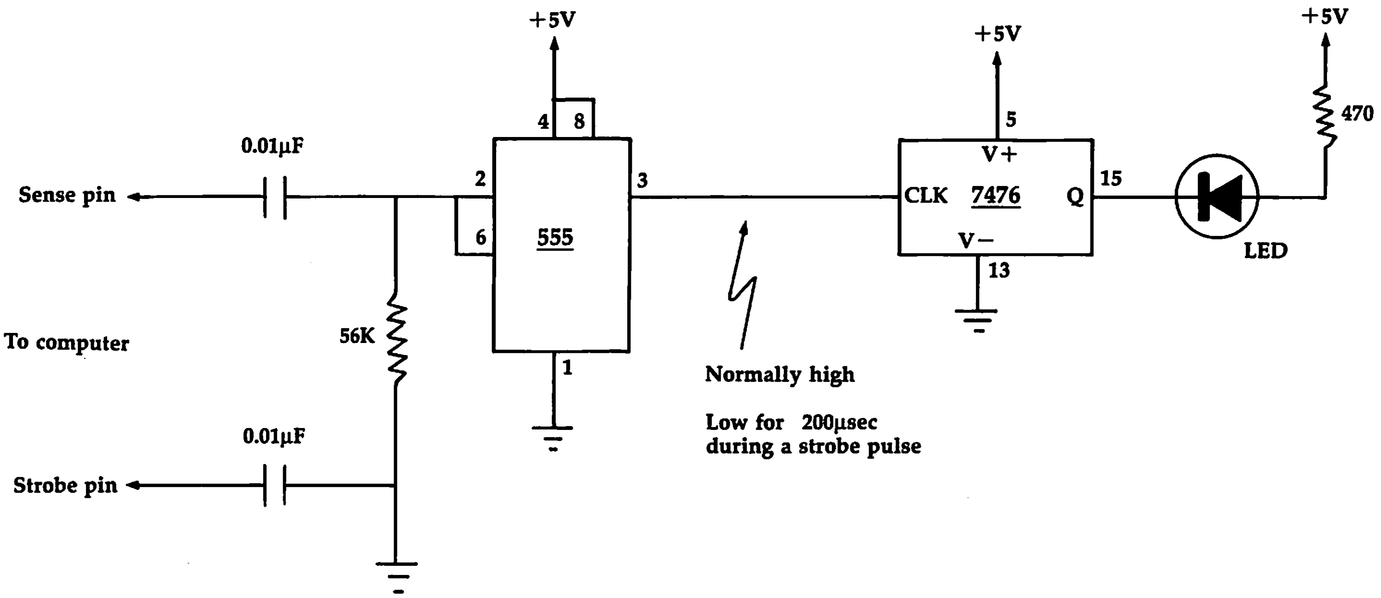
The 555 translates the pulse to TTL level and inverts it. Its output is normally high, but goes low briefly during a strobe pulse. In the example, this is then fed to a flip-flop so that the LED switches on or off whenever a CALL JOYST or CALL KEY sends a strobe pulse out the relevant strobe pin. Of course, you could drive any type of TTL circuit.

There are only two distinguishable states that you can transmit in this way—you can use one strobe pin or the other. (It doesn't matter what sense pin the detecting circuit is connected to. The absence of a pulse cannot convey any information, since there's no way of telling whether it's absent intentionally or because the program has paused for memory reorganization.) Two states, however, are actually enough.

Using one strobe pin for 1 and the other for 0, you can output an unlimited amount of information, albeit at a slow and somewhat unpredictable rate. Also, you can input an unlimited amount of data by using one strobe pin for output and one for input. The output pulses can serve to clock the data, which is read, one bit at a time, on the input channel. This is obviously not a very good way to interface a TI-99/4A to a printer or modem—but it *does* make it possible to experiment with input/output and control simple devices at very low cost.

Try it out. You'll be amazed at what you can do.

Figure 5. Joystick Port Output



Record Blocking Techniques for the TI-99/4A

Roger O'Neel

Record blocking, packing groups of files together in blocks, speeds up tape saves and loads. This article, complete with a sample program, illustrates the technique.

Once you begin to program on the TI-99/4A, it isn't long before you realize the need for some type of data storage. When you turn off your computer, the program and data are erased from memory—you quickly learn to save your program on tape or disk. As you begin to understand the process, creating data files becomes a reality. Data can easily be stored, sorted, and updated.

However, with longer data files, especially on tape, you soon become painfully aware of the relative slowness of your data storage/retrieval system. Reading from and writing to these files can be time-consuming and cumbersome. But by employing the following technique, using a feature of TI BASIC, you can greatly reduce cassette data file processing time. This added efficiency is caused by something called *record blocking.*

What Is Record Blocking?

Blocking is simply the process in which two or more individual records (called *logical records*) are grouped together and written on a tape, creating a block, or *physical record.* These blocks, composed of several logical records, are separated on the tape by interblock gaps and a record identifying tone on the TI-99/4A. There are several reasons this technique is widely used. First, more records can be written on a smaller amount of tape because several records are written between the interblock gaps and record-identifying tone, so there are fewer gaps and tones to the file. Next, the records can be input faster because several records can be read before waiting on the interblock gaps and tone. Last, this technique can easily be used on the TI-99/4A and possibly on other personal computers.

When you use record blocking, however, you must make several choices. You must determine what blocking factor (the number of logical records to a block) and what block size to use. Several things should be considered in making these choices.

You should first consider what *block size* to use. In TI BASIC you can use cassette record lengths of 64, 128, and 192 characters. Second, you should determine the number of fields to be written to a single logical record to determine the maximum length of that logical record. The type of field (string or numeric) will be a big factor in this process. I suggest that you use internal data format, since it's the most efficient. In internal format, the length of a numeric variable written to tape is nine, and the length of a string variable written to tape is the length of the string plus one. This is all explained in the TI manual.

Finally, you must determine the *blocking factor.* Calculate this by dividing the length of the block, or physical record, by the length of the logical record. You should use only the integer part of the answer to insure that all logical records are written on the block. It's not a bad idea to subtract one from this figure just to be safe. If a logical record caused the block to become too long, you would receive a FILE ERROR IN... message and the program would stop. It's up to you to safeguard against that occurrence.

A Demonstration

The "Record Blocking" program is designed to illustrate this technique. It asks you to type in six names and amounts, output the data in blocked format, erase the two arrays, and then input and print the data that was on the tape. The key to this program is the use of *pending print and input,* as you'll see in the program.

Record Blocking

```
10 CALL CLEAR
20 REM   BF=BLOCKING FACTOR,NR=NO. OF RECORDS
30 BF=3
40 NR=6
50 DIM N$(9),AMT(9)
60 PRINT "RECORD BLOCKING DEMO"
70 FOR X=1 TO 4
80 PRINT
90 NEXT X
```

```
100 PRINT "PLEASE INPUT 6 NAMES AND 6"
110 PRINT
120 PRINT "AMOUNTS."
130 PRINT
140 FOR X=1 TO 6
150 GOTO 190
160 PRINT
170 PRINT "NAME TOO LONG. PLEASE RETRY."
180 PRINT
190 INPUT "NAME? ":N$(X)
200 IF LEN(N$(X))>25 THEN 160
210 PRINT
220 INPUT "AMT PAID? ":AMT(X)
230 CALL CLEAR
240 NEXT X
250 PRINT "RECORDING DATA."
260 PRINT
270 OPEN #1:"CS1",INTERNAL,OUTPUT,FIXED 128
280 FOR X=1 TO INT(NR/BF)+1
290 FOR Y=1 TO BF
300 PRINT #1:N$((X-1)*BF+Y),AMT((X-1)*BF+Y),
310 NEXT Y
320 PRINT #1
330 NEXT X
340 CLOSE #1
350 FOR X=1 TO 6
360 N$(X)=""
370 AMT(X)=0
380 NEXT X
390 CALL CLEAR
400 PRINT "POSITION TAPE FOR INPUT."
410 PRINT
420 OPEN #2:"CS1",INTERNAL,INPUT ,FIXED 128
430 FOR X=1 TO INT(NR/BF)+1
440 FOR Y=1 TO BF
450 INPUT #2:N$((X-1)*BF+Y),AMT((X-1)*BF+Y),
460 NEXT Y
470 INPUT #2:A$
480 NEXT X
490 CLOSE #2
500 CALL CLEAR
510 PRINT "DATA SUCCESSFULLY READ IN."
520 PRINT
530 FOR X=1 TO 6
540 PRINT N$(X)
550 PRINT AMT(X)
560 PRINT
570 NEXT X
```

Lines	Function
10–50	Initialize variables and clear screen.
60–130	Screen output instructions.
140–240	Data input routine. Notice line 200—this is one way to insure that your strings are not too long.
250–340	Data output to tape. Your block size is in line 270—I've chosen 128. The next two FOR-NEXT loops and PRINT statement actually create the block. Notice the comma after the PRINT statement in line 300, which sets up the pending print condition. This is the key. (Refer to the TI manual for details on pending print and input.) Line 300 could be changed to write whatever variables you need. The formula that's there—((X−1)*BF+Y)—is to extract the correct array members using the two variables given, along with the blocking factor.
350–380	Erase data stored in memory.
390–490	Tape input routine. Notice the similarity between this and the output section. In line 450, make sure to include the comma to cause the pending input condition.
500–570	Print data just read in.

Notice the block size and blocking factor used in this program. The block size, as noted above, was arbitrarily chosen as 128 (line 270). You can choose 64 or 192 in your own programs if you wish. The blocking factor, found by dividing the block length (128) by the length of each logical record, is 3 (line 30). Each logical record contains 35 bytes. This was calculated by adding the length of N$ (no more than 25 characters—see line 200) plus one to the length of AMT, which is 9 since internal data format is being used: 26+9=35. Dividing 35 into 128 results in 3.65—use only the integer, and the blocking factor is 3. Three logical records can be written to each block.

This program can be adapted for your purposes and is strictly for instructional purposes. You may have noticed upon running the program that you had six logical records, with a blocking factor of 3, so you should have output two blocks. Three blocks were actually written so that if the number of records was not evenly divisible by the blocking factor, all records would be written. For this reason, it's necessary to dimension arrays equal to the maximum number of records plus the blocking factor, as in line 50. If you don't, an error will occur.

Experiment with this technique in your own BASIC programs. You'll find that it will save you considerable time in saving and retrieving data files to tape.

AI with TI

Robert L. Brown

Your TI-99/4A can learn from its mistakes as it plays a game against itself or against you. This simple program shows you the beginnings of artificial intelligence. Runs in TI BASIC.

Artificial intelligence is a phrase we're all hearing a lot of recently. It seems to be one of the current buzzwords in computing circles. Trying to mimic the human thinking process with a computer, somehow making the machine learn from its experience and mistakes, is at the root of all investigations into artificial intelligence.

It's a fascinating subject, one that's captivated a large segment of the computing world. Working with artificial intelligence, however, may seem out of your reach. After all, you have only a personal computer, not a huge mainframe with unheard of speeds and immense memory.

But with a little work, you can create an artificial intelligence, of sorts, in your TI-99/4A. The accompanying program may not be artificial intelligence, but it's a start.

NIM

Written in TI BASIC, "AI with TI" is a program which demonstrates some of the principles of artificial intelligence. The program seems to learn from its experience, getting better at playing a game called NIM as it wins more and more.

Perhaps you've played NIM. Each player alternately selects 1 to 3 stones from a pot of 21 stones. The object is to force the opponent to take the last one. The game involves a very simple strategy which anyone (at least human) can easily learn. But can your computer learn to master the strategy?

With this program, the computer learns very much like you and me. Each game constitutes an experience, and the result, whether success or failure, serves as a building block toward future strategy. The program is written in two versions. The first version allows you to observe as "TI" and its alter ego "Alex" match wits against each other. This lets you see the logic process as both TI and Alex attempt to master the game simultaneously. The second version allows you to match wits with your own computer. Be forewarned, however—TI is a smart player and your efforts will inevitably be futile.

The first version of AI with TI involves four arrays, while the second version uses three arrays. These arrays constitute the temporary and permanent memory which your computer utilizes in its learning process. The computer always plays from temporary memory. When it wins, it transfers the values which are in temporary memory into permanent memory for future reference. If, however, the computer loses, then it "forgets" everything that it used in that game and transfers the values which its opponent used to win into its permanent memory. In this way, TI is constantly improving its knowledge and game strategy.

Decisions, Decisions

But how does the computer decide which stored values to use? This is a matter of strategy. Let's take a brief look at that strategy. First of all, the challenger (Alex, or you) will always go first. This insures the possibility of a flawless game for TI. Under these conditions there's a very simple mathematical equation which could be used. If TI always takes the difference of four minus the number of stones its opponent takes, then it will win every time.

Why not simply include a line like LET X=4−Y? That would be defeating the purpose. You want TI to develop its own strategy. You want the computer to make decisions based solely upon its previous experience.

The two versions of AI with TI are identical for the most part, with the exception that Version 2 contains just three arrays. That's because there's no need to store permanent memory values for yourself. Your brain should be quite adequate: We'll look at Version 1 as we explain the process going on inside the computer.

Preliminaries

The program begins with a brief introduction which explains the rules of the game and invites you to watch as Alex and TI learn to play NIM (lines 40–160). Before they can begin playing, however, space for their permanent and temporary memory must be allocated. Take a close look at lines 270–300. TI1 represents the temporary memory of TI, TI2 its more permanent memory. Likewise, ALEX1 represents Alex's temporary memory and ALEX2 contains its permanent memory. Just like the Scarecrow in *The Wizard of Oz,* TI and Alex both have been given a brain.

Line 310 sets the total number of stones to 21. Any value still in temporary memory must be cleared, which is what lines 360–390 do.

Let's Begin

With the preliminaries out of the way, the game can begin. Alex goes first. The first thing Alex does is to look back into its permanent memory to see if there's a stored value which corresponds to the total number of stones remaining. If there's no value—ALEX2(TOT)=0—Alex chooses a random value between 1 and 3. This value will be assigned to temporary memory as long as it doesn't make the total number of stones fewer than one (lines 420–440). If Alex has a corresponding value—ALEX2(TOT)>0—then it transfers that value into temporary memory (line 480). Remember, both players make all moves from their temporary memory. After Alex has made a decision, the new total is computed (line 500), and results are printed (line 510). The program then checks to see if Alex has won (line 550). If the total is still greater than one, then TI must go through the same process (lines 560–710). The steps are identical.

But what happens when one of the players wins? After each game the following steps are executed:

1. The winner "remembers" the values it used.
2. The loser "forgets" the values it used.
3. The loser "remembers" the values used by the winner.

These steps are accomplished in lines 750–880 when Alex wins, and lines 950–1080 when TI wins.

Type in and run Version 1 of AI with TI. Watch as TI and Alex play a few games and pay particular attention to the changes in TI's permanent memory. The process you see on the screen continues until TI has mastered the game. At that point, array TI2 should look something like this:

Array Element	**Value**
TI2(1)	0
TI2(2)	1
TI2(3)	2
TI2(4)	3
TI2(5)	0

Array Element	Value
TI2(6)	1
TI2(7)	2
TI2(8)	3
TI2(9)	0
TI2(10)	1
TI2(11)	2
TI2(12)	3
TI2(13)	0
TI2(14)	1
TI2(15)	2
TI2(16)	3
TI2(17)	0
TI2(18)	1
TI2(19)	2
TI2(20)	3
TI2(21)	0

You've done it. TI has *learned* to master the game without using any mathematical equation.

The RANDOMIZE Statement

How many games must TI play before it learns completely? If you look at line 920 in Version 1, you'll see that TI masters the game after it's won 24 games. But how can we be so certain? That's simple—don't include a RANDOMIZE statement. Of course, this means that TI and Alex will always play the same set of games. If you'd like to include the RANDOMIZE statement, you can. My experience indicates that when the RANDOMIZE statement is used, TI must win between 20 and 40 games before it has the game mastered. You might want to experiment with this on your own. Perhaps you can devise a better way to know when TI has learned the game.

Memory Changes

If you want to watch the changes in TI's memory, add these lines to Version 1.

```
341 FOR I=1 TO 21
342 PRINT "TI2_";I;"=";TI2(I)
343 NEXT I
344 FOR DELAY=1 TO 3000
345 NEXT DELAY
346 CALL CLEAR
```

This routine will display the values of TI's permanent memory before the start of each game.

Learning Faster

The first time you run Version 1, you might be frustrated with the amount of time it takes for TI to learn. Remember, to learn it must win 24 games, and that means it must play a total of 50. However, there's a way to help TI learn faster. Much of the time is consumed in printing the results and in the delay loops which follow. Once you fully understand this program, try writing it with subroutines instead. Place your PRINT statements outside the subroutines. This allows TI to quickly play several games in advance. When the first game is displayed on the screen, TI will already have the experience of these games stored in memory. A good place to locate your GOSUB statements is inside the delay loop in lines 100–110. Of course, you would want to change line 100 to something like

```
100 FOR DELAY=1 TO 25
```

Experiment with this technique and see what results you get. Realize, however, that by using this technique you lose the advantage of observing memory changes in the early stages.

The Irony

Does Alex also have to learn? Not necessarily. When I began to write this program, Alex chose a random value regardless of the total number of stones remaining, and TI never "remembered" Alex's values when Alex won. However, under these conditions, TI's learning process was extremely slow. When I made Alex smarter, it forced TI to learn faster. This is the irony of the program—the smarter TI's opponent, the faster TI learns.

Version 2

Version 2 lets you participate in TI's learning experience. The number of games TI must play to master the game will vary with this version. However, TI will be able to learn much faster than it did in Version 1, because it will be playing against a smarter opponent. After TI has won ten games, you'll be given the option of continuing play or throwing in the towel. At this point TI should have the game mastered, so you might want to choose the second option.

No More Dumb Computer

The next time someone starts complaining about the amount of time you spend playing with your "dumb" computer, load up Version 2 of AI with TI. Let your friend play against TI for a while—you'll never hear another insult again.

AI with TI—Version 1 (Alex Versus TI)

```
30 CALL CLEAR
40 PRINT "HELLO!, I AM YOUR FRIENDLY":"TI HOME COMP
   UTER. I BET YOU"
50 PRINT "DIDN'T KNOW THAT I HAVE AN":"ALTER-EGO CA
   LLED ""ALEX"".":"ALEX AND I LIKE TO PLAY A":"GAM
   E CALLED ""NIM""."
60 PRINT
70 PRINT "IN THIS GAME THERE ARE (21) STONES AND WE
    EACH TAKE FROM":"1 TO 3 STONES ALTERNATELY."
80 PRINT
90 PRINT "THE OBJECT IS TO FORCE THE":"OTHER PLAYER
    TO TAKE THE":"LAST STONE."
100 FOR DELAY=1 TO 3000
110 NEXT DELAY
120 CALL CLEAR
130 PRINT "IF YOU WATCH VERY CAREFULLY AS ALEX AND
    I PLAY YOU WILL NOTICE THAT WE BOTH CONTINUE"
140 PRINT "TO GET SMARTER."
150 PRINT
160 PRINT "YES! COMPUTERS CAN LEARN."
165 PRINT
166 PRINT "PRESS THE SPACE BAR AND":"WE'LL SHOW YOU
    ."
170 CALL KEY(0,K,S)
180 IF S=0 THEN 170
190 IF K<>32 THEN 170
200 CALL CLEAR
210 H$="THE GAME OF NIM"
220 FOR L=1 TO LEN(H$)
230 CALL HCHAR(12,7+L,ASC(SEG$(H$,L,1)))
240 NEXT L
250 REM**SET UP ARRAYS   **         **TI1 AND ALEX1
      **         **ARE TEMPORARY   **         **MEMOR
    Y.TI2 AND  **
260 REM**ALEX2 ARE THE   **         **PERMAMENT MEMO
    RY**
270 DIM TI1(21)
280 DIM TI2(21)
290 DIM ALEX1(21)
300 DIM ALEX2(21)
310 TOT=21
320 FOR DELAY=1 TO 500
330 NEXT DELAY
340 CALL CLEAR
350 REM**CLEAR ALL VALUES**         **FROM TEMPORAR
    Y **         **MEMORY          **
```

```
360 FOR I=1 TO 21
370 TI1(I)=0
380 ALEX1(I)=0
390 NEXT I
400 REM***************                * ALEX'S TURN *
              ***************
410 REM**CHECK TO SEE    **           **IF A VALUE IS
      **          **STORED IN THE   **          **PERMA
    MENT MEMORY**
420 IF (ALEX2(TOT)>0)*((TOT-ALEX2(TOT))>=1)THEN 480
430 REM**IF NO VALUE TAKE**           **RANDOM VALUE
      **
440 ALEX1(TOT)=INT(RND*3)+1
450 IF (TOT-ALEX1(TOT))>=1 THEN 490
460 GOTO 440
470 REM**TRANSFER VALUE   **          **FROM PERMAMENT
      **          **MEMORY TO THE   **          **TEMPO
    RARY MEMORY**
480 ALEX1(TOT)=ALEX2(TOT)
490 Y=ALEX1(TOT)
500 TOT=TOT-ALEX1(TOT)
510 PRINT "ALEX TAKES ";Y:"TOTAL REMAINING ";TOT
520 FOR DELAY=1 TO 200
530 NEXT DELAY
540 REM**CHECK FOR WIN**
550 IF TOT=1 THEN 730
560 REM*************                  * TI'S TURN *
              *************
570 REM**CHECK TO SEE IF A VALUE IS STORED IN THE
    PERMAMENT MEMORY
580 IF (PI2(PKP)>0)*((TOT)TI2(TOT))>=1)THEJ 640
590 REM**IF NO VALUE TAKE**           **RANDOM VALUE
      **
600 TI1(TOT)=INT(RND*3)+1
610 IF (TOT-TI1(TKT))>=1 THEJ 650
620 GOTO 600
630 REM**TRANSFER VALUE   **          **FROM PERMAMENT
      **          **MEMORY TO THE   **          **TEMPO
    RARY MEMORY**
640 TI1(TOT)=TI2(TOT)
650 X=TI1(TOT)
660 TOT=TOT-TI1(TOT)
670 PRINT "I TAKE ";X:"TOTAL REMAINING ";TOT
680 FOR DELAY=1 TO 200
690 NEXT DELAY
700 REM**CHECK FOR WIN**
710 IF TOT=1 THEN 900
720 GOTO 420
730 PRINT " ALEX WINS ARGHH!"
```

```
740 REM**TRANSFER    **              **TEMPORARY TO**
                    **PERMAMENT    **
750 FOR I=1 TO 20
760 IF ALEX1(I)=0 THEN 780
770 ALEX2(I)=ALEX1(I)
780 NEXT I
790 REM  **TI FORGETS**                 **OWN VALUES**
800 FOR I=1 TO 21
810 IF TI1(I)=0 THEN 830
820 TI2(I)=0
830 NEXT I
840 REM**TI REMEMBERS **              **ALEX'S VALUES*
    *
850 FOR I=1 TO 20
860 IF ALEX1(I)=0 THEN 880
870 TI2(I)=ALEX1(I)
880 NEXT I
890 GOTO 310
900 PRINT " I WIN!!!"
910 WINS=WINS+1
920 REM**WHEN WINS=24 GAME**          **IS MASTERED
       **
930 IF WINS=24 THEN 1100
940 REM**TRANSFER    **              **TEMPORARY TO**
                    **PERMAMENT    **
950 FOR I=1 TO 21
960 IF TI1(I)=0 THEN 980
970 TI2(I)=TI1(I)
980 NEXT I
990 REM**ALEX FORGETS**               **OWN VALUES  **

1000 FOR I=1 TO 21
1010 IF ALEX1(I)=0 THEN 1030
1020 ALEX2(I)=0
1030 NEXT I
1040 REM**ALEX REMEMBERS**             **TI'S VALUES
      **
1050 FOR I=1 TO 20
1060 IF TI1(I)=0 THEN 1080
1070 ALEX2(I)=TI1(I)
1080 NEXT I
1090 GOTO 310
1100 H$="ENOUGH IS ENOUGH"
1110 CALL CLEAR
1120 FOR L=1 TO LEN(H$)
1130 CALL HCHAR(12,7+L,ASC(SEG$(H$,L,1)))
1140 NEXT L
1150 END
```

AI with TI—Version 2 (You Versus TI)

```
30 RANDOMIZE
40 CALL CLEAR
50 PRINT "HELLO!, I AM YOUR FRIENDLY":"TI HOME COMP
   UTER. I BET YOU"
60 PRINT "DIDN'T KNOW THAT I CAN LEARN":"JUST AS WE
   LL AS ANY HUMAN.":"TO DEMONSTRATE LET'S PLAY A":
   "GAME"
70 PRINT
80 PRINT "IN THIS GAME THERE ARE (21) STONES AND WE
    EACH TAKE FROM":"1 TO 3 STONES ALTERNATELY."
90 PRINT
100 PRINT "THE OBJECT IS TO FORCE THE":"OTHER PLAYE
    R TO TAKE THE":"LAST STONE."
110 PRINT
120 PRINT "IT IS REALLY QUITE A SIMPLE GAME."
125 PRINT
130 PRINT "WHY DON'T YOU PRESS THE":"SPACE BAR AND
    WE'LL BEGIN."
140 CALL KEY(0,K,S)
150 IF S=0 THEN 140
160 IF K<>32 THEN 140
170 REM**DIMENSION ARRAYS**
180 DIM TI1(21)
190 DIM TI2(21)
200 DIM YOU1(21)
205 WINS=0
210 TOT=21
220 REM**CLEAR ALL VALUES**         **FROM TEMPORARY
      **          **MEMORY         **
230 FOR I=1 TO 21
240 TI1(I)=0
250 YOU1(I)=0
260 NEXT I
270 CALL CLEAR
280 REM ***************          *   YOUR TURN   *
                 ***************
290 REM
300 PRINT "YOU TAKE"
310 PRINT "(1 TO 3 ONLY)"
320 INPUT YOU1(TOT)
330 IF (YOU1(TOT)<>1)*(YOU1(TOT)<>2)*(YOU1(TOT)<>3)
    THEN 300
335 IF TOT-YOU1(TOT)<1 THEN 300
340 TOT=TOT-YOU1(TOT)
350 PRINT "TOTAL REMAINING ":TOT
360 REM**CHECK FOR WIN**
370 IF TOT=1 THEN 540
380 REM
```

```
390 REM***************          *  TI'S TURN   *
                  ***************
400 REM**CHECK TO SEE    **     **IF A VALUE IS
      **          **STORED IN THE    **        **PERMA
    MENT MEMORY**
410 IF (TI2(TOT)>0)THEN 470
420 REM**IF NO VALUE TAKE**     **RANDOM VALUE
      **
430 TI1(TOT)=INT(RND*3)+1
440 IF (TOT-TI1(TOT))>=1 THEN 480
450 GOTO 430
460 REM**TRANSFER VALUE  **     **FROM PERMAMENT
      **          **MEMORY TO THE    **        **TEMPO
    RARY MEMORY**
470 TI1(TOT)=TI2(TOT)
480 X=TI1(TOT)
490 TOT=TOT-TI1(TOT)
500 PRINT "I TAKE ";X:"TOTAL REMAINING ";TOT
510 REM**CHECK FOR WIN**
520 IF TOT=1 THEN 660
530 GOTO 300
540 PRINT " YOU WIN ARGHH!"
550 REM**TI FORGETS**           **OWN VALUES**
560 FOR I=1 TO 21
570 IF TI1(I)=0 THEN 590
580 TI2(I)=0
590 NEXT I
600 REM**TI REMEMBERS**         **YOUR VALUES **
610 FOR I=1 TO 20
620 IF YOU1(I)=0 THEN 640
630 TI2(I)=YOU1(I)
640 NEXT I
650 GOTO 210
660 PRINT " I WIN!!!"
665 FOR X=1 TO 30
666 NEXT X
670 WINS=WINS+1
680 IF WINS=10 THEN 750
690 REM**TRANSFER    **         **TEMPORARY TO**
                  **PERMAMENT   **
700 FOR I=1 TO 21
710 IF TI1(I)=0 THEN 730
720 PI2(I)=TI1(I)
730 NEXT I
740 GOTO 210
750 CALL CLEAR
760 PRINT "WOULD YOU LIKE TO PLAY SOME MORE?"
770 PRINT
780 PRINT "PRESS THE SPACE BAR TO":"CONTINUE"
```

```
790 PRINT
800 PRINT "PRESS ANY OTHER KEY TO EXIT"
810 CALL KEY(0,K,S)
820 IF S=0 THEN 810
830 IF K=32 THEN 205
840 H$="MAYBE NEXT TIME"
850 CALL CLEAR
860 FOR L=1 TO LEN(H$)
870 CALL HCHAR(12,7+L,ASC(SEG$(H$,L,1)))
880 NEXT L
890 END
```

2

Fun and Games

Labyrinth

Rick Rothstein

After you play this game awhile, you may think you're in Sartre's play No Exit. *Be calm and methodical—there is an escape. All you have to do is find it, and find it fast. For either TI BASIC or Extended BASIC.*

"Labyrinth" is a direct descendant of an award-winning program first written by Robert Tusk for the Apple. Later versions for a number of other personal computers, including the Commodore 64, VIC-20, and Atari, ran in the September 1983 issue of *COMPUTE!* magazine. With a bit of persistence, I've duplicated this game for the TI-99/4A.

Where's the Door

You find yourself somewhere in the midst of a labyrinth, a five-story maze. Each floor has 25 rooms, arranged in a 5 X 5 grid. Within these rooms are openings, sometimes one, other times several. You can move from room to room, heading north, east, south, west, up, or down by pressing the N, E, S, W, U, or D key, respectively.

Only one door within these 125 rooms opens to the outside. It could be an opening in a wall, a floor, or even a ceiling. You need to get out as fast as possible. You never know what may lurk in a labyrinth.

The four compass directions are displayed at the bottom of the screen—the direction you're currently facing is underlined. You can change your orientation by pressing the < key (less than symbol) or the > key (greater than symbol) in order to turn counterclockwise or clockwise, respectively. (Don't press the SHIFT key along with the < or > key.)

Help

For a break from your escape, press AID (FCTN-7), and the game timer freezes, the elapsed time displays, and a menu appears. The six options you'll see offer a number of alternatives.

1. Abandon the current game and start a new maze.
2. Abandon the current game and restart the same maze over again.

3. Return to the current game at exactly the same point where it was interrupted.
4. View the coordinates of the room that you are currently located in.
5. View the coordinates of the room containing the exit.
0. Quit the program.

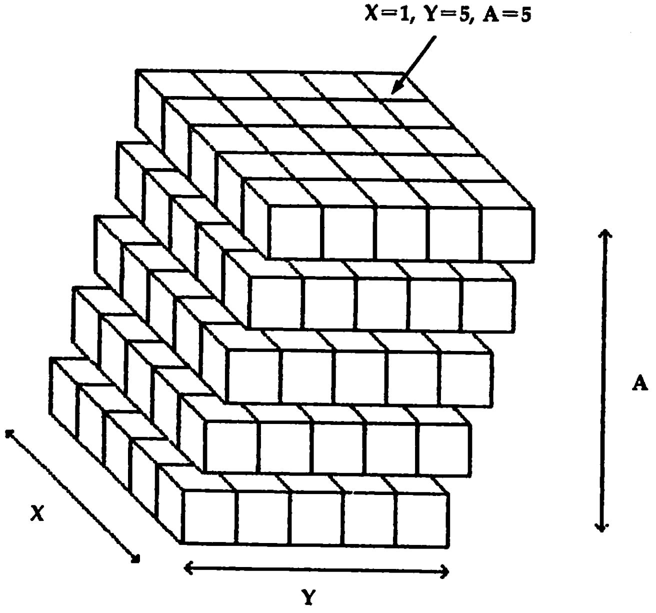
Although options 4 and 5 are provided, save them for only the most desperate situation. Obviously, they make it a snap to escape.

The coordinates provided by options 4 and 5 are in an X,Y,A format. You might see something like

X=1 Y=5 A=5

which means that this particular room is on the topmost level, at the upper-right corner (if you were looking at the floor from above).

The Labyrinth



If you successfully find your way out, you'll be congratulated, advised of the time it took to complete the maze, and offered the option menu. You can play again from there.

A Few Changes

As listed, Labyrinth runs in TI BASIC, the built-in language of the TI-99/4A. If you have Extended BASIC, you need to make the following changes in order for the timer to work correctly:

With Standard 16K Console

```
1090 QA=.0515
1100 QB=.1728
1110 QC=.8748
```

With 32K Memory Expansion

```
1090 QA=.0497
1100 QB=.1627
1110 QC=.8346
```

If you have the original, Version 100 Extended BASIC module, use the TI BASIC version instead—it runs faster for this particular program.

Labryinth

```
940 REM  LABYRINTH
950 REM  (FOR THE TI-99/4A)
990 REM
1000 CALL SCREEN(6)
1010 CALL CLEAR
1020 RANDOMIZE
1030 OPTION BASE 1
1040 DIM W(5,25)
1050 FOR N=1 TO 8
1060 CALL COLOR(N,16,1)
1070 NEXT N
1080 K=49
1090 QA=.0721
1100 QB=.1741
1110 QC=1.6115
1120 FOR N=33 TO 48
1130 READ C$
1140 CALL CHAR(N,C$)
1150 NEXT N
1160 FOR N=94 TO 142
1170 READ C$
1180 CALL CHAR(N,C$)
1190 NEXT N
1200 FOR N=1 TO 5
1210 A$=A$&CHR$(136)
1220 B$=B$&CHR$(129)
1230 NEXT N
1240 CALL CLEAR
```

```
1250 IF K<>49 THEN 1270
1260 PRINT " <SETTING UP ^ PLEASE WAIT>": : : : : :
     : : : : :
1270 FOR N=9 TO 14
1280 CALL COLOR(N,1,1)
1290 NEXT N
1300 IF K=50 THEN 1840
1310 IF K=51 THEN 1880
1320 FOR F=1 TO 5
1330 FOR R=1 TO 25
1340 W(F,R)=0
1350 NEXT R
1360 NEXT F
1370 FOR F=1 TO 5
1380 FOR R=1 TO 16 STEP 5
1390 FOR P=0 TO 4
1400 X=-(RND<.8)
1410 W(F,R+P)=W(F,R+P)+100000*X
1420 W(F,R+P+5)=W(F,R+P+5)+1000*X
1430 NEXT P
1440 NEXT R
1450 NEXT F
1460 FOR F=1 TO 5
1470 FOR R=21 TO 24
1480 FOR P=0 TO -20 STEP -5
1490 X=-(RND<.8)
1500 W(F,R+P)=W(F,R+P)+10000*X
1510 W(F,R+P+1)=W(F,R+P+1)+100*X
1520 NEXT P
1530 NEXT R
1540 NEXT F
1550 FOR F=1 TO 4
1560 FOR R=1 TO 21 STEP 5
1570 FOR P=0 TO 4
1580 X=-(RND<.8)
1590 W(F,R+P)=W(F,R+P)+X
1600 W(F+1,R+P)=W(F+1,R+P)+10*X
1610 NEXT P
1620 NEXT R
1630 NEXT F
1640 XD=INT(6*RND)+1
1650 IF XD<5 THEN 1690
1660 XF=1-4*(XD=6)
1670 XR=INT(25*RND)+1
1680 GOTO 1800
1690 XF=INT(5*RND)+1
1700 P=INT(5*RND)+1
1710 IF XD<>1 THEN 1730
1720 XR=20+P
1730 IF XD<>2 THEN 1750
1740 XR=5*P
```

```
1750 IF XD<>3 THEN 1770
1760 XR=P
1770 IF XD<>4 THEN 1790
1780 XR=5*P-4
1790 IF W(XF,XR)=0 THEN 1640
1800 SF=INT(5*RND)+1
1810 SR=INT(25*RND)+1
1820 IF W(SF,SR)=0 THEN 1800
1830 SD=INT(4*RND)+1
1840 F=SF
1850 R=SR
1860 D=SD
1870 TM=1
1880 W(XF,XR)=W(XF,XR)+10^(6-XD)
1890 CALL CLEAR
1900 IF K<>49 THEN 1960
1910 CALL SOUND(100,700,0)
1920 PRINT "  <PRESS ANY KEY TO BEGIN>": : : : : :
     : : : : :
1930 CALL KEY(0,K,S)
1940 IF S=0 THEN 1930
1950 CALL CLEAR
1960 PRINT "############################":")*
      "&A$&A$&"      -,"
1970 PRINT " +)*     "&CHR$(137)&CHR$(138)&"      "
     &CHR$(140)&CHR$(141)&"     -,."
1980 PRINT "   +)*    "&CHR$(139)&CHR$(136)&A$&CHR$
     (142)&"   -,.":"     +)*           -,*"
1990 PRINT "       /((((((((((((_":"       $
         %":" `ef   $           %   mnh"
2000 PRINT " ` gef $           % mno h":" `aaac $
       pqrrst   % liiih":" `    ` $   uwrrxv   % h
      h"
2010 FOR N=1 TO 5
2020 PRINT " `   ` $   uu  vv   % h   h"
2030 NEXT N
2040 PRINT " `   ` $   yzrr{|   % h   h":" `bbbd-,(
     ((((((((((()*kjjjh"
2050 PRINT " `  -,.   "&CHR$(128)&CHR$(129)&B$&CHR$
     (130)&"   +)*  h"
2060 PRINT " `-,.    "&CHR$(131)&CHR$(132)&"      "
     &CHR$(133)&CHR$(134)&"    +)*h"
2070 PRINT "-,.      "&B$&B$&"      +)*":"(((((((((
     (((((((((((((((((((":"  NORTH  EAST  SOUTH  WE
     ST"
2080 CALL VCHAR(1,2,33)
2090 CALL VCHAR(2,2,36,20)
2100 CALL VCHAR(22,2,38)
2110 CALL VCHAR(1,31,34)
2120 CALL VCHAR(2,31,37,20)
```

```
2130 CALL VCHAR(22,31,39)
2140 GOTO 2850
2150 CALL KEY(0,K,S)
2160 TM=TM+QA
2170 IF S<1 THEN 2150
2180 IF K=1 THEN 2580
2190 IF K<>78 THEN 2240
2200 IF SEG$(W$,1,1)="0" THEN 2550
2210 IF (XF=F)*(XR=R)*(XD=1)THEN 2580
2220 R=R+5
2230 GOTO 2850
2240 IF K<>83 THEN 2290
2250 IF SEG$(W$,3,1)="0" THEN 2550
2260 IF (XF=F)*(XR=R)*(XD=3)THEN 2580
2270 R=R-5
2280 GOTO 2850
2290 IF K<>87 THEN 2340
2300 IF SEG$(W$,4,1)="0" THEN 2550
2310 IF (XF=F)*(XR=R)*(XD=4)THEN 2580
2320 R=R-1
2330 GOTO 2850
2340 IF K<>69 THEN 2390
2350 IF SEG$(W$,2,1)="0" THEN 2550
2360 IF (XF=F)*(XR=R)*(XD=2)THEN 2580
2370 R=R+1
2380 GOTO 2850
2390 IF K<>85 THEN 2440
2400 IF SEG$(W$,6,1)="0" THEN 2550
2410 IF (XF=F)*(XR=R)*(XD=6)THEN 2580
2420 F=F+1
2430 GOTO 2850
2440 IF K<>68 THEN 2490
2450 IF SEG$(W$,5,1)="0" THEN 2550
2460 IF (XF=F)*(XR=R)*(XD=5)THEN 2580
2470 F=F-1
2480 GOTO 2850
2490 IF K<>44 THEN 2520
2500 D=D-1-4*(D=1)
2510 GOTO 2850
2520 IF K<>46 THEN 2550
2530 D=D+1+4*(D=4)
2540 GOTO 2850
2550 CALL SOUND(200,110,0)
2560 TM=TM+QB
2570 GOTO 2150
2580 W(XF,XR)=W(XF,XR)-10^(6-XD)
2590 CALL CLEAR
2600 T$="ELAPSED TIME:"
2610 IF K=1 THEN 2690
2620 PRINT "   << CONGRATULATIONS >>": :
2630 FOR K=1 TO 3
```

```
2640 FOR N=500 TO 800 STEP 20
2650 CALL SOUND(50,N,0)
2660 NEXT N
2670 NEXT K
2680 T$="YOU SOLVED THE MAZE IN:"
2690 MN=INT(TM/60)
2700 SC=INT(TM-60*MN)
2710 C$=STR$(MN)&" MIN "&STR$(SC)&" SEC"
2720 PRINT TAB((28-LEN(T$))/2);T$: :TAB((28-LEN(C$)
     )/2);C$: : : :
2730 PRINT "PRESS  TO":"^^^^^  ^^": :"  1    PLAY A
      NEW MAZE": :"  2    REPLAY THE SAME MAZE": :
2740 PRINT "  3    RETURN TO MAZE": :"  4    SEE PR
     ESENT LOCATION": :"  5    SEE EXIT LOCATION":
     :"  0    QUIT":
2750 CALL SOUND(100,700,0)
2760 CALL KEY(0,K,S)
2770 WS=RND
2780 IF S=0 THEN 2760
2790 IF (K<48)+(K>53)<0 THEN 2830
2800 IF K=52 THEN 3010
2810 IF K=53 THEN 3070
2820 IF K=48 THEN 3150 ELSE 1240
2830 CALL SOUND(200,110,0)
2840 GOTO 2760
2850 CALL HCHAR(24,1,32,32)
2860 FOR N=9 TO 14
2870 CALL COLOR(N,1,1)
2880 NEXT N
2890 W$="000000"&STR$(W(F,R))
2900 W$=SEG$(W$,LEN(W$)-5,6)
2910 TM=TM+QC
2920 CALL COLOR(9,1-15*(SEG$(W$,D-1-4*(D=1),1)="1")
     ,1)
2930 CALL COLOR(10,1-15*(SEG$(W$,D+1+4*(D=4),1)="1"
     ),1)
2940 CALL COLOR(11,1-15*(SEG$(W$,D,1)="1"),1)
2950 CALL COLOR(12,1-15*(SEG$(W$,D,1)="1"),1)
2960 CALL COLOR(13,1-15*(SEG$(W$,5,1)="1"),1)
2970 CALL COLOR(14,1-15*(SEG$(W$,6,1)="1"),1)
2980 CALL SOUND(100,700,0)
2990 CALL HCHAR(24,5+7*(D-1)+(D>2),94,5+(INT(D/2)=D
     /2))
3000 GOTO 2150
3010 XX=R-20*(R>20)*(R<26)-15*(R>15)*(R<21)-10*(R>1
     0)*(R<16)-5*(R>5)*(R<11)
3020 YY=INT((R-1)/5)+1
3030 ZZ=F
3040 CALL CLEAR
3050 PRINT "   YOU ARE AT ";
```

```
3060 GOTO 3120
3070 XX=XR-20*(XR>20)*(XR<26)-15*(XR>15)*(XR<21)-10
     *(XR>10)*(XR<16)-5*(XR>5)*(XR<11)
3080 YY=INT((XR-1)/5)+1
3090 ZZ=XF
3100 CALL CLEAR
3110 PRINT "EXIT LOCATED AT ";
3120 T$="X="&STR$(XX)&" Y="&STR$(YY)&" A="&STR$(ZZ)
3130 PRINT T$: : : :
3140 GOTO 2730
3150 CALL CLEAR
3160 END
3170 DATA 0000000000000303,000000000000C0C,00000000
     0000FFFF,0303030303030303,C0C0C0C0C0C0C0C,0303
     ,C0C,FFFF
3180 DATA C0F07C1F0701,00000000C0F07C1F,0701,030F3E
     F8E08,00000000030F3EF8,E08,0703030303030303,00
     38444C54644438
3190 DATA 0000FFFF,E0C0C0C0C0C0C0C,0303030303030303
     ,FFFF,000000000000FFFF,FFFF030303030303,030303
     030303FFFF
3200 DATA C0F07C1F0701,00000000C0F07C1F,0701,C0C0C0
     C0C0C0C0C,FFFF,000000000000FFFF,C0C0C0C0C0C0FF
     FF
3210 DATA FFFFC0C0C0C0C0C,00000000030F3EF8,030F3EF8
     E08,E08,3F3F3E373331303,FFFF000080C0E07,FFFF
3220 DATA FFFF00000103070E,FCFC7CECCC8C0C0C,3030303
     03030303,0C0C0C0C0C0C0C0C,3F3F30303030303,FCFC
     0C0C0C0C0C0C
3230 DATA 3030303133373E3C,3F7FE0C08,FCFE070301,0C0
     C0C8CCCEC7C3C,0,0,0,0103070F1F3B73E3,FFFF,80C0
     E0F0F8DCCEC7
3240 DATA 0103070E1C3870E,C383030303030303,C3C1C0C0
     C0C0C0C,80C0E070381C0E07,0,000000000000FFFF
3250 DATA E070381C0E070301,03030303030383C3,E3733B1
     F0F070301,C0C0C0C0C0C0C1C3,070E1C3870E0C08,C7C
     EDCF8F0E0C08
```

Spitfire

Edward F. Roberts, Jr.

The fate of England rests with The Few, the pilots of the Royal Air Force, as they try to drive off the clouds of German bombers. You're one of those Few. "Spitfire" uses Extended BASIC.

The Junkers JU-88 bomber is in your sights, but only briefly. You press the firing button, and your Spitfire's machine guns hammer at the enemy's tail. For a moment it jinks out of the way, but you bank and climb and fire again. The enemy bomber explodes in a fireball of flame.

"Spitfire" is an action game which puts you in the cockpit of a fighter plane during the Battle of Britain. You've managed to get behind an enemy bomber and you're ready to bring it down.

Just Like the First Joystick

Spitfire uses the joystick as though it were the control stick of a real fighter. If the enemy aircraft is up and to the left, just pull back and to the left to center it in your sights. Destroying the bomber gives you 100 points. Keep track of your remaining ammunition and fuel (upper left and right on the screen, respectively) as you chase down one bomber after another.

A score of 3500 points advances you to the next difficulty level. A string of six misses lets the enemy bomb your city and costs you ten rounds of ammunition.

Spitfire includes 12 levels of difficulty. Numeric variables are used to control the random motion and velocity of the enemy bomber. After level 6, the joystick sensitivity is increased, allowing a balance between the speed of your aircraft and that of your enemy. The ease at which enemy bombers are targeted and destroyed also becomes more difficult as you advance to other levels.

These levels allow even a beginning pilot to bring down the occasional aircraft, yet still challenge the best of The Few.

As Fast as Possible

Lines 560–620 contain the core of the program. This loop is kept as short as possible to increase the chance of recognizing the fire command from the joystick and also to provide a quicker response to joystick motion inputs.

The conditional branch in line 560 checks the fire button. If it's been pressed, the program jumps to a routine which checks for coincidence, decrements the ammo remaining, and increments the score if a hit is declared. If a miss is declared, it increments the miss count and checks to see if the skyline must be displayed. The skyline begins to appear with the third miss. A hit prior to the sixth miss clears the miss count and removes the skyline from the bottom of the screen.

Spitfire

```
100 REM  ******************
110 REM  **** SPITFIRE ****
120 REM  ******************
150 REM
160 OPTION BASE 1
170 CALL CLEAR
180 FOR COL=1 TO 14 :: CALL COLOR(COL,16,1):: NEXT
    COL
190 CALL SCREEN(2)
200 DISPLAY AT(10,12):"SPITFIRE"
220 FOR T=1 TO 500 :: NEXT T
230 FOR COL=1 TO 14 :: CALL COLOR(COL,2,1):: NEXT C
    OL
240 CALL CLEAR
250 AMMO=60 :: FUEL=600 :: SCORE=0 :: DIM SCENE$(5)
260 DIM SONG(11,2)
270 FOR X=1 TO 11 :: READ SONG(X,1),SONG(X,2):: NEX
    T X
280 DATA 750,117,750,117,183,117,558,117,750,139,18
    3,131,558,131,183,117,558,117,750,110,1500,117
290 TOL=14 :: VMUL=8 :: HMUL=18 :: N=1 :: JMUL=3
300 CALL SCREEN(8)
330 CALL MAGNIFY(4)
340 CALL CHAR(45,"000000FF")
350 CALL CHAR(136,"914925158B47370FFF070B1525499121
    12244952A4C8D0FFC0E0586651282422")
360 CALL CHAR(104,"02070F1F0F1F3F7FFFFF7F3F1F030100
    0080C0E0F0F8FCFFFEFCF0E0E0E0C080")
370 CALL COLOR(10,11,1)
380 CALL CHAR(140,"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    ")
390 CALL COLOR(14,3,3)
400 CALL HCHAR(24,1,140,32)
410 CALL CHAR(112,"99FF99FF99FF99FF090F090F090F090F
    90F090F090F090F099FF99FFFFFFFFFF")
420 CALL CHAR(116,"FFFFFFFFFFFFFFFF")
```

```
430 CALL CHAR(117,"00000000FFFFFFFFC3C3C3C3C3C3C3C3
    00000000F0F0F0F0000000000F0F0F0F")
440 LET SCENE$(5)="    utttttttttttttttu" :: SCENE$(
    4)="      tttttttttttsp"
450 SCENE$(3)="       vututwtxpppp" :: SCENE$(2)="
          v   t t ppuu"
460 SCENE$(1)="       v   t w uu"
470 CALL CHAR(46,"101010FF10101010")
480 CALL CHAR(36,"00183C3E7FFFFF7F7F3F7FFFFF7F3F0F0
    00000080E0F0F8FDFFFFEFCFCFEFC")
485 CALL SPRITE(#3,36,16,80,45,0,-1):: CALL SPRITE(
    #4,36,16,68,35,0,2)
490 DISPLAY AT(11,13):"--" :: DISPLAY AT(15,13):"--
    "
500 DISPLAY AT(13,10):"-.-  -.-"
510 RANDOMIZE
520 CALL CHAR(40,"00"):: CALL CHAR(41,"00"):: CALL
    CHAR(43,"00")
530 CALL CHAR(96,"01010F010307FF37330000000000000000
    000E00080C0FED8980000000000000")
540 CALL SPRITE(#7,96,5,60,24,-9+(RND*9),8+(RND*9))
550 CALL SPRITE(#2,42,7,91,98)
560 CALL JOYST(1,H,V):: CALL KEY(1,KEY,STATUS):: IF
     KEY=18 THEN 630
570 FUEL=FUEL-1
580 DISPLAY AT(1,20):"FUEL";FUEL
590 IF FUEL<=0 THEN 1160
600 CALL MOTION(#7,(VMUL*(RND-.5)-(JMUL*V)),(HMUL*(
    RND-.5)-(JMUL*H)))
610 CALL MOTION(#4,-(1*V),2-(1*H)):: CALL MOTION(#3
    ,-(1*V),-2-(1*H))
620 GOTO 560
630 CALL COINC(#7,86,105,TOL,COIN)
640 CALL SOUND(100,-5,0)
650 AMMO=AMMO-1
660 MISS=MISS+1
670 IF MISS>=3 THEN 910 ELSE 680
680 IF AMMO<0 THEN AMMO=0
690 DISPLAY AT(1,3):"AMMO";AMMO
700 REM IF AMMO<1 THEN 900
710 IF (COIN=-1)*(KEY=18)THEN 720 ELSE 560
720 CALL SOUND(500,-7,0):: CALL SCREEN(7):: CALL SC
    REEN(9)
730 CALL DELSPRITE(#7)
740 CALL SPRITE(#11,104,11,80,106)
750 CALL SPRITE(#10,136,14,88,95)
760 FOR D=1 TO 50 :: NEXT D :: CALL COLOR(#10,2)::
    CALL COLOR(#11,7)
770 FOR D=1 TO 50 :: NEXT D
780 CALL COLOR(#10,7):: CALL COLOR(#11,16)
```

```
790 CALL SCREEN(7):: FOR D=1 TO 30 :: NEXT D :: CAL
    L SCREEN(8)
800 FOR D=1 TO 20 :: NEXT D
810 FREQ=1000
820 MISS=0
830 CALL HCHAR(19,1,32,160)
840 FOR S=1 TO 8
850 CALL SOUND(-100,FREQ,3):: FREQ=FREQ-100
860 NEXT S
870 SCORE=SCORE+100
880 DISPLAY AT(2,3):"SCORE";SCORE
890 CALL DELSPRITE(#7,#10,#11):: IF AMMO<1 THEN 900
     ELSE 540
900 IF SCORE>(N*3500)THEN 1080 ELSE 1210
910 REM  SHOW CITY
920 IF MISS<>3 THEN 940
930 DISPLAY AT(23,1):SCENE$(1):: GOTO 680
940 IF MISS=4 THEN 980
950 IF MISS=5 THEN 990
960 IF MISS=6 THEN 1000
970 IF MISS=7 THEN 1020
980 DISPLAY AT(22,1):SCENE$(1):: DISPLAY AT(23,1):S
    CENE$(2):: GOTO 680
990 DISPLAY AT(21,1):SCENE$(1):: DISPLAY AT(22,1):S
    CENE$(2):: DISPLAY AT(23,1):SCENE$(3):: GOTO 68
    0
1000 DISPLAY AT(20,1):SCENE$(1):: DISPLAY AT(21,1):
     SCENE$(2):: DISPLAY AT(22,1):SCENE$(3)
1010 DISPLAY AT(23,1):SCENE$(4):: GOTO 680
1020 DISPLAY AT(19,1):SCENE$(1):: DISPLAY AT(20,1):
     SCENE$(2):: DISPLAY AT(21,1):SCENE$(3)
1030 CALL SPRITE(#9,136,7,160,100):: CALL SPRITE(#8
     ,104,11,158,90)
1040 CALL SOUND(1500,-7,0)
1050 DISPLAY AT(22,1):SCENE$(4):: DISPLAY AT(23,1):
     SCENE$(5):: MISS=0 :: AMMO=AMMO-9
1060 FOR D=1 TO 50 :: NEXT D :: CALL DELSPRITE(#8,#
     9):: CALL HCHAR(19,1,32,160):: GOTO 680
1070 END
1080 AMMO=60 :: FUEL=600 :: MISS=0 :: N=N+1 :: READ
      TOL :: READ VMUL :: READ HMUL
1090 DISPLAY AT(24,21):"LEVEL";N
1100 IF N>6 THEN JMUL=JMUL+1
1110 CALL HCHAR(19,1,32,160)
1120 IF N=13 THEN 1210
1130 GOTO 560
1140 DATA 12,9,20,11,10,22,10,13,26,9,16,30,9,18,32
     ,9,20
1150 DATA 36,9,21,37,9,22,38,9,24,39,10,26,41,10,28
     ,42,12,30,43,13,40,50
```

```
1160 DISPLAY AT(8,6):"YOUR FUEL IS GONE"
1170 DISPLAY AT(9,6):"YOU HAVE CRASHED"
1180 CALL SOUND(2000,-7,0)
1190 FOR NOTE=1 TO 11 :: CALL SOUND(SONG(NOTE,1),SO
     NG(NOTE,2),0)
1200 NEXT NOTE
1210 DISPLAY AT(12,10):"GAME OVER"
1220 CALL MOTION(#7,10*RND,10*RND):: DISPLAY AT(1,2
     0):"FUEL"
1230 DISPLAY AT(10,5):"WANT TO PLAY AGAIN? Y/N"
1240 ACCEPT AT(11,16):A$ :: RESTORE :: IF ((ASC(A$)
     =89)+(ASC(A$)=121))THEN 240
1250 CALL CLEAR
1260 END
```

BOG'L

Rick Rothstein

Duplicating the popular word game, "BOG'L" displays letters and provides a three-minute timer on the screen as you list all the words you can. Extended BASIC and Memory Expansion necessary.

Did you buy your computer so that your children would have access to a personal teaching tool? So that you could operate your business more profitably? If you're anything like me (and most people), one reason you bought it was to play games.

Word games have always held a particular fascination for me, and so it seemed logical to let my TI-99/4A help satisfy this passion. One of my all-time favorites is a word-find game made by Parker Brothers under the name Boggle.

First, the Rules

The mechanical version of Boggle is played with 16 letter cubes, a shaker tray, and an egg timer. The letter cubes are placed into the shaker tray, mixed thoroughly, and allowed to come to rest in a 4 × 4 grid. The timer is started, and the top faces of the letter cubes are revealed by removing the cover from the shaker tray.

Each player (two to six recommended) then takes paper and pencil and writes down as many words as he or she can find, within the allotted time, according to the following rules:

- All words must contain three or more letters.
- Each word is formed by linking adjoining letters together in their proper sequence.
- The letters may be joined horizontally, vertically, diagonally, or a by combination of those directions.
- No letter cube may be used more than once within any single word.
- Any word—including plurals, forms, and tenses—may be used as long as it can be found in a standard English dictionary.
- Proper names, abbreviations, and words spelled with an apostrophe or hyphen are not allowed.
- Words within words are permissible. For example, SCARE, CARE, SCAR, CAR, and ARE would all be counted.

- When the timer runs out, all players *must* stop writing.

Playing BOG'L

"BOG'L" is written in Extended BASIC and simulates the play of the Parker Brothers game perfectly. Even the faces of the letter "cubes" are randomly displayed rightside up, upside down, or sideways.

First, type in, save, and then load and run the program. If you have a disk system with no Memory Expansion unit attached, you must type and enter CALL FILES(1), followed by NEW *before* loading this program. The program will then ask whether you want to see a display of the alphabet used in the game. Pressing Y displays all 26 letters of the alphabet exactly as they appear during play. If you're playing BOG'L for the first time, take a look at the alphabet to become accustomed to the shapes. This is a good idea since BOG'l often displays letters upside down or on their sides. (Special notice should be taken of the *W* and *M*, the *U* and *V*, the *N* and Z—note the underline—as well as the letter *Q*, which is always displayed together with the letter *U*.)

If you press any key after viewing the alphabet, or if you pressed N instead of viewing the alphabet, the program begins setting up the first game board. When the onscreen timer, set for three minutes, appears to the right of the letter tray, play may be started by pressing any key. Four beeps will sound, giving you time to pick up a pencil and paper, and then a 4 × 4 grid of letters appears. Game play proceeds according to the rules listed above. When three minutes have elapsed, an end-of-game tone is sounded and all letters are blacked out.

Pressing any key redisplays the game board. Use this feature for scoring. At the same time, the program begins to calculate the next game board. Hence, redisplaying the game board *before* scoring begins will minimize the wait required to start the next game. Pressing FCTN-8 (REDO) anytime after the board is redisplayed blacks out the tray of letters, resets the timer (invisibly), and continues setting up the next game board. Pressing FCTN-3 (ERASE) ends the program.

During game play, while the timer is counting down, you may do any of the following:

- Press FCTN-8 (REDO) to restart another game. This feature is included in case no usable vowels are displayed.

- Press FCTN-7 (AID) to suspend play, in other words, to freeze the timer and black out the letter board. This feature is included in case a neighbor, or nature, calls. Press FCTN-7 a second time to restart the game at the exact point where it was interrupted.
- Press FCTN-3 (ERASE) to end the program.

How to Score

Scoring in BOG'L follows these rules:

- At the end of a round, each player in turn reads his or her list of words out loud.
- If a word appears on more than one player's list, it's crossed off all the lists on which it appears—*including the reader's list.*
- After all players have read from their lists, one point is scored for each word remaining on a player's list.
- All words of five or more letters receive a bonus:

Number of Letters	**Bonus Points**
5	1
6	2
7	4
8+	10

- The player having the highest number of points wins.

In the Program

There are two versions of Extended BASIC. Version 100 is the older and slower of the two. In particular, it processes FOR-NEXT loops at about *half* the speed of Version 110. Since the onscreen timer operates by means of three nested FOR-NEXT loops, it's necessary to take this speed difference into account so that BOG'L will work correctly for either version. Line 150 accomplishes this.

The first statement asks the Extended BASIC module which version it is. The second statement sets the variable named TIMER to 12 for Version 100 or to 22 for Version 110. (Remember, the relational expression in the parentheses evaluates to minus one if I=100, and zero if I<>100.)

As it turns out, there's another speed of operation difference which must be accounted for. Version 110 processes FOR-NEXT loops just a little bit faster if a 32K Memory Expansion unit is attached to your computer than if no memory expansion is hooked up. (Version 100 seems not to be affected by this.) The last three statements in line 150 adjust the TIMER variable if necessary.

The ON ERROR 830 statement sets a path for the program to follow should an error occur. The CALL INIT statement checks to see if the 32K Memory Expansion is attached. If it is, error trapping is reset to normal and the program continues. However, if the memory unit is not attached, an error occurs and program execution is sent to line 830. If Version 110 of Extended BASIC is in operation, the TIMER variable is reduced by one. If Version 100 is in operation, the TIMER variable is not changed. The program then returns to the last statement of line 150, at which point error trapping is reset to normal before continuing.

Describing the Program

Lines	Function
150–160	Initialize program.
170–180	Ask players whether they want to see the alphabet and check for answer.
190–230	Display entire alphabet if requested and check for program continuation.
240	Additional program initializing.
250–260	Define the graphics to be used on the timer.
270–280	Define the graphics to be used on the letter tray.
290–310	Display the letter tray (invisibly).
320–340	Display the onscreen timer (invisibly).
350	Places the sprites which will be used to display the letter cubes onto the screen.
360	Randomly picks one letter to be displayed on the face of each of the 16 letter cubes.
370–660	For each letter cube, select the appropriate DATA statement, randomly pick one of the four possible orientations for the letter to be displayed, and assign the character pattern to the FACE$( ) array. Also check to see if the timer is visible or not. The timer is invisible only if FCTN-8 (REDO) has already been pressed.
670–680	Wait for FCTN-8 (REDO) or, alternatively, FCTN-3 (ERASE) to be pressed. These lines are reached only if the timer was visible when line 660 was executed.
690	Assigns the appropriate cube face patterns to the characters which define each sprite.
700	Sounds an alert and then makes the onscreen timer visible.

710	Checks for the starting of a game or, alternatively, the quitting of the program.
720	Beeps four times and then makes the game board visible.
730–750	Run the onscreen timer.
760	Signals the end of a game and blacks out the screen.
770	Checks to see if the players want to redisplay the game board for verification purposes during scoring. If yes, the board is redisplayed and the program branches to line 360, at which point calculations for the next game board are started.
780–790	This subroutine checks to see if FCTN-8 (REDO) is pressed while the timer is visible. If so, the tray is blacked out and the timer is made invisible.
800–810	Redisplay the game board if FCTN-7 (AID) is pressed and start the timer again upon returning. This subroutine is activated only if AID was pressed in order to freeze the timer during the playing of a game.
820	This subroutine makes the timer invisible and blacks out the tray.
830	Adjusts the variable named TIMER if required.
840–1350	Contain the character string expressions which specify the shapes of the letters. The first four items of data contain the patterns for the four possible orientations of the letter *A*, the next four for the letter *B*, and so on.
1360–1370	Each item of data contains the six letters which make up each letter cube.
1380	Clears the screen and quits the program.

BOG'L

```
150 CALL VERSION(I):: TIMER=22+10*(I=100):: ON ERRO
    R 830 :: CALL INIT :: ON ERROR STOP
160 CALL SCREEN(4):: CALL MAGNIFY(4):: DIM FACE$(16
    ),LETTER$(16):: DEF F$(N)=RPT$("F",N)
170 CALL CLEAR :: DISPLAY "DO YOU WANT TO SEE THE"
    :: DISPLAY BEEP:"ALPHABET DISPLAYED? (Y/N)"
180 CALL KEY(0,K,S):: IF S=0 THEN 180 ELSE IF K=78
    THEN 240 ELSE IF K<>89 THEN 180
190 N=0 :: CALL CLEAR :: CALL CHAR(33,"000000000000
    0303",35,"000000000000FCFC")
200 FOR ROW=-3 TO 166 STEP 28 :: FOR COL=65 TO 161
    STEP 32 :: N=N+1 :: READ T$,A$,A$,A$ :: CALL CH
    AR(32+4*N,T$)
210 CALL SPRITE(#N,32+4*N,2,ROW-256*(ROW=-3),COL)::
     IF N=17 THEN DISPLAY AT(14,7):"!#" ELSE IF ROW
    =165 AND COL=97 THEN 230
```

```
220 NEXT COL :: NEXT ROW
230 CALL KEY(0,K,S):: IF S=0 THEN 230 ELSE CALL DEL
    SPRITE(ALL)
240 VISIBLE=0 :: CALL CLEAR :: CALL CHARSET :: CALL
     COLOR(2,1,1,3,1,1,4,1,1,5,5,2,6,5,2)
250 CALL CHAR(52,RPT$("01",8)&RPT$("0",14)&"FC"&RPT
    $("0",14)&"E00"):: N=55
260 FOR I=14 TO 0 STEP -2 :: N=N+1 :: CALL CHAR(N,R
    PT$("0",I)&F$(16-I)):: NEXT I
270 CALL CHAR(64,F$(13)&"CFC"&F$(12)&"0000"&F$(12)&
    "3C3C"&F$(12)&"3F3F",68,"FCFC"&F$(12)&"0000"&F$
    (12)&"3C3C"&F$(12)&"3F3"&F$(13))
280 CALL CHAR(72,RPT$("FC",10)&F$(9)&"CFC"&"3F3"&F$
    (9)&RPT$("3F",10),76,"3C3C"&F$(8)&"3C3C0",78,"0
    000"&F$(8),79,RPT$("3C",8))
290 DISPLAY AT(3,1):"@AAAABAAAABAAAABAAAAC"
300 FOR I=4 TO 19 STEP 5 :: DISPLAY AT(I,1):RPT$("H
    MMMMOMMMMOMMMMOMMMMK+++++++",4)&"INNNNLNNNNLNNN
    NLNNNNJ" :: NEXT I
310 DISPLAY AT(23,1):"DEEEEFEEEEFEEEEFEEEEG"
320 GOSUB 820 :: CALL HCHAR(3,26,55,5):: CALL HCHAR
    (4,26,55,5):: CALL HCHAR(22,26,55,5):: CALL HCH
    AR(23,26,55,5)
330 CALL VCHAR(4,26,52,18):: CALL VCHAR(4,27,55,19)
    :: CALL VCHAR(4,28,63,18):: CALL VCHAR(4,29,63,
    18):: CALL VCHAR(4,30,55,19)
340 FOR I=0 TO 3 :: CALL HCHAR(6*I+3,27,53):: CALL
    HCHAR(6*I+4,27,51-I):: CALL HCHAR(6*I+6,27,54):
    : NEXT I :: CALL HCHAR(24,27,32)
350 RANDOMIZE :: N=0 :: FOR J=25 TO 145 STEP 40 ::
    FOR I=25 TO 145 STEP 40 :: N=N+1 :: CALL SPRITE
    (#N,76+4*N,2,J,I):: NEXT I :: NEXT J
360 RESTORE 1360 :: FOR I=1 TO 16 :: GOSUB 780 :: R
    EAD T$ :: LETTER$(I)=SEG$(T$,INT(RND*6+1),1)::
    NEXT I
370 FOR I=1 TO 16 :: GOSUB 780
380 IF LETTER$(I)<"N" THEN ON ASC(LETTER$(I))-64 GO
    TO 400,410,420,430,440,450,460,470,480,490,500,
    510,520
390 ON ASC(LETTER$(I))-77 GOTO 530,540,550,560,530,
    580,590,600,610(620(630,200,650
400 RESTORE 840 :: GOTO 660
410 RESTORE 860 :: GOTO 660
420 RESTORE 880 :: GOTO 660
430 RESTORE 900 :: GOTO 660
440 RESTORE 920 :: GOTO 660
450 RESTORE 940 :: GOTO 660
460 RESTORE 960 :: GOTO 660
470 RESTORE 980 :: GOTO 660
480 RESTORE 1000 :: GOTO 660
```

```
490 RESTORE 1020 :: GOTO 660
500 RESTORE 1040 :: GOTO 660
510 RESTORE 1060 :: GOTO 660
520 RESTORE 1080 :: GOTO 660
530 RESTORE 1100 :: GOTO 660
540 RESTORE 1120 :: GOTO 660
550 RESTORE 1140 :: GOTO 660
560 RESTORE 1160 :: GOTO 660
570 RESTORE 1180 :: GOTO 660
580 RESTORE 1200 :: GOTO 660
590 RESTORE 1220 :: GOTO 660
600 RESTORE 1240 :: GOTO 660
610 RESTORE 1260 :: GOTO 660
620 RESTORE 1280 :: GOTO 660
630 RESTORE 1300 :: GOTO 660
640 RESTORE 1320 :: GOTO 660
650 RESTORE 1340
660 GOSUB 780 :: FOR J=0 TO INT(RND*4):: READ FACE$
    (I):: NEXT J :: NEXT I :: IF VISIBLE=0 THEN 690
670 WASTER=RND :: CALL KEY(0,K,S):: IF S=0 THEN 670
680 IF K=7 THEN 1380 ELSE IF K=6 THEN GOSUB 820 ::
    GOTO 690 ELSE 670
690 FOR I=16 TO 1 STEP -1 :: N=INT(I*RND+1):: CALL
    CHAR(76+4*I,FACE$(N)):: FACE$(N)=FACE$(I):: NEX
    T I
700 CALL SOUND(150,800,0):: CALL COLOR(3,2,16,4,7,1
    6):: VISIBLE=1
710 WASTER=RND :: CALL KEY(0,K,S):: IF S=0 THEN 710
     ELSE IF K=7 THEN 1380
720 FOR I=1 TO 4 :: CALL SOUND(200,400,0):: CALL SO
    UND(800,400,30):: NEXT I :: CALL COLOR(5,5,15,6
    ,5,15)
730 FOR J=4 TO 21 :: FOR I=62 TO 55 STEP -1 :: FOR
    H=1 TO TIMER :: T$=""
740 CALL KEY(0,K,S):: IF K=6 THEN GOSUB 820 :: GOTO
     360 ELSE IF K=7 THEN 1380 ELSE IF K=1 AND S=1
    THEN CALL COLOR(5,5,2,6,5,2):: GOSUB 800
750 NEXT H :: CALL HCHAR(J,28,I,2):: NEXT I :: NEXT
     J
760 CALL SOUND(400,110,9,220,9,330,9):: CALL COLOR(
    5,5,2,6,5,2)
770 WASTER=RND :: CALL KEY(0,K,S):: IF S=0 THEN 770
     ELSE IF K=7 THEN 1380 ELSE CALL COLOR(5,5,15,6
    ,5,15):: GOTO 360
780 CALL KEY(0,K,S):: IF K=7 THEN 1380 ELSE IF VISI
    BLE=1 AND K=6 THEN GOSUB 820
790 RETURN
800 CALL KEY(0,K,S):: IF S=0 THEN 800
810 IF K=1 AND S=1 THEN CALL COLOR(5,5,15,6,5,15)::
     RETURN ELSE IF K=7 THEN 1380 ELSE 800
```

```
820 VISIBLE=0 :: CALL COLOR(3,1,1,4,1,1,5,5,2,6,5,2
    ):: CALL VCHAR(4,28,63,18):: CALL VCHAR(4,29,63
    ,18):: RETURN
830 TIMER=TIMER-I-(I=100):: RETURN NEXT
840 DATA 00000103070E1C181F1F181818180000000080C0E0
    703818F8F818181818,000000003F3F03030303033F3F00
    0000000000C0E070381C1C3870E0C
850 DATA 0000181818181F1F181C0E07030100000000181818
    18F8F8183870E0C08,00000003070E1C38381C0E0703000
    00000000000FCFCC0C0C0C0C0FCFC
860 DATA 00001F1F1818181F1F1818181F1F00000000E0F038
    1838F0F0381838F0E,0000003F3F31313131313B1F0E000
    00000000000FCFC8C8C8C8CDCF87
870 DATA 0000070F1C181C0F0F1C181C0F0700000000F8F818
    1818F8F8181818F8F8,0000000E1F3B31313131313F3F00
    00000000000070F8DC8C8C8C8CFCFC
880 DATA 0000070F1C18181818181C0F070000000000F0F818
    0000000000000018F8F,0000000F1F383030303030381800
    00000000000F0F81C0C0C0C0C1C18
890 DATA 00000F1F18000000000000181F0F00000000E0F038
    18181818181838F0E,00000018383030303030381F0F000
    00000000000181C0C0C0C0C1CF8F
900 DATA 00001F1F18181818181818181F1F00000000C0E070
    38181818183870E0C,0000003F3F30303030381C0F07000
    00000000000FCFC0C0C0C1C38F0E
910 DATA 000003070E1C181818181C0E070300000000F8F818
    18181818181818F8F8,000000070F1C38303030303F3F00
    000000000000E0F0381C0C0C0CFCFC
920 DATA 00001F1F1818181F1F1818181F1F00000000F8F800
    0000E0E0000000F8F8,0000003F3F313131313131303000
    000000000000FCFC8C8C8C8C8C0C0C
930 DATA 00001F1F00000007070000001F1F00000000F8F818
    1818F8F8181818F8F8,00000030303131313131313F3F00
    0000000000000C0C8C8C8C8C8CFCFC
940 DATA 00001F1F1818181F1F181818181800000000F8F800
    0000E0E,0000003F3F0101010101000000000000000000F
    CFC8C8C8C8C8C0C0C
950 DATA 00000000000000070700000001F1F00000000181818
    1818F8F8181818F8F8,00000030303131313131313F3F00
    00000000000000008080808080FCFC
960 DATA 0000070F1C18181818181C0F070000000000F0F818
    00000078781818F8F8,0000000F1F3830303033333F3F00
    000000000000F0F81C0C0C0C0C1C18
970 DATA 00001F1F18181E1E000000181F0F00000000E0F038
    18181818181838F0E,00000018383030303030381F0F000
    0000000000FCFCCCC0C0C0C1CF8F
980 DATA 0000181818181F1F18181818180000000000181818
    1818F8F81818181818,000000003F3F01010101013F3F00
    000000000000FCFC8080808080FCFC
```

```
990 DATA 000018181818181F1F18181818180000000000181818
    1818F8F81818181818,0000003F3F01010101013F3F00
    00000000000FCFC8080808080FCFC
1000 DATA 0000070701010101010101070700000000E0E08
     080808080808080E0E,00000000030303F3F303000000
     00000000000000000C0CFCFC0C0C
1010 DATA 0000070701010101010101070700000000E0E08
     080808080808080E0E,00000000030303F3F303000000
     00000000000000000C0CFCFC0C0C
1020 DATA 0000000000000000000000181C0F07000000000018181
     818181818181838F0E,0000000C1C383030303030381F0F0
     00000000000000000000000000000FCFC
1030 DATA 0000070F1C18181818181818180000000000E0F03
     81800000,0000003F3F0000000000000000000000000000
     0F0F81C0C0C0C0C1C383
1040 DATA 000018181819181F1F1F191818180000000000003870E
     0C080000080C0E07038,0000003F3F0303070E1C383020
     0000000000000FCFC80C0E070381C0C04
1050 DATA 00001C0E070301000001030 70E1C00000000018181
     898F8F8F8D898181818,0000002030381C0E0703013F3F
     0000000000000040C1C3870E0C0C0FCFC
1060 DATA 000018181818181818181818181F1F0000000000000
     000000000000000F8F8,0000003F3F303030303030303030
     0000000000000FCFC0000000
1070 DATA 00001F1F0000000000000000000000000000F8F81
     818181818181818181818,00000000000000000000003F3F
     0000000000000C0C0C0C0C0C0CFCFC
1080 DATA 0000181C1E1F1B191818181818180000000000018387
     8F8D89818181818181818,0000003F3F00000000000003F3F
     0000000000000FCFC3870E0E07038FCFC
1090 DATA 00001818181818181 91B1F1E1C18000000000018181
     818181898D8F873818,0000003F3F1C0E07070E1C3F3F
     0000000000000FCFC00000000000000FCFC
1100 DATA 0000181C1E1F1B19191818181818000000000018181
     818189898D8F8783818,0000003F3F00001070E1C3F3F
     0000000000000FCFC3870E0800000FCFC
1110 DATA 0000181C1E1F1B19191818181818000000000018181
     818189898D8F8783818,0000003F3F00001070E1C3F3F
     0000000000000FCFC3870E0800000FCFC
1120 DATA 0000070F1C18181818181C0F0700000000E0F03
     818181818181838F0E,0000000F1F383030303030381F0F0
     00000000000F0F81C0C0C0C1CF8F
1130 DATA 0000070F1C18181818181C0F0700000000E0F03
     818181818181838F0E,0000000F1F383030303030381F0F0
     00000000000F0F81C0C0C0C1CF8F
1140 DATA 00001F1F181818 1F1F1818181818000000000E0F03
     81838F0E,0000003F3F01010101010000000000000000
     0FCFC8C8C8C8CDCF87
1150 DATA 00000000000000000070F1C181C0F07000000000018181
     81818F8F8181818F8F8,0000000E1F3B31313131313F3F
     0000000000000000008080808080FCFC
```

```
1160 DATA 001E3F73616167733F1E000000000000000000808
     08080C040006666667E3C,0000010303030301003D7C60
     607C3C0000F8FC8E0646CEFCF88
1170 DATA 003C7E66666600020301010101000000000000000
     00078FCCEE68686CEFC78,000000000000011F3F736260
     713F1F00003C3E06063EBC0080C0C0C08
1180 DATA 00001F1F1818181F1F1B1918181800000000E0F03
     81838F0E080C0E07038,0000003F3F0103070F1D393020
     000000000000FCFC8C8C8C8C8CDCF87
1190 DATA 00001C0E070301070F1C181C0F070000000018181
     898D8F8F8181818F8F8,0000000E1F3B31313131313F3F
     000000000000040C9CB8F0E0C080FCFC
1200 DATA 0000070F1C181C0F0700181C0F0700000000E0F03
     81800E0F0381838F0E,0000000C1C39313131313B1F0E0
     000000000000070F8DC8C8C8C9C383
1210 DATA 0000070F1C181C0F0700181C0F0700000000E0F03
     81800E0F0381838F0E,0000000C1C39313131313B1F0E0
     000000000000070F8DC8C8C8C9C383
1220 DATA 00001F1F010101010101010101010000000000F8F88
     080808080808080808,000000000000003F3F000000000
     000000000000C0C0C0CFCFC0C0C0C0C
1230 DATA 000001010101010101010101011F1F0000000080808
     080808080808080F8F8,000000303030303F3F30303030
     00000000000000000000FCFC
1240 DATA 00001818181818181818181C0F07000000000018181
     818181818181838F0E,0000000F1F3830303030381F0F0
     00000000000FCFC00000000000FCFC
1250 DATA 0000070F1C181818181818181818000000000E0F03
     8181818181818181818,0000003F3F0000000000003F3F
     0000000000000F0F81C0C0C0C1CF8F
1260 DATA 000018181818181818181C0E070301000000018181
     8181818183870E0C08,0000000003070E1C38381C0E07030
     00000000000FCFC00000000000FCFC
1270 DATA 00000103070E1C18181818181818000000000080C0E
     0703818181818181818,0000003F3F0000000000003F3F
     000000000000C0E070381C1C3870E0C
1280 DATA 0000181818181919191B1F0F06000000000018181
     81898989898D8F8F06,0000000F1F383C1F1F3C381F0F0
     00000000000FCFC0000C0C00000FCFC
1290 DATA 0000060F1F1B1919191818181818000000000060F0F
     8D89898981818181818,0000003F3F0000030300003F3F
     000000000000F0F81C3CF8F83C1CF8F
1300 DATA 0000181C1C0E070303070E1C1C180000000018383
     870E0C0C0E070383818,000000383C1E070303071E3C38
     0000000000001C3C78E0C0C0E0783C1C
1310 DATA 0000181C1C0E070303070E1C1C180000000018383
     870E0C0C0E070383818,000000383C1E070303071E3C38
     0000000000001C3C78E0C0C0E0783C1C
```

```
1320 DATA 00001818181C0E0703010101010100000000018181
     83870E0C0808080808,00000000000000013F3F010000000
     000000000003C7CE0C08080C0E07C3C
1330 DATA 0000010101010103070E1C1818180000000080808
     08080C0E07038181818,0000003C3E07030101030773E3C
     00000000000000000080FCFC8
1340 DATA 00001F1F0000010307 0E1F1F001F00000000F8F87
     0E0C0800000F8F800F8,0000002C2E2F2F2D2C2C2C2C2C
     0000000000000C0C0C8CCCEC7C3C1C0C
1350 DATA 00001F001F1F00000103070E1F1F00000000F800F
     8F870E0C0800000F8F8,00000030383C3E373331303030
     0000000000003434343434B4F4F47434
1360 DATA RSLCAE,DENOSW,ISEHNP,AMORHS,GVNTEI,TAOCAI
     ,BMJAOQ,YHEFIE
1370 DATA USETLP,EDANVZ,FBRXOI,PECAMD,DTNKOU,GURIWL
     ,ULEKYG,YATILB
1380 CALL DELSPRITE(ALL):: CALL CLEAR :: END
```

Speed Demon

J. C. Hilty

Avoid, then pass the onrushing cars trying to block your path as you press the accelerator to the floorboard. This 3-D racing game is written in Extended BASIC.

Look in any arcade and you'll see several varieties of car-racing videogames. Though certainly not able to display the kind of sophisticated arcadelike graphics you see on these machines, nor with their speed, your TI-99/4A can produce an exciting racing game.

Unsafe at Any Speed

The object of "Speed Demon," as in many racing games, is to pass other race cars while keeping the speedometer needle pressed against the edge of the dial. You move your car from side to side with the left and right arrow keys (just press the keys themselves—you don't need to use the FCTN key as you do when moving the cursor during program line editing). The up and down arrow keys control your speed—pressing the up key increases speed, while the down key applies the brakes. The faster you go, the more points are added to your score. Don't exceed the speed limit—300 mph—of Speed Demon, or you'll crash.

You'll notice that there's a slight delay in your car's response when a new car approaches. Going off the road or, of course, hitting another car, turns your million-dollar racing machine into a pile of smoking metal.

Your final score is a sum of the number of cars you pass and the amount of acceleration.

Speed Demon Details

Speed Demon simulates the 3-D effect of the arcade games. This involves designing a screen which gives a feeling of depth and motion, without becoming too crowded or complicated. Various CHAR designs allow the program to have this feel. Take a look at the following description, which roughly outlines the Speed Demon program.

Lines	Program Description
100–220	Define the CHAR patterns and colors.
230–260	Ask whether or not you require instructions.
270–350	Title screen is set up.
360–530	The game screen is set up using DISPLAY statements instead of numerous HCHAR and VCHAR statements.
540–600	Set up the constantly moving highway lines which give the illusion of the race car moving down the road. A "hidden sprite" set in line 410 makes the lines appear to be coming from the end of the road. This sprite is the same color as the sky, and it's never set in motion. Thus, the highway line sprites can be constantly in motion, without appearing to go from the top to the bottom of the screen.
610–650	Finish the screen and position the player's race car.
660–670	Highway lines set in motion.
680–970	Main game loop.
980–1110	Car crashes and score is displayed.
1120–1270	Instructions.

Besides the hidden sprite, shadow sprites are also used to give the cars some appearance of depth. Speed Demon also shows how to use screen boundaries by accessing CALL POSITION and CALL COINC statements. One of the biggest challenges in creating this game was trying to design the passing cars, making it seem that they're rushing at you from a distance. This was done with CALL PATTERN statements and a CALL MAGNIFY(4) statement, giving the largest possible area to design ever-increasing CHAR patterns for the cars.

You can increase the other cars' speeds by changing the dot row speed in lines 920 and 930 from 12 to 18. This means you have to react even faster than before, but it makes for a more exciting game.

Speed Demon

```
100 CALL CLEAR
110 CALL COLOR(11,5,1):: CALL COLOR(12,5,1)
120 FOR A=3 TO 8 :: CALL COLOR(A,16,1):: NEXT A
130 CALL COLOR(2,16,1,10,6,6):: CALL COLOR(9,14,14)
140 CALL CHAR(118,"FFFFFFFFFFFFFFFF",119,"F8F8FCFCF
    EFEFFFF",120,"8080C0C0E0F0F0F8")
150 CALL CHAR(121,"1F1F3F3F7F7FFFFF",122,"010103030
    7070F0F")
160 CALL CHAR(104,RPT$("F",64))
170 CALL CHAR(108,"00000000000000000000000000000000
    00000000008080808080800000000000")
```

```
180 CALL CHAR(132,"01313F3101070607E7E1E1FFE3E1E1E0
    808CFC8C80E060E0E78787FFC7878707")
190 CALL CHAR(136,"0000090F09030203313F333131000000
    000020E02080808018F8981818000000")
200 CALL CHAR(140,"00000000010301030507050500000000
    000000000080008040C0404000000000")
210 CALL CHAR(36,"000000000000000000010100000000000
    00000000000008080C0400000000000")
220 CALL CHAR(40,"000000000001030105070500000000000
    00000000000800040C0400000000000")
230 PRINT "FOR INSTRUCTIONS PRESS Y"
240 PRINT :: PRINT :: PRINT
250 PRINT "PRESS N IF INSTRUCTIONS"
260 PRINT "ARE NOT NEEDED."
270 CALL KEY(0,K,S):: IF S=0 THEN 270 :: IF K=89 TH
    EN GOSUB 1120
280 CALL CLEAR
290 CALL SCREEN(5)
300 CALL MAGNIFY(4)
310 DISPLAY AT(6,6):"SPEED"
320 DISPLAY AT(8,6):"DEMON"
330 CALL SOUND(2000,-8,2)
340 CALL SPRITE(#1,132,11,189,170,-6,0):: CALL SPRI
    TE(#2,132,2,192,167,-6,0)
350 CALL SOUND(4000,-8,2):: FOR DELAY=1 TO 1500 ::
    NEXT DELAY
360 CALL CLEAR :: CALL DELSPRITE(ALL):: CALL SCREEN
    (8):: CALL COLOR(2,6,6)
370 CALL HCHAR(1,1,42,128)
380 CALL SPRITE(#10,104,6,1,110)
390 DISPLAY AT(5,13):"zvvx" :: DISPLAY AT(6,13):"yv
    vw"
400 DISPLAY AT(7,12):"zvvvvx" :: DISPLAY AT(8,12):"
    yvvvvw"
410 DISPLAY AT(9,11):"zvvvvvvx" :: DISPLAY AT(10,11
    ):"yvvvvvvw"
420 DISPLAY AT(11,10):"zvvvvvvvvx" :: DISPLAY AT(12
    ,10):"yvvvvvvvvw"
430 DISPLAY AT(13,9):"zvvvvvvvvvvx" :: DISPLAY AT(1
    4,9):"yvvvvvvvvvvw"
440 DISPLAY AT(15,8):"zvvvvvvvvvvvvx" :: DISPLAY AT
    (16,8):"yvvvvvvvvvvvvw"
450 DISPLAY AT(17,7):"zvvvvvvvvvvvvvvx" :: DISPLAY
    AT(18,7):"yvvvvvvvvvvvvvvw"
460 DISPLAY AT(19,6):"zvvvvvvvvvvvvvvvvx" :: DISPLA
    Y AT(20,6):"yvvvvvvvvvvvvvvvvw"
470 DISPLAY AT(21,5):"zvvvvvvvvvvvvvvvvvvx" :: DISP
    LAY AT(22,5):"yvvvvvvvvvvvvvvvvvvw"
480 DISPLAY AT(23,4):"zvvvvvvvvvvvvvvvvvvvvx" :: DI
    SPLAY AT(24,4):"yvvvvvvvvvvvvvvvvvvvvw"
```

```
490 CALL HCHAR(4,1,118,7):: CALL HCHAR(3,3,118,3)::
     CALL HCHAR(2,4,118,1)
500 CALL HCHAR(3,7,97,5):: CALL HCHAR(4,8,97,3):: C
    ALL HCHAR(2,10,97,1)
510 CALL HCHAR(4,11,118,3):: CALL HCHAR(3,13,118,1)
520 CALL HCHAR(4,22,118,6):: CALL HCHAR(3,22,118,1)
530 CALL HCHAR(3,27,97,2):: CALL HCHAR(4,28,97,4)
540 REM HIGHWAY LINES
550 CALL SPRITE(#11,108,16,1,112)
560 CALL SPRITE(#12,108,16,40,112)
570 CALL SPRITE(#13,108,16,79,112)
580 CALL SPRITE(#14,108,16,118,112)
590 CALL SPRITE(#15,108,16,157,112)
600 CALL SPRITE(#16,108,16,196,112)
610 DISPLAY AT(10,2)SIZE(4):"MPH" :: DISPLAY AT(10,
    23)SIZE(5):"SCORE"
620 CALL SPRITE(#1,132,11,155,150):: CALL SPRITE(#2
    ,132,2,158,147)
630 SCORE=0 :: MPH=0 :: HL=0
640 DISPLAY AT(11,2)SIZE(4):MPH :: DISPLAY AT(11,25
    )SIZE(4):SCORE
650 FOR DELAY=1 TO 200 :: NEXT DELAY
660 HL=8 :: MPH=10 :: FOR H=11 TO 16 :: CALL MOTION
    (#H,HL,0):: NEXT H
670 DISPLAY AT(11,3)SIZE(4):MPH
680 GOSUB 860
690 CALL POSITION(#4,Y,X):: IF Y>175 THEN 980
700 CALL COINC(#1,#4,25,R):: IF R=-1 THEN 1000
710 CALL POSITION(#1,Y,X):: IF X<40 THEN 1000 :: IF
     X>190 THEN 1000
720 CALL KEY(0,K,S):: IF S=0 THEN 690
730 IF K=83 THEN 770
740 IF K=68 THEN 780
750 IF K=69 THEN 790
760 IF K=88 THEN 830
770 CALL MOTION(#1,0,-6):: CALL MOTION(#2,0,-6):: G
    OTO 690
780 CALL MOTION(#1,0,6):: CALL MOTION(#2,0,6):: GOT
    O 690
790 MPH=MPH+20 :: DISPLAY AT(11,3)SIZE(4):MPH :: IF
     MPH>300 THEN 1000
800 HL=HL+1 :: FOR H=11 TO 16 :: CALL MOTION(#H,HL,
    0):: NEXT H
810 SCORE=SCORE+10 :: DISPLAY AT(11,25)SIZE(4):SCOR
    E
820 GOTO 690
830 MPH=MPH-20 :: IF MPH<0 THEN 1000
840 DISPLAY AT(11,3)SIZE(4):MPH :: SCORE=SCORE-20 :
    : DISPLAY AT(11,25)SIZE(4):SCORE
850 GOTO 690
```

```
860 RANDOMIZE
870 Z=INT(2*RND)+1
880 S=INT(9*RND)
890 C=INT(11*RND)+6
900 CALL SOUND(1200,-8,2)
910 CALL SPRITE(#4,36,C,20,120):: CALL SPRITE(#5,36
    ,2,23,117)
920 IF Z=1 THEN CALL MOTION(#4,12,S):: IF Z=1 THEN
    CALL MOTION(#5,12,S)
930 IF Z=2 THEN CALL MOTION(#4,12,-S):: IF Z=2 THEN
     CALL MOTION(#5,12,-S)
940 FOR DELAY=1 TO 20 :: NEXT DELAY :: CALL PATTERN
    (#4,40):: CALL PATTERN(#5,40):: FOR DELAY=1 TO
    20 :: NEXT DELAY
950 CALL PATTERN(#4,140):: CALL PATTERN(#5,140):: F
    OR DELAY=1 TO 20 :: NEXT DELAY :: CALL PATTERN(
    #4,136):: CALL PATTERN(#5,136)
960 FOR DELAY=1 TO 20 :: NEXT DELAY :: CALL PATTERN
    (#4,132):: CALL PATTERN(#5,132)
970 RETURN
980 CALL DELSPRITE(#4,#5):: SCORE=SCORE+10 :: DISPL
    AY AT(11,25)SIZE(4):SCORE
990 GOTO 680
1000 CALL MOTION(#1,0,0,#2,0,0,#4,0,0,#5,0,0)
1010 CALL SOUND(3000,-7,2)
1020 FOR Y=1 TO 10 :: CALL COLOR(#1,7,#4,7):: FOR D
     ELAY=1 TO 20 :: NEXT DELAY :: CALL COLOR(#1,11
     ,#4,C):: FOR DELAY=1 TO 20 :: NEXT DELAY :: NE
     XT Y
1030 CALL CLEAR :: CALL COLOR(2,16,1)
1040 CALL DELSPRITE(ALL)
1050 CALL SCREEN(13)
1060 DISPLAY AT(8,2):"SORRY, YOU CRASHED"
1070 DISPLAY AT(12,2):"YOUR SCORE WAS";SCORE
1080 DISPLAY AT(20,2):"PLAY AGAIN-Y/N"
1090 CALL KEY(0,K,S):: IF S=0 THEN 1090
1100 IF K=89 THEN 360
1110 END
1120 CALL CLEAR
1130 CALL SCREEN(13)
1140 PRINT "         SPEED DEMON" :: PRINT
1150 PRINT "  ALPHA LOCK MUST BE ON."
1160 PRINT :: PRINT "USE THE ARROW KEYS:"
1170 PRINT "RIGHT:MOVES RIGHT"
1180 PRINT "LEFT:MOVES LEFT"
1190 PRINT "UP:ACCELERATOR"
1200 PRINT "DOWN:BRAKE"
1210 PRINT
1220 PRINT "SCORING:POINTS FOR PASSING" :: PRINT "A
     ND ACCELERATING" :: PRINT
```

```
1230 PRINT "DANGERS:CRASHES AND GOING" :: PRINT "OF
     F THE ROAD" :: PRINT
1240 PRINT
1250 PRINT "PRESS ANY KEY"
1260 CALL KEY(0,K,S):: IF S=0 THEN 1260
1270 RETURN
```

Blackjack

Jim Rubino

An excellent adaptation of the classic card game, "Blackjack" lets you bet, hit, double-down, or split. Play against the computer dealer or in a three-way hand with a friend. For the unexpanded TI-99/4A with Extended BASIC.

"Blackjack," a computer simulation of the casino-style card game, entertains as it gives you the chance to sharpen your card skills—and unlike "practice" in Las Vegas or Atlantic City—without going broke.

The object of the game is simple—hold a hand of cards which add up to no more than 21. To win, of course, you must have a higher total than the dealer (or the optional other player). In Blackjack, aces may count as either 1 or 11 points, face cards (jack, queen, or king) count as 10, and all other cards count at their face value (2–10).

A *blackjack* is a two-card hand—one ace and either a face card or a ten. Blackjack means you win automatically. If you end up with more than 21 points, you're *busted,* and lose.

In Front of the Green Felt

After a title screen, select either the one- or two-player game. The computer always acts as the dealer. Players place their bets, and the game begins.

Each player is dealt two cards, face up. The computer deals itself one card face up and one face down. If any player has a blackjack, it's announced and that player is finished. If the dealer's face-up card is an ace, each remaining player is allowed to place an *insurance bet.* This is equal to one half the original bet and pays off at two-to-one odds if the dealer has a blackjack. If the dealer doesn't get blackjack, the insurance money is lost and play continues. Depending on your current point value, you may be offered the opportunity to *double-down.* You can now double the amount of your original bet. To make the game more playable, this happens only when it *may* be advantageous. If you choose to double-down, you're dealt a single card face up. Your round is through.

If your first two cards are a pair, you can choose to split them into two hands. If you do, one card is erased and another dealt. This hand is then played to a conclusion. It's then erased, and another hand is dealt.

At all other times, you can ask for a *hit*—another card. The dealer will give you another card to add to the two already showing. Keep asking for cards as long as you want—just make sure not to go bust.

When all players have finished, either through drawing a blackjack, going bust, or staying with their cards, it's the dealer's (computer's) turn. If necessary, it will draw cards until it has either gone bust or reached a score from 17 to 21. Aces will be counted as 11 if they bring the points to 17 to 21, and 1 at all other times.

The hands are then counted and a new round begins. Play continues until one player either runs out of money or enters a bet of $9999999. During the game, players are asked if they want these options: double, hit, insurance, or split. The response to these questions is simply Y for yes, N for no.

Just for informational purposes, the computer uses a two-deck shoe from which the cards are dealt. When the number of cards remaining in the computer dealer's "shoe" reaches 25 or fewer, the dealer reshuffles. Most casinos use a three- or four-deck shoe. They also usually shuffle the decks once 50 or 75 percent of the cards have been dealt. This makes it harder for "card counters" to succeed.

With a two-deck shoe, and reshuffling happening when about 25 percent of the cards remain, Blackjack is close to casino reality. You *could* practice card-counting techniques with this program, but we're making no guarantees that it will help you out at the table, whether on the screen or in real life.

Lines, Lines

Most of the information necessary for the game is contained in the array X(5,14). The computer uses X(1,N). Player 1 uses X(2, N) and X(4, N), while player 2 uses X(3,N) and X(5,N).

Lines	**Function**
100–150	Control prescan. Lines 100, 110, and 150 may be deleted without affecting operation of the program, but doing so will add about 12 seconds to the time between when RUN is entered and execution begins.
160–170	Initialize array variables.
180–240	Subroutine to select and display a card.
250	Deals first cards to players and computer.

260–280	Check for blackjack. If any player has one, a message is displayed.
290–320	Offer an insurance bet if appropriate.
330	If dealer has blackjack, displays it.
340–500	Split, double, and hit routine. This series of lines offers each player the appropriate option at the proper time.
510–560	Draw dealer's hand.
570–730	Score all hands and display scores. Also indicate if a player has won, lost, or tied the computer.
750–830	Subroutine to accept bets. Each player begins the game with $3,000. (Minimum bet is $1. Maximum is the amount of cash the player has left.)
850	Creates a double deck of cards using arrays DEK(104) and SUIT(104).
880–890	Initialize variables.
900–1140	Display titles and define graphics characters.

Blackjack

```
100 GOTO 160 :: P,S,R,C,L,CAS,CARD$,CA,N,A,I,NP,M,J
    ,M$,Y
110 CALL VCHAR :: CALL SCREEN :: CALL HCHAR :: CALL
     KEY :: CALL SOUND :: CALL CLEAR :: CALL COLOR
    :: CALL CHAR
120 OPTION BASE 1 :: DIM X(5,14),R1(8),C1(5),BJ$(8)
    ,DEK(104),SUIT(104),SCORE$(6),CAR$(13)
130 DATA A,2,3,4,5,6,7,8,9,10,J,Q,K,A10,3,10A,7,AJ,
    13,JA,15,AQ,20,QA,23,AK,9,KA,
15,5,10,15,20,25
140 DATA 1,13,3,2,13,6,3,13,9,4,13,6,5,13,9,1,14,1,
    2,14,7,3,14,13,4,14,7,5,14,13
150 !@P-
160 FOR A=1 TO 13 :: READ CAR$(A):: NEXT A :: FOR A
    =1 TO 8 :: READ BJ$(A),R1(A):: NEXT A :: FOR A=
    1 TO 5 :: READ C1(A):: NEXT A
170 CALL CLEAR :: ON WARNING NEXT :: FOR A=1 TO 10
    :: READ R,C,S :: X(R,C)=S :: NEXT A :: GOSUB 91
    0 :: GOTO 250
180 X(S,2)=X(S,2)+1 :: RANDOMIZE :: I=INT(RND*N)+1
    :: CA=DEK(I):: DEK(I)=DEK(N):: CAS=SUIT(I):: SU
    IT(I)=SUIT(N):: N=N-1
190 CARD$=CAR$(CA):: IF CA>9 THEN CA=10 ELSE IF (CA
    =1)*(X(S,7)=0)THEN X(S,4)=X(S,4)+10 :: X(S,7)=1
200 IF (P<>1)*(X(P,9)=0)*(SCORE$(P)=CARD$)THEN X(P,
    9)=1 :: SCORE$(L)=CARD$ :: X(L,3)=X(P,3):: X(L,
    4)=X(P,4):: X(L,7)=X(P,7):: X(L,8)=X(P,8)
210 X(S,3)=X(S,3)+CA :: X(S,4)=X(S,4)+CA :: SCORE$(
    S)=SCORE$(S)&CARD$ :: IF P=1 THEN M$=CARD$ :: Y
    =CAS
```

```
220 R=X(S,14):: C=C1(X(S,2)+1):: CALL SOUND(90,-6,0
    ):: CALL VCHAR(R,C,112,5):: CALL VCHAR(R,C+1,11
    2,5):: CALL VCHAR(R,C+2,112,5)
230 IF ((NP=1)*(M=4))+(M=6)THEN M=0 :: RETURN ELSE
    DISPLAY AT(R+1,C-1)SIZE(LEN(CARD$)):CARD$ :: R=
    R+3 :: C=C+1
240 CALL HCHAR(R,C,CAS):: CALL HCHAR(R,C+1,CAS+1)::
     CALL HCHAR(R+1,C,CAS+2):: CALL HCHAR(R+1,C+1,C
    AS+3):: RETURN
250 GOSUB 760 :: FOR A=1 TO I :: M=M+1 :: P=S :: L=
    P+2 :: CALL SCREEN(X(S,13)):: GOSUB 180 :: IF S
    =NP+1 THEN S=1 ELSE S=S+1
260 NEXT A :: CALL SCREEN(13):: CALL SOUND(150,110,
    3,300,3,500,3)
270 FOR P=1 TO 3 :: FOR I=1 TO 8 :: IF BJ$(I)=SCORE
    $(P)THEN X(P,1)=X(P,1)+INT(X(P,8)*1.5):: X(P,5)
    =1
280 NEXT I :: IF (P>1)*(X(P,5)=1)THEN DISPLAY AT(R1
    (P+5),13)BEEP:"BLACKJACK"
290 NEXT P :: FOR P=2 TO NP+1
300 IF (SEG$(SCORE$(1),1,1)="A")*(X(P,5)=0)*((X(P,1
    )-X(P,8))>=X(P,8)/2)THEN DISPLAY AT(R1(P+5),13)
    BEEP:"INSURANCE?" ELSE 330
310 CALL KEY(3,I,J):: IF J=0 THEN 310 :: CALL SOUND
    (90,220,0):: CALL HCHAR(R1(P+5),13,32,15)
320 IF (I=89)*(X(1,5)=1)THEN X(P,1)=X(P,1)+X(P,8)EL
    SE IF (I=89)*(X(1,5)=0)THEN X(P,1)=X(P,1)-INT((
    X(P,8)*.5)+.5)
330 NEXT P :: S=0 :: IF X(1,5)=1 THEN S,P=1 :: CARD
    $=M$ :: CAS=Y :: GOSUB 220 :: DISPLAY AT(3,13):
    "BLACKJACK" :: GOTO 570
340 X(5,2),X(0,2)=0 :: S=S+2 :: IF (S=6)*(NP=1)THEN
     510 ELSE IF (S=6)*(NP=2)THEN S=3 ELSE IF S=7 T
    HEN 510
350 IF X(S,5)=1 THEN 500 ELSE IF S/2=INT(S/2)THEN P
    =2 ELSE P=3
360 ON X(S,9)+1 GOTO 420,370,400,500
370 L=P+2 :: CALL SCREEN(X(S,13)):: DISPLAY AT(X(S,
    14)+2,13)BEEP:"SPLIT?"
380 CALL KEY(3,I,J):: IF J=0 THEN 380 :: CALL SOUND
    (90,220,3):: CALL HCHAR(X(S,14)+2,13,32,8):: IF
     I=78 THEN 420 ELSE IF I<>89 THEN 370
390 X(L,9)=2 :: SCORE$(S)=SCORE$(L):: X(S,2)=0 :: X
    (S,3)=X(L,3):: X(S,4)=X(L,4):: X(S,7)=X(L,7)::
    GOTO 410
400 CALL SCREEN(X(S,13)):: CALL SOUND(90,220,3):: X
    (S,2)=0
410 A=X(S,14):: FOR A=A TO A+5 :: CALL HCHAR(A,9,32
    ,23):: NEXT A :: GOSUB 180
```

```
420 CALL SCREEN(X(S,13)):: IF ((X(S,7)=0)*((X(S,3)<
    8)+(X(S,3)>11)))+((X(S,7)=1)*((X(S,4)<13)+(X(S,
    4)>18)))THEN 460
430 IF (X(P,1)-X(P,8))>=X(P,8)THEN DISPLAY AT(R1(P+
    5),13)BEEP:"DOUBLE?" ELSE 460
440 CALL KEY(3,I,J):: IF J=0 THEN 440 :: CALL SOUND
    (90,220,3):: CALL HCHAR(R1(P+5),15,32,7)
450 IF I=89 THEN X(S,8)=X(S,8)*2 :: GOSUB 180 :: X(
    S,6)=1 :: GOTO 500 ELSE IF I<>78 THEN 430
460 IF X(S,5)=1 THEN 500 ELSE R=X(S,14)+2 :: C=C1(X
    (S,2)):: DISPLAY AT(R,C+8)BEEP:"HIT?"
470 CALL KEY(3,I,J):: IF J=0 THEN 470 :: CALL SOUND
    (90,220,3):: CALL HCHAR(R1(P+5),C+8,32,6):: IF
    I=78 THEN 500 ELSE IF I<>89 THEN 460
480 GOSUB 180 :: IF (X(S,2)=4)*(X(S,3)<22)THEN X(S,
    1)=X(S,8)*2+X(S,1):: X(S,5)=1
490 IF X(S,3)<21 THEN 460
500 GOTO 340
510 S,P=1 :: X(1,2)=1 :: CALL SCREEN(X(P,13)):: CAR
    D$=M$ :: CAS=Y :: GOSUB 220
520 FOR A=2 TO 5 :: IF ((X(A,3)>0)*(X(A,3)<22))*(X(
    A,5)=0)THEN 550
530 NEXT A :: GOTO 570
540 IF X(1,2)=4 THEN X(1,2)=-1
550 IF X(1,4)<17 THEN GOSUB 180 :: GOTO 540 ELSE IF
     X(1,4)<22 THEN 570
560 IF X(1,3)<17 THEN GOSUB 180 :: GOTO 540
570 CALL SOUND(1250,200,3,300,3,400,3,-2,3)!SCORE R
    OUTINE
580 FOR P=1 TO 5 :: IF X(P,4)<22 THEN X(P,3)=X(P,4)
590 NEXT P :: DISPLAY AT(20,1):"DEALER   ";X(1,3)::
     DISPLAY AT(22,1):"PLAYER 1 ";X(2,3):: IF NP=2
    THEN DISPLAY AT(24,1):"PLAYER 2 ";X(3,3)
600 IF X(4,9)=2 THEN DISFLAY AT(22,20):X(4,3)
610 IF X(5,9)=2 THEN DISPLAY AT(24,20):X(5,3)
620 IF X(1,3)>21 THEN X(1,3)=0
630 FOR A=2 TO 5 :: IF (A=2)+(A=4)THEN R=22 ELSE R=
    24
640 IF A<4 THEN C=15 ELSE C=25
650 IF (X(A,9)=3)+(X(A,8)=0)THEN 710
660 IF X(1,3)=X(A,3)THEN DISPLAY AT(R,C)SIZE(3):"TI
    E" :: GOTO 710
670 IF X(A,5)=1 THEN DISPLAY AT(R,C)SIZE(3):"WIN" :
    : GOTO 710
680 IF X(A,3)>21 THEN X(A,1)=X(A,1)-X(A,8):: DISPLA
    Y AT(R,C)SIZE(4):"BUST" :: GOTO 710
690 IF X(A,3)<X(1,3)THEN X(A,1)=X(A,1)-X(A,8):: DIS
    PLAY AT(R,C)SIZE(4):"LOSE" :: GOTO 710
700 IF X(A,3)>X(1,3)THEN X(A,1)=X(A,1)+X(A,8):: DIS
    PLAY AT(R,C)SIZE(3):"WIN"
```

```
710 NEXT A :: IF X(4,9)=2 THEN X(2,1)=X(2,1)+X(4,1)
720 IF X(5,9)=2 THEN X(3,1)=X(3,1)+X(5,1)
730 FOR A=1 TO 1000 :: NEXT A :: CALL CLEAR :: GOTO
     250
740 CALL CLEAR :: END
750 REM ACCEPT BETS
760 FOR A=1 TO 11 :: CALL COLOR(A,2,1):: NEXT A ::
    FOR A=1 TO NP :: IF X(A+1,1)<
1 THEN 740 :: NEXT A
770 FOR I=1 TO 5 :: SCORE$(I)="" :: FOR J=2 TO 11 :
    : X(I,J)=0 :: NEXT J :: NEXT I :: P=6 :: CALL S
    CREEN(14):: RESTORE 860
780 X(4,1),X(5,1)=0 :: X(4,9),X(5,9)=3
790 FOR A=1 TO NP :: FOR I=1 TO P :: READ C,M$ :: D
    ISPLAY AT(R1(I),C):M$ :: NEXT I :: P=P-2 :: NEX
    T A
800 DISPLAY AT(9,4)SIZE(7):"$";X(2,1):: IF NP=2 THE
    N DISPLAY AT(9,17)SIZE(7):"$";X(3,1)
810 ACCEPT AT(15,6)VALIDATE(DIGIT)SIZE(7)BEEP:X(2,8
    ):: X(4,8)=X(2,8):: IF X(2,8)=9999999 THEN 740
    ELSE IF X(2,8)>X(2,1)THEN 810
820 IF NP=2 THEN ACCEPT AT(15,19)VALIDATE(DIGIT)SIZ
    E(7)BEEP:X(3,8):: X(5,8)=X(3,8):: IF X(3,8)=999
    9999 THEN 740 ELSE IF X(3,8)>X(3,1)THEN 820
830 CALL SOUND(125,1400,0):: CALL CLEAR :: IF N>25
    THEN 880 ELSE DISPLAY AT(12,10):"SHUFFLING"
840 A=1 :: FOR P=1 TO 2 :: FOR I=1 TO 4 :: FOR J=1
    TO 13 :: DEK(A)=J :: A=A+1 :: NEXT J :: NEXT I
    :: NEXT P :: P=1 :: A=96 :: N=104
850 FOR I=1 TO 4 :: FOR J=1 TO 26 :: SUIT(P)=A :: P
    =P+1 :: NEXT J :: A=A+4 :: NEXT I
860 DATA 4,PLAYER 1,6,CASH,6,BET?,4,$,7,PLACE YOUR
    BET,6,THEN PRESS ENTER
870 DATA 17,PLAYER 2,19,CASH,19,BET?,17,$
880 CALL CLEAR :: FOR A=2 TO 8 :: CALL COLOR(A,5,16
    ):: NEXT A :: CALL COLOR(9,7,16,10,2,16,11,16,1
    6)
890 X(1,2),X(2,2),X(3,2),X(4,2),X(5,2)=-1 :: S=2 ::
     IF NP=1 THEN I=4 :: RETURN ELSE I=6 :: RETURN
900 REM TITLES
910 X(2,1)=3000 :: X(3,1)=3000 :: CALL SCREEN(4)::
    CALL CLEAR :: RESTORE 980 :: FOR A=1 TO 4 :: RE
    AD R,C,M$ :: DISPLAY AT(R,C):M$ :: NEXT A
920 DISPLAY AT(R,C)BEEP:M$
930 FOR A=1 TO 27 :: CALL KEY(3,I,J):: IF J<>0 THEN
     950
940 NEXT A :: CALL HCHAR(R,C,32,20):: FOR A=1 TO 25
     :: NEXT A :: GOTO 920
950 IF P=1 THEN 1030
```

```
960 P=P+1 :: CALL SOUND(90,220,0):: CALL CLEAR :: F
    OR A=1 TO 9 :: READ R,C,M$ :: DISPLAY AT(R,C):M
    $ :: NEXT A
970 GOTO 920
980 DATA 6,10,BLACKJACK,11,4,A CARD GAME FOR 1 OR 2
    ,16,11,PLAYERS
990 DATA 24,8,PRESS ANY KEY
1000 DATA 1,1,ACES COUNT 1 OR 11. FACE    CARDS COU
     NT 10. OTHER CARDS ARE WORTH FACE VALUE.
1010 DATA 6,8,WINNING HANDS,8,2,BLACKJACK
     3 TO 2,10,2,5 CARDS UNDER 21   2 TO 1,12,2,INS
     URANCE          2 TO 1
1020 DATA 14,2,OTHER BETS          1 TO 1,18,1,TO EN
     D THE GAME ENTER A BET,20,1,OF 9999999.,24,8,P
     RESS ANY KEY
1030 REM HOW MANY PLAYERS?
1040 CALL SOUND(90,220,0):: CALL CLEAR :: CALL SCRE
     EN(8)
1050 CALL CHAR(96,"00001C3E7F7F7F7F0000387CFEFEFEFE
     3F1F0F07030100000FCF8F0E0C08")!HEART
1060 CALL CHAR(100,"000103070F1F3F7F0080C0E0F0F8FCF
     E7F3F1F0F07030100FEFCF8F0E0C08")!DIAMOND
1070 CALL CHAR(104,"010103070F1F3F7F000080C0E0F0F8F
     C7F7F7F7F3F190101FCFCFCFCF83")!SPADE
1080 CALL CHAR(108,"0103070707033179080C0C0C080183
     C7F7F79310101010 0FCFC3C18")!CLUB
1090 CALL CHAR(112,"FFFFFFFFFFFFFFFF")
1100 DISPLAY AT(11,7)BEEP:"1 OR 2 PLAYERS?"
1110 FOR A=1 TO 24 :: CALL KEY(3,I,J):: IF (I>48)*(
     I<51)THEN 1130
1120 NEXT A :: CALL CLEAR :: FOR A=1 TO 25 :: NEXT
     A :: GOTO 1100
1130 NP=I-48 :: CALL SOUND(90,220,0):: CALL CLEAR
1140 RETURN
```

Flood Waters

Lee Suydam

An entertaining game for children and adults alike, "Flood Waters" tests your addition agility as you move up an incline, just ahead of the rising water, before the rolling ball knocks you down. Works on the unexpanded TI-99/4A with TI BASIC.

It's even worse than the Johnstown Flood. The dam's not burst, but there's a huge water faucet descending from the clouds, and it's pouring water down around your ears as if some giant just turned on the tap.

The Water's Rising

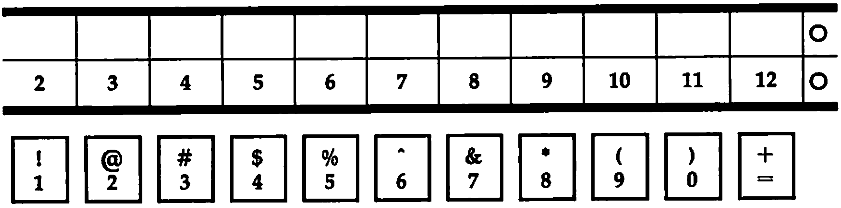
As the game begins, the sky faucet turns on and water streams down to flood the base of the incline, threatening to drown you. Fortunately, the rising waters stop just in the nick of time. A pair of dice appear at the top of the screen—at the same instant a red ball begins rolling down the incline. Not to worry—simply add the dice together, respond correctly, and you're temporarily safe and one step higher on the incline. If you fail to give the correct response before the red ball strikes you, the water faucet turns on again, flooding another three levels. If you're trapped underwater—well, you know you don't have gills. The trick in "Flood Waters" is to answer correctly and stay high and dry.

As you climb higher, it becomes more and more difficult to answer before the red ball bowls you over. As you shorten the distance, there's less and less time to respond. When you get past the ladder on the side of the incline, your error rate usually increases. Only the quick and resourceful will reach the top and receive the CONGRATULATIONS message.

Make an Overlay

Before you play, mark a keyboard overlay from 2 to 12, as in the figure below. This allows you to use the top 12 keys of the keyboard as answers 2–12. A bundle of spare overlays was supplied with your computer.

Flood Waters Overlay



How Flood Waters Works

Lines	Program Structure
100	Title.
110–240	Give reward or failure message.
250–310	Establish initial variables and conditions.
320–360	Define color and numbers charts 88–95 (blue water).
370–410	Define color and numbers charts 96–103 (red faucet and building parts).
420–480	Define color and numbers charts 104–122 (black incline, running figure, and other parts).
500–550	Cause blue water to rise in each of 32 columns or one line.
560–570	Cause splash of falling water (24) to advance.
580	Counts A, for rows of water.
600	Returns program to end of game message when water overtakes running figure.
620–860	Draw water faucet, incline, and running figure.
870–890	Description of water.
900–910	Description of ball and faucet.
920–930	Description of ladder, incline, figure, and window.
960–970	Description of dice one, two, three, and four.
980–990	Description of dice five, six, and water drop.
1000	Causes water to advance three rows.
1010	Draws a string of water drops.
1020	Resets value of A.
1030–1160	Give random pairs of dice, draw them on the screen, and assign the correct response (B).
1170–1220	Advance red ball.
1240	Tests for correct response.
1250–1300	Test for contact between red ball and running figure.
1310–1350	Advance figure and test for win message.

Flood Waters

```
100 REM FLOOD WATERS
110 CALL CLEAR
120 IF R1=2 THEN 130 ELSE 160
130 PRINT "   CONGRATULATIONS!!!"
140 PRINT : : : : : :
150 GOTO 230
160 IF ROW=R1-2 THEN 170 ELSE 250
170 FOR X=1 TO 18
180 PRINT TAB(X);"0"
190 PRINT
200 NEXT X
210 PRINT "GLUB! GLUB! GLUB!"
220 PRINT : : : : : :
230 FOR D=1 TO 1000
240 NEXT D
250 RESTORE
260 RANDOMIZE
270 A=1
280 ROW=23
290 R1=21
300 C1=11
310 CALL SCREEN(16)
320 CALL COLOR(8,5,1)
330 FOR C=88 TO 95
340 READ C$
350 CALL CHAR(C,C$)
360 NEXT C
370 CALL COLOR(9,10,1)
380 FOR C=96 TO 103
390 READ C$
400 CALL CHAR(C,C$)
410 NEXT C
420 FOR X=10 TO 12
430 CALL COLOR(X,2,1)
440 NEXT X
450 FOR C=104 TO 122
460 READ C$
470 CALL CHAR(C,C$)
480 NEXT C
490 GOSUB 620
500 FOR Y=24 TO 1 STEP -1
510 FOR N=88 TO 95
520 CALL VCHAR(4,7,95,20)
530 CALL HCHAR(Y,1,N,32)
540 CALL SOUND(10,1000-(20*X),2)
550 NEXT N
560 ROW=ROW-1
570 CALL HCHAR(ROW+1,6,24,3)
580 A=A+1
```

```
590 GOTO 1000
600 IF ROW=R1-2 THEN 110
610 NEXT Y
620 REM   FAUCET&INCLINE
630 PRINT "zbcd"
640 PRINT "aaaag"
650 PRINT "zzefah";TAB(28);"j"
660 PRINT TAB(27);"ja"
670 PRINT TAB(26);"jla"
680 PRINT TAB(25);"jaaa"
690 PRINT TAB(24);"jaala"
700 PRINT TAB(23);"jaaaaa"
710 PRINT TAB(22);"jalaala"
720 PRINT TAB(21);"jiaaaaaa"
730 PRINT TAB(20);"jaialaala"
740 PRINT TAB(19);"jaaiaaaaaa"
750 PRINT TAB(18);"jalaialaala"
760 PRINT TAB(17);"jaaaaiaaaaaa"
770 PRINT TAB(16);"jaaalaialaala"
780 PRINT TAB(15);"jalaaaaiaaaaaa"
790 PRINT TAB(14);"jaaaaalaialaala"
800 PRINT TAB(13);"jaaaaaaaaiaaaaaa"
810 PRINT TAB(12);"jalaalaalaialaala"
820 PRINT TAB(11);"jaaaaaaaaaaiaaaaaa"
830 PRINT TAB(9);"kjaaalaalaalaialaala"
840 PRINT TAB(9);"jaaaaaaaaaaaaiaaaaaa"
850 PRINT TAB(8);"jaalaalaalaalaialaala"
860 RETURN
870 REM FLOODING WATER
880 DATA 00000000000000FF,000000000000FFFF,00000000
    00FFFFFF,00000000FFFFFFFF
890 DATA 000000FFFFFFFFFF,0000FFFFFFFFFFFF,00FFFFFF
    FFFFFFFF,FFFFFFFFFFFFFFFF
900 REM BALL  a b c d
910 DATA 003C7E7E7E7E3C,FFFFFFFFFFFFFFFF,1038FF5F38
    10,0018FFFF18187EFF,081CFAFE1C08
920 REM e f g h
930 DATA 9F7E,0000000003070307,FCE4F3FFFFFFFFFF,000
    00000E0C0E0C0
940 REM i j k l
950 DATA 427E42427E42427E,0103070F1F3F7FFF,18187FF0
    3C24E483,111111FF111111FF
960 REM  m n o p
970 DATA 0000001818,C0C0000000000303,C0C00018180003
    03(C3C300000000C3C3
980 REM q r s t u v w x y z
990 DATA C3C3001818000C3C3,C3C300C3C300C3C3,10101010
    10101010,0808080808080808,000010103838783,,,,,
1000 IF A<>4 PHAN 600
1010 CALL VCHAR(4,7,117,ROW-1)
1020 A=1
```

```
1030 Q=INT(RND*6)+109
1040 W=INT(RND*6)+109
1050 CALL VCHAR(1,14,Q)
1060 CALL SOUND(100,-1,2)
1070 CALL VCHAR(1,17,W)
1080 CALL SOUND(50,-2,10)
1090 IF Q+W<=226 THEN 1100 ELSE 1120
1100 B=Q+W-169
1110 GOTO 1170
1120 IF Q+W=227 THEN 1130 ELSE 1150
1130 B=48
1140 GOTO 1170
1150 IF Q+W=228 THEN 1160 ELSE 1170
1160 B=61
1170 C2=32
1180 FOR R2=1 TO R1
1190 C2=C2-1
1200 CALL VCHAR(R2,C2,96)
1210 CALL SOUND(300,-2,15)
1220 CALL VCHAR(R2,C2,122)
1230 CALL KEY(0,K,ST)
1240 IF K=(B)THEN 1300
1250 IF ST=0 THEN 1260
1260 IF R2<>R1 THEN 1290
1270 CALL VCHAR(R1,C1,107)
1280 GOTO 600
1290 NEXT R2
1300 CALL VCHAR(R1,C1,122)
1310 R1=R1-1
1320 C1=C1+1
1330 CALL VCHAR(R1,C1,107)
1340 IF R1=2 THEN 110
1350 GOTO 1030
```

Macro/Micro Maze

Bob Foley

A maze game with a difference. You're trapped in a complex maze, and you can't even see the entire pattern. Escape before time—and you—expire. For an expanded TI-99/4A (see exception below) with either TI BASIC or Extended BASIC.

Requiring a good memory and quick thinking, "Macro/Micro Maze" isn't your standard maze game. It's more than a rat-to-the-cheese kind of game.

It's unique because it displays three different representations of the same maze.

- **The Minimaze.** This is the maze that's randomly constructed during the game setup. It's a 23 × 31 character maze that takes up the entire screen.
- **The Macromaze.** A "zoom in" of nine characters of the original maze. You play the game here.
- **The Micromaze.** Perhaps the most interesting of the three, it's a 4 × 3 character representation of the original maze, displayed above the Macromaze to help you find your way.

The object of the game is to move through the Macromaze, pick up the five green squares, and get out of the Minimaze in as little time as possible. You must exit through the yellow square on the left (but only after picking up all five green squares).

If you've picked up all the green squares and you think you're running out of time (assuming you're far from the exit), you can press the T key (for Transport). This moves your red square randomly to another part of the maze, perhaps closer to the exit.

Move through the Macromaze by pressing the arrow keys (make sure the ALPHA LOCK key is depressed). On the screen, it will seem as if the red square is stationary and the maze moves around *it*.

The number of times you get to see the complete Minimaze during the game is determined by your answers to the questions at the beginning of the program. To see the Minimaze, press M. The number of Minimaze "looks" you have left is displayed in the upper-left corner.

If you're frustrated or lost, with no Minimaze looks left, you can press G to give up. This displays the original maze and ends the program.

Of course, all this movement and looking is being done under a time limit. Fail to retrieve the green squares and exit before the time's up, and you've lost the game. Type RUN to play again. (You can increase the time limit by altering line 2570. For instance, change it to IF TIME>2000 THEN 3110 to double the time you have to exit.)

Memory Notes

Macro/Micro Maze is a large program, and takes up almost 16K of your TI-99/4A's memory. If you're loading it from disk, you *must* have the 32K Memory Expansion card installed. If you're using tape, however, you *can* load and run this game with an unexpanded TI. (If you purchase the companion disk to this book—see the coupon in the back of the book for details—and want to run the game on the 16K TI, you'll have to load it into the computer from disk, then save it out to tape. Load the tape file—with OLD CS1—follow the prompts, and you should have no trouble.)

Macro/Micro Maze

```
50 CALL CLEAR
60 PRINT TAB(8);"MACRO/MICRO-MAZE": : : : : : : : :
   : :"(4-5 MINUTE SETUP TIME)"
70 FOR I=1 TO 400
80 NEXT I
90 RANDOMIZE
100 A$="FFFFFFFFFFFFFFFF"
110 CALL CHAR(128,A$)
120 CALL CHAR(136,A$)
130 CALL CHAR(142,"0")
140 CALL CHAR(96,A$)
150 CALL CHAR(143,"FF818181818181FF")
160 CALL CHAR(97,"0")
170 CALL COLOR(10,15,15)
180 CALL COLOR(11,5,1)
190 CALL COLOR(12,5,1)
200 CALL CLEAR
210 INPUT "DO YOU WANT TO SEE THE MAZE BEING BUILT(
   Y/N)? (Y-5 MINI-MAZES, N-6 MINIMAZES)":BU$
220 IF (BU$<>"N")*(BU$<>"Y")THEN 210
230 COL=-ASC(BU$)+94
240 MINI=17-INT((ASC(BU$)/7))
250 CALL COLOR(13,COL,1)
```

```
260 CALL COLOR(14,16,7)
270 CALL COLOR(9,13,11)
280 PRINT : : : : :"DO YOU WANT THE MICROMAZE   (Y/
    N)? (Y-SUBTRACT 2 MINI-  MAZES, N-ADD 2 MINIMAZ
    ES)": :
290 INPUT " NOTE: IF 'Y',ADD 1 MINUTE  TO STARTUP T
    IME -":MMAZ$
300 IF (MMAZ$<>"Y")*(MMAZ$<>"N")THEN 280
310 MINI=MINI+(-2*(SGN(ASC(MMAZ$)-80)))
320 CALL CLEAR
330 CALL SCREEN(16)
340 OPTION BASE 1
350 DIM MAC(24,32)
360 DIM B$(12)
370 TRANS=1
380 FOR I=1 TO 23 STEP 2
390 CALL VCHAR(1,I+1,128,23)
400 CALL HCHAR(I,2,128,31)
410 NEXT I
420 FOR I=26 TO 32 STEP 2
430 CALL VCHAR(1,I,128,23)
440 NEXT I
450 X=(2*(INT(RND*8)+4))+1
460 Y=2*(INT(RND*6)+4)
470 XX=X
480 YY=Y
490 FOR S=1 TO 40
500 FOR Z=1 TO 25
510 TR=0
520 Q=INT(RND*4)+1
530 TR=TR+1
540 IF TR>11 THEN 850
550 QW=0
560 ON Q GOTO 570,640,710,780
570 YY=YY-2
580 SY=2
590 SX=0
600 GOSUB 970
610 IF QW THEN 520
620 CALL VCHAR(YY,XX,136,3)
630 GOTO 840
640 XX=XX+2
650 SX=-2
660 SY=0
670 GOSUB 970
680 IF QW THEN 520
690 CALL HCHAR(YY,XX-2,136,3)
700 GOTO 840
710 YY=YY+2
720 SY=-2
```

```
730 SX=0
740 GOSUB 970
750 IF QW THEN 520
760 CALL VCHAR(YY-2,XX,136,3)
770 GOTO 840
780 XX=XX-2
790 SX=2
800 SY=0
810 GOSUB 970
820 IF QW THEN 520
830 CALL HCHAR(YY,XX,136,3)
840 NEXT Z
850 D=D+1
860 IF D<4 THEN 910
870 XX=(2*(INT(RND*15)+1))+1
880 YY=2*(INT(RND*11)+1)
890 RANDOMIZE
900 GOTO 930
910 XX=X
920 YY=Y
930 CALL GCHAR(YY,XX,GC)
940 IF GC<>136 THEN 860
950 NEXT S
960 GOTO 1040
970 IF (XX<2)+(XX>32)+(YY<1)+(YY>23)THEN 1000
980 CALL GCHAR(YY,XX,GC)
990 IF GC=32 THEN 1030
1000 QW=1
1010 YY=YY+SY
1020 XX=XX+SX
1030 RETURN
1040 FOR YY=2 TO 22 STEP 2
1050 FOR XX=3 TO 31 STEP 2
1060 CALL GCHAR(YY,XX,GC)
1070 IF GC<>32 THEN 1290
1080 TR=0
1090 Q=INT(RND*4)+1
1100 TR9TR+1
1110 IF TR=11 THEN 1290
1120 IF ((YY=2)*(Q=1))+((YY=22)*(Q=3))+((XX=0)*(Q=4
     ))+((XX=31)*(Q=2))THEN 1090
1130 ON Q GOTO 1100,1180,1220,1260
1140 CALL GCHAR(YY-2,XX,GC)
1150 IF GC<>136 THEN 1090
1160 CALL VCHAR(YY-2,XX,136,3)
1170 GOTO 1290
1180 CALL GCHAR(YY,XX+2,GC)
1190 IF GC<>136 THEN 1090
1200 CALL HCHAR(YY,XX,136,3)
1210 GOTO 1290
```

```
1220 CALL GCHAR(YY+2,XX,GC)
1230 IF GC<>136 THEN 1090
1240 CALL VCHAR(YY,XX,136,3)
1250 GOTO 1290
1260 CALL GCHAR(YY,XX-2,GC)
1270 IF GC<>136 THEN 1090
1280 CALL HCHAR(YY,XX-2,136,3)
1290 NEXT XX
1300 NEXT YY
1310 IF TR=0 THEN 1340
1320 TR=0
1330 GOTO 1040
1340 CALL COLOR(13,5,1)
1350 CALL HCHAR(24,1,136,32)
1360 CALL VCHAR(1,1,136,24)
1370 FOR XX=1 TO 32
1380 FOR YY=1 TO 24
1390 CALL GCHAR(YY,XX,MAC(YY,XX))
1400 NEXT YY
1410 NEXT XX
1420 IF MMAZ$="N" THEN 2070
1430 FOR CH=116 TO 127
1440 A=0
1450 READ STY,STX
1460 DATA 1,1,9,1,17,1,1,9,9,9,17,9,1,17,9,17,17,17
     ,1,25,9,25,17,25
1470 FOR I=1 TO 16
1480 M1=MAC(STY+A,STX+B)
1490 M2=MAC(STY+A,STX+B+1)
1500 M3=MAC(STY+A,STX+B+2)
1510 M4=MAC(STY+A,STX+B+3)
1520 IF B=4 THEN 1550
1530 B=4
1540 GOTO 1570
1550 B=0
1560 A=A+1
1570 IF (M1=136)*(M2=136)*(M3=136)*(M4=136)=0 THEN
     1600
1580 A$="0"
1590 GOTO 2030
1600 IF (M1=136)*(M2=136)*(M3=136)*(M4=128)=0 THEN
     1630
1610 A$="1"
1620 GOTO 2030
1630 IF (M1=136)*(M2=136)*(M3=128)*(M4=136)=0 THEN
     1660
1640 A$="2"
1650 GOTO 2030
1660 IF (M1=136)*(M2=136)*(M3=128)*(M4=128)=0 THEN
     1690
```

```
1670 A$="3"
1680 GOTO 2030
1690 IF (M1=136)*(M2=128)*(M3=136)*(M4=136)=0 THEN
     1720
1700 A$="4"
1710 GOTO 2030
1720 IF (M1=136)*(M2=128)*(M3=136)*(M4=128)=0 THEN
     1750
1730 A$="5"
1740 GOTO 2030
1750 IF (M1=136)*(M2=128)*(M3=128)*(M4=136)=0 THEN
     1780
1760 A$="6"
1770 GOTO 2030
1780 IF (M1=136)*(M2=128)*(M3=128)*(M4=128)=0 THEN
     1810
1790 A$="7"
1800 GOTO 2030
1810 IF (M1=128)*(M2=136)*(M3=136)*(M4=136)=0 THEN
     1840
1820 A$="8"
1830 GOTO 2030
1840 IF (M1=128)*(M2=136)*(M3=136)*(M4=128)=0 THEN
     1870
1850 A$="9"
1860 GOTO 2030
1870 IF (M1=128)*(M2=136)*(M3=128)*(M4=136)=0 THEN
     1900
1880 A$="A"
1890 GOTO 2030
1900 IF (M1=128)*(M2=136)*(M3=128)*(M4=128)=0 THEN
     1930
1910 A$="B"
1920 GOTO 2030
1930 IF (M1=128)*(M2=128)*(M3=136)*(M4=136)=0 THEN
     1960
1940 A$="C"
1950 GOTO 2030
1960 IF (M1=128)*(M2=128)*(M3=136)*(M4=128)=0 THEN
     1990
1970 A$="D"
1980 GOTO 2030
1990 IF (M1=128)*(M2=128)*(M3=128)*(M4=136)=0 THEN
     2020
2000 A$="E"
2010 GOTO 2030
2020 A$="F"
2030 B$(CH-115)=B$(CH-115)&A$
2040 NEXT I
2050 CALL CHAR(CH,B$(CH-115))
```

```
2060 NEXT CH
2070 CALL VCHAR(Y,X,143)
2080 MAC(Y,X)=143
2090 FOR I=1 TO 5
2100 XX=(2*(INT(RND*15)+1))+1
2110 YY=2*(INT(RND*11)+1)
2120 IF (MAC(YY,XX)=143)+(MAC(YY,XX)=96)THEN 2100
2130 MAC(YY,XX)=96
2140 CALL VCHAR(YY,XX,96)
2150 NEXT I
2160 YY=2*(INT(RND*11)+1)
2170 CALL VCHAR(YY,2,97)
2180 MAC(YY,2)=97
2190 CALL CLEAR
2200 PRINT "M=MINIMAZE-";MINI
2210 PRINT "T=TRANSPORT-";TRANS
2220 PRINT "G=GIVE UP": : : : : : : : : : : : : :
     : : : : : :
2230 Q$="USE ARROW KEYS TO MOVE"
2240 FOR I=1 TO LEN(Q$)
2250 CALL VCHAR(I+1,29,ASC(SEG$(Q$,I,1)))
2260 NEXT I
2270 CALL HCHAR(4,5,104,21)
2280 CALL HCHAR(24,5,104,21)
2290 CALL VCHAR(4,5,104,21)
2300 CALL VCHAR(4,25,104,21)
2310 IF MMAZ$="N" THEN 2390
2320 CH=115
2330 FOR I=19 TO 22
2340 FOR J=1 TO 3
2350 CH=CH+1
2360 CALL VCHAR(J,I,CH)
2370 NEXT J
2380 NEXT I
2390 FOR I=6 TO 11
2400 CALL VCHAR(5,I,MAC(Y-1,X-1),6)
2410 CALL VCHAR(11,I,MAC(Y,X-1),7)
2420 CALL VCHAR(18,I,MAC(Y+1,X-1),6)
2430 NEXT I
2440 FOR I=12 TO 18
2450 CALL VCHAR(5,I,MAC(Y-1,X),6)
2460 IF (I<13)+(I>17)THEN 2480
2470 CALL VCHAR(12,I,142,5)
2480 CALL VCHAR(18,I,MAC(Y+1,X),6)
2490 NEXT I
2500 FOR I=19 TO 24
2510 CALL VCHAR(5,I,MAC(Y-1,X+1),6)
2520 CALL VCHAR(11,I,MAC(Y,X+1),7)
2530 CALL VCHAR(18,I,MAC(Y+1,X+1),6)
2540 NEXT I
```

```
2550 CALL KEY(0,K,S)
2560 TIME=TIME+1
2570 IF TIME>1000 THEN 3110
2580 CALL SOUND(10,1000,0)
2590 IF K<>69 THEN 2650
2600 IF MAC(Y-1,X)=128 THEN 2550
2610 Y=Y-1
2620 SY=1
2630 SX=0
2640 GOTO 2820
2650 IF K<>68 THEN 2710
2660 IF MAC(Y,X+1)=128 THEN 2550
2670 X=X+1
2680 SX=-1
2690 SY=0
2700 GOTO 2820
2710 IF K<>88 THEN 2770
2720 IF MAC(Y+1,X)=128 THEN 2550
2730 Y=Y+1
2740 SY=-1
2750 SX=0
2760 GOTO 2820
2770 IF K<>83 THEN 2890
2780 IF (MAC(Y,X-1)=128)+((MAC(Y,X-1)=97)*(M<>5))TH
     EN 2550
2790 X=X-1
2800 SX=1
2810 SY=0
2820 IF (MAC(Y,X)<>96)*(MAC(Y,X)<>97)THEN 2860
2830 M=M+1
2840 CALL SOUND(10,300,1)
2850 IF M=6 THEN 3010
2860 MAC(Y,X)=143
2870 MAC(Y+SY,X+SX)=136
2880 GOTO 2390
2890 IF (K=77)*(MINI>0)=0 THEN 2990
2900 MINI=MINI-1
2910 FOR XX=1 TO 32
2920 FOR YY=1 TO 24
2930 CALL VCHAR(YY,XX,MAC(YY,XX))
2940 NEXT YY
2950 NEXT XX
2960 FOR I=1 TO 300
2970 NEXT I
2980 IF K=71 THEN 3140 ELSE 2190
2990 IF K<>71 THEN 3040
3000 GOTO 2910
3010 MSG$="YOU MADE IT!"
3020 CALL SOUND(900,1000,0)
3030 GOTO 3130
```

```
3040 IF (K=84)*(M=5)*(TRANS=1)=0 THEN 2550
3050 TRANS=0
3060 MAC(Y,X)=136
3070 Y=2*(INT(RND*11)+1)
3080 X=(2*(INT(RND*15)+1))+1
3090 MAC(Y,X)=143
3100 GOTO 2190
3110 MSG$="TOO MUCH TIME!"
3120 CALL SOUND(300,110,0)
3130 PRINT TAB(6);MSG$: : : : : : : : : :
3140 FOR I=1 TO 1000
3150 NEXT I
```

Mad Hatter Ladder

Scott Parsons

This climbing game pits you against buzz saws and flames as you leap and dash from floor to floor. Extended BASIC required; joystick optional.

The Mad Hatter is desperate. Buzz saws and flames are chasing him from one end of his two-tiered home to another. You're his only chance—you've got to help him out by pressing the right keys or moving the joystick in the right direction. Points and more points are your only reward.

Two Tiers

Starting at the lower left, the Mad Hatter glides silently from side to side when you press the right or left arrow key (or move the joystick right or left). Buzz saws, always heading left, block the Hatter's path. They can be jumped, however, by pressing the period (.) key or the joystick fire button. Be extremely careful, though, that you help the Hatter jump at just the right moment. Even the slightest touch of the buzz saw and the Mad Hatter is only a Mad Cap.

Climbing from floor to floor, using the ladders, is even more difficult. Press the up arrow key when the Hatter is in position. Unfortunately, he's a bit afraid of heights, and the only time you can persuade him to climb is when he's just a bit to the right of a ladder, with more than half his body directly under it. Hit the up arrow key or move the joystick up and, his fears gone, he pops up a floor.

Up and up you help the Hatter climb, until he reaches the top of the screen. At that moment, you're given the bonus points which have been ticking away at the bottom center of the screen. The faster you get the Hatter to the top, the more points you score. (The game ends if the bonus runs out before the Hatter makes it.)

Three chances are all you have to help the Hatter. Once he's at the top, the screen flickers into the next tier of the Hatter's home. Now there are four floors and only two ladders. The saws are gone, but flames have taken their place.

Flaming Hats

Two bolts on each floor must be removed by moving over them. That's easier than it looks, for you can "wrap around" the screen and sneak up behind the flame to steal the bolt. As in the first part of the Mad Hatter's house, ladders are provided to get to the next floor. On this screen, you can even go back down a ladder. Hatter has an even greater fear of heights here, and climbing the ladders takes more time than before. At least he's safe when he's on a ladder, for the flames move only from side to side.

A time limit ticks away—get to the top of the second tier before it runs out, and points equal to the remaining time are awarded. Then you're back to the first screen for more buzz saw fun.

Mad Hatter Ladder

```
100 CALL CHAR(98,"818181FFFF818181")
110 CALL COLOR(9,1,1)
120 CALL SCREEN(2)
130 P=5
140 CALL CHAR(128,"0E13070F1F3F2F2713000000000000000
    00C8E4F4FCF8F0E0C870")
150 CALL MAGNIFY(3)
160 LIFE=3
170 SC=0
180 E=0
190 CALL CLEAR
200 FOR I=1 TO 8 :: CALL COLOR(I,5,1):: NEXT I
210 CALL CHAR(96,"FF814242242418FF")
220 M=0
230 FOR I=1 TO 4 :: CALL HCHAR(I*6,1,96,32):: NEXT
    I
240 E=0
250 DISPLAY AT(1,7):"SCORE" :: DISPLAY AT(2,6):SC
260 DISPLAY AT(1,20):"HIGH" :: DISPLAY AT(2,19):HIG
    H
270 TIME=5000 :: DISPLAY AT(24,13)SIZE(5):TIME
280 CALL CHAR(97,"818181FFFF818181"):: FOR I=1 TO 2
     :: CALL VCHAR(18,25+I,96+I,6):: CALL VCHAR(12,
    5+I,96+I,6):: CALL VCHAR(6,25+I,96+I,6)
290 CALL VCHAR(1,5+I,96+I,5):: NEXT I
300 CALL HCHAR(1,3,96,5)
310 CALL COLOR(9,10,1)
320 FOR IR=1 TO 2
330 FOR I=1 TO 4 :: CALL SPRITE(#1+I+E,128,16,I*6*8
    -16,1,0,P):: P=-P :: NEXT I
340 IF E=4 THEN 360
```

```
350 FOR R=1 TO 1000 :: NEXT R
360 E=4
370 NEXT IR
380 CALL CHAR(132,"3F3F3F1F1F1F1F1F1F3A02020222323E
    FCFCFCF8F8F8F8F8F85C404040444C7C")
390 CALL SPRITE(#1,132,5,21*8,16)
400 REM 1st screen play
410 CALL JOYST(1,X,Y):: GOSUB 1290 :: CALL MOTION(#
    1,0,X*1.5)
420 CALL POSITION(#1,Q,W):: IF W<16 AND X=-4 THEN C
    ALL MOTION(#1,0,0):: CALL LOCATE(#1,Q,24)
430 IF W>240 AND X=4 THEN CALL MOTION(#1,0,0):: CAL
    L LOCATE(#1,Q,232)
440 CALL COINC(ALL,Z):: IF Z<0 THEN 610
450 TIME=TIME-50 :: IF TIME=0 THEN 1260 ELSE DISPLA
    Y AT(24,13)SIZE(5):TIME
460 CALL POSITION(#1,Q,W):: CALL GCHAR(Q/8,W/8,Z)::
     IF Z=97 THEN 470 ELSE 540
470 CALL JOYST(1,X,Y):: GOSUB 1290 :: CALL POSITION
    (#1,Q,W):: IF Q<48 THEN 500 ELSE IF Y=4 THEN CA
    LL LOCATE(#1,Q-48,W)
480 CALL COINC(ALL,Z):: IF Z<0 THEN 610
490 GOTO 540
500 IF Y=4 THEN CALL DELSPRITE(ALL)ELSE 540
510 FOR AQW=1 TO TIME STEP 100 :: SC=SC+100 :: DISP
    LAY AT(2,6)SIZE(5):SC :: NEXT AQW
520 IF SC>HIGH THEN HIGH=SC :: CALL CLEAR
530 GOTO 640
540 CALL KEY(1,X,Y):: GOSUB 1310 :: IF X=18 THEN 57
    0
550 CALL COINC(ALL,Z):: IF Z<0 THEN GOTO 610
560 CALL KEY(1,X,Y):: GOSUB 1310 :: IF X=18 THEN 57
    0 ELSE 600
570 CALL POSITION(#1,Q,W):: CALL LOCATE(#1,Q-16,W)
580 FOR I=1 TO 130 :: NEXT I
590 CALL POSITION(#1,Q,W):: CALL LOCATE(#1,Q+16,W)
600 CALL COINC(ALL,Z):: IF Z<0 THEN 610 ELSE 630
610 LIFE=LIFE-1 :: CALL DELSPRITE(ALL):: IF HIFE=0
    THEN 1260 :: GOTO 230
620 CALL SOUND(-1000(200,0)
630 GOTO 400
640 CALL CLEAR
650 CALL CHAR(97,"FF81424224218FF")
660 CALL CHAR(98,"818181FFFB818181")
670 AQW=-5
680 CALL CHAR(136,"31377B7F7FFFFFFFFF7F7F3F3F1F0F00
    98FCF8FCFCFFFFFFFEFEFEFEFCFCF0F0")
690 FOR I=2 TO 5 :: F(I)=5 :: NEXT I
700 CALL COLOR(10,1,1):: CALL CHAR(104,"FF7E7E7E7E7
    E7E7E7E")
```

```
710 M=0
720 TIME=5000
730 CALL CHAR(120,"3F3F3F1F1F1F1F1F3A02020222323E
    FCFCFCF8F8F8F8F85C404040444C7C")
740 CALL MAGNIFY(3)
750 CALL COLOR(9,1,1):: CALL CHAR(96,"818181FFFF818
    181"):: CALL CLEAR
760 FOR I=1 TO 4 :: CALL HCHAR(I*6,1,97,32):: NEXT
    I
770 CALL VCHAR(1,5,96,24):: CALL VCHAR(1,6,98,24)::
     CALL VCHAR(1,26,96,24):: CALL VCHAR(1,27,98,24
    )
780 FOR I=1 TO 4 :: CALL HCHAR(I*6,11,104):: CALL H
    CHAR(I*6,21,104):: NEXT I
790 CALL COLOR(10,5,1):: CALL COLOR(9,10,1)
800 DISPLAY AT(1,2):"HIGH                 SCORE"
810 DISPLAY AT(2,1):HIGH :: DISPLAY AT(2,22):SC
820 DISPLAY AT(24,12)SIZE(4):TIME
830 CALL SPRITE(#1,120,5,168,224)
840 FOR I=1 TO 4 :: CALL SPRITE(#1+I,136,7,I*6*8-16
    ,1,0,AQW):: AQW=-AQW :: NEXT I
850 CALL JOYST(1,X,Y):: GOSUB 1290 :: CALL MOTION(#
    1,0,X*1.25)
860 CALL POSITION(#1,Q,W)
870 IF W<16 OR W>240 THEN 880 ELSE CALL GCHAR(Q/8+3
    ,W/8,Z):: IF Z=104 THEN CALL SOUND(-1000,210,0)
    :: CALL HCHAR(Q/8+3,W/8,32):: GOTO 1170
880 TIME=TIME-100 :: IF TIME=0 THEN 1260
890 DISPLAY AT(24,12)SIZE(5):TIME
900 CALL COINC(ALL,Z):: IF Z<0 THEN LIFE=LIFE-1 ::
    IF LIFE=0 THEN 1260 ELSE TIME=5000-(M*400):: GO
    TO 830
910 IF ASD=1 THEN IT=IT+1 :: GOTO 920 ELSE 930
920 IF IT=1 THEN 980
930 CALL KEY(1,S,D):: GOSUB 1310 :: IF S=18 THEN 94
    0 ELSE 990
940 CALL POSITION(#1,Q,W):: CALL LOCATE(#1,Q-16,W)
950 ASD=1
960 IF YY=1 THEN YY=0 :: GOTO 1010
970 GOTO 990
980 IT=0 :: ASD=0 :: CALL POSITION(#1,Q,W):: CALL L
    OCATE(#1,Q+16,W)
990 FOR I=2 TO 5 :: CALL POSITION(#I,Q,W):: IF W<16
     THEN CALL MOTION(#I,0,5):: F(I)=-F(I)
1000 CALL JOYST(1,X,Y):: GOSUB 1290 :: CALL MOTION(
     #1,0,X*1.25)
1010 CALL POSITION(#1,Q,W)
1020 IF W<16 OR W>250 THEN 1030 ELSE CALL GCHAR(Q/8
     +3,W/8,Z):: IF Z=104 THEN CALL SOUND(-1000,310
     ,2):: CALL HCHAR(Q/8+3,W/8,32):: YY=1 :: GOTO
     1170
```

```
1030 IF ASD=1 THEN 1070 ELSE CALL KEY(1,S,D):: IF S
     =18 THEN YY=1 :: GOTO 940
1040 CALL COINC(ALL,Z):: IF Z<0 THEN LIFE=LIFE-1 ::
      IF LIFE=0 THEN 1260 ELSE TIME=5000-(M*400)::
     GOTO 830
1050 CALL JOYST(1,X,Y):: GOSUB 1290 :: IF X=0 AND Y
     =0 THEN CALL MOTION(#1,0,X*1.5)
1060 YY=0
1070 IF W>232 THEN CALL MOTION(#I,0,-5):: F(I)=-F(I
     )
1080 IF W>8 AND W<232 THEN 1090 ELSE 1100
1090 CALL GCHAR(Q/8+2,W/8,Z):: IF Z=32 THEN F(I)=-F
     (I):: CALL MOTION(#I,0,F(I))
1100 NEXT I
1110 CALL POSITION(#1,Q,W):: IF Q<28 THEN 1130
1120 IF Y=4 THEN CALL POSITION(#1,Q,W):: CALL GCHAR
     (Q/8,W/8,Z):: IF Z=96 THEN CALL LOCATE(#1,Q-48
     ,W)
1130 IF Q>8*20 THEN 1150
1140 IF Y=-4 THEN CALL POSITION(#1,Q,W):: CALL GCHA
     R(Q/8,W/8,Z):: IF Z=96 THEN CALL LOCATE(#1,Q+4
     8,W)
1150 CALL POSITION(#1,Q,W)
1160 IF W>250 OR W<16 THEN 850 ELSE CALL GCHAR(Q/8+
     3,INT(W/8+.5),Z):: IF Z=104 THEN CALL SOUND(-1
     000,110,1):: CALL HCHAR(Q/8+3,W/8,32)ELSE 1230
1170 SC=SC+20
1180 DISPLAY AT(2,22):SC
1190 M=M+1 :: IF M=8 THEN 1200 ELSE IF YY=1 THEN 10
     60 ELSE 850
1200 CALL DELSPRITE(ALL):: FOR T=1 TO TIME STEP 100
1210 CALL SOUND(200,563,0):: CALL SOUND(200,282,0):
     : SC=SC+100 :: DISPLAY AT(2,22):SC :: NEXT T :
     : GOTO 190
1220 DISPLAY AT(2,22):SC
1230 GOTO 850
1240 IF SC>HIGH THEN HIGH=SC :: DISPLAY AT(2,1)SIZE
     (6):HIGH
1250 GOTO 210
1260 IF SC>HIGH THEN HIGH=SC
1270 CALL SCREEN(16):: CALL DELSPRITE(ALL)
1280 DISPLAY AT(10,6)SIZE(17):"PLAY AGAIN (Y/N)" ::
      CALL KEY(0,Q,W):: IF Q=121 OR Q=89 THEN 110 E
     LSE IF Q=110 OR Q=78 THEN END ELSE 1280
1290 CALL KEY(0,B,C):: IF B=83 THEN X=-4 ELSE IF B=
     68 THEN X=4 ELSE IF B=69 THEN Y=4 ELSE IF B=88
      THEN Y=-4
1300 RETURN
1310 CALL KEY(0,B,C):: IF C=-1 THEN RETURN ELSE IF
     B=46 THEN X=18 :: S=18
1320 RETURN
```

3

Inside the TI

TI Memory Organization

William S. Miller

Whether you're an advanced programmer or just starting out, you can profit from knowing how your computer is put together. This tutorial describes the unique, rarely documented memory organization of the TI-99/4A and includes two memory maps.

The TI-99/4A computer is a powerful machine, and there's more inside its console than most documentation shows. Surprisingly, though the TI's memory organization is unlike that of any other personal computer, very little has been published on the subject. Let's explore the major features of TI architecture and see how the computer's internal structure affects its performance.

Addressing Memory

Before looking at specifics, let's review how computers handle memory. Computer memory is simply a collection of individual *memory locations,* or addresses. Each location can hold a *byte,* a value in the range 0–255. Memory is commonly measured in units of *kilobytes* (1024 bytes), abbreviated as K. Thus, 8K of memory is actually 8192 (8*1024) bytes.

At the most fundamental level, a computer can perform only two simple memory operations: It can *read* a memory location (see what number is stored there) or *write* to a location (store a new value). There's a limit to the number of locations a computer can address, determined by the addressing capability of its microprocessor chip. The TI's TMS9900 microprocessor can address a maximum of 64K (65,536) locations. Thus, the lowest memory address the TI can refer to is location 0, and the highest is location 65,535.

Though the computer can read or write to as many as 64K locations, memory chips may not be present in all those places, so the *usable* memory in the computer may be much less than 64K. When a memory chip is present, it may contain either Read Only Memory (ROM) or Random Access Memory (RAM). A ROM chip contains information in permanent form; the computer can read ROM memory, but can't write to it. RAM is programmable memory; the computer can write to a

RAM location as well as read its contents. The numbers stored in ROM are retained when you shut down the computer, but RAM is *volatile*—as soon as you turn off the power, its contents are lost.

Most computers also have a few locations that aren't exactly ROM or RAM. These are *control* addresses, used to control input/output chips. By writing to or reading such locations, the computer controls graphics displays, disk storage, and similar functions.

You may have heard that the TI-99/4A has a 16-bit microprocessor rather than the 8-bit processor found in most home computers. In fact, the TI actually sees its 64K address space as 32,768 two-byte *words* of memory rather than as 65,536 bytes. Since the microprocessor can manipulate a word either as an entire 16-bit package or as two 8-bit packages, it combines some capabilities of 16-bit and 8-bit processors. (Most other 16-bit processors, however, can address considerably more than 64K of RAM.)

TI Memory Map

A *memory map* is simply a chart of a computer's memory organization showing the location of ROM and RAM. Figure 1 is a general memory map for the TI-99/4A.

Several questions may spring to mind when you scan the map. For instance, where's the 16K RAM that's supposed to be available in an unexpanded machine? What is a *GPL interpreter* and what is *GROM?* Only 8K of internal ROM is apparent—where is the advertised 26K ROM? Since only 8K of memory space is assigned to the module port, how is it possible for modules to hold as much as 36K of program information? Finally, how can all of the peripheral ROMs occupy the same 8K space? After all, they're frequently used at the same time.

To track down the missing RAM, we'll need to review some hardware. In addition to its main microprocessor, the TI has a second microprocessor devoted exclusively to graphics. The TMS9918A Video Display Processor (VDP) controls sprites, high-resolution displays, and other graphics features, and can address a maximum of 16K of RAM. Since this special RAM area can be accessed only by the VDP (hence the term *VDP RAM*), it's not included in the general memory map in Figure 1.

Figure 1. TI-99/4A General Memory Map

$FFFF		
	High memory expansion	8K
	High memory expansion	8K
	High memory expansion	8K
$A000		
	Memory-mapped I/O for speech synthesizer, YDP, GROM, sound chip. CPU scratchpad RAM at $8300.	8K
$8000		
	8K available at module port	8K
$6000		
	ROMs for peripheral devices—up to 11 peripherals such as disk controller, RS-232 interface, etc.	8K
$4000		
	Low 8K portion of 32K Expansion RAM	8K
$2000		
	Console ROM, contains part of operating system, GPL interpreter, part of TI BASIC.	8K
$0000		

The main microprocessor can access VDP RAM only in a secondhand way, by writing to or reading special addresses in the VDP chip itself. The VDP then writes to or fetches data from VDP RAM. But VDP RAM is used for nongraphics purposes as well. When you use the built-in TI BASIC, your program is stored entirely in VDP RAM. TI Extended BASIC (a 36K module) uses VDP RAM for the same purpose when no expansion RAM is present. If the 32K RAM Memory Expansion unit is installed, TI Extended BASIC uses 24K of expansion RAM to store the program and numeric variables, about 14K of VDP RAM to store string variables, and the remaining 8K of expansion RAM to store machine language routines (accessed with the CALL statement). When a disk drive is hooked up, the disk operating system also uses 2K of VDP RAM as a data buffer area.

Incidentally, the TI-99/4A is not the only computer to use the VDP chip. Spectravideo, Coleco, and all MSX computers employ the VDP, and it has also been marketed as part of a sprite board for the Apple II with Logo.

A Hidden Interpreter

As you may know, a computer can't directly understand BASIC words like PRINT and GOTO. The BASIC language is actually a large machine language program that *interprets* (decodes) BASIC commands, translating each statement into machine language instructions which the computer can perform directly. A program that translates commands from one language to another is called, not surprisingly, an *interpreter.* Since the translating takes time, interpreted programs always run slower than machine language programs.

In addition to the BASIC interpreter, the TI has a Graphics Programming Language (GPL) interpreter. This is a special-purpose machine language program (4K in length, stored in the 8K console ROM) which interprets programs stored in a special kind of ROM known as Graphics Read Only Memory (GROM). Since GROM space is not directly addressable, you can't use it to store machine language routines (or anything else, for that matter). The TI may use up to eight GROMs, each 6K in size. A typical module contains three resident GROMs along with one 8K ROM.

GPL routines are ideal for graphics and are quite fast, though comparable machine language routines seem to run from two to ten times faster. But don't get excited about writing your own GPL routines. As far as I know, GPL programs *must* reside in GROM or GRAM (Graphics RAM), and the main processor's address space contains only a few locations for communicating between the GPL interpreter and GROM. Even if you learn how to program in GPL (no small feat, since

TI provides no documentation), you have no way to put such programs in GROM where the GPL interpreter can run them. For all practical purposes, GPL and GROM are usable only by TI's own programmers.

In any case, we've answered some of our questions. The missing console ROM is actually 18K of GROM, and the extra module ROM is up to 30K of GROM. The 18K of console GROM contains parts of the operating system like the cassette service routine, as well as most of the TI BASIC interpreter.

That's bad news for BASIC programmers (see below), but it's ideal for module-based software. Modules have 8K of directly addressable ROM for speed-intensive routines and up to 30K of GROM for slightly less speedy GPL routines. Very large and elaborate module-based programs may be installed without sacrificing any of the user RAM that's normally available. Since the system is so flexible, some modules provide a bit of everything. The Mini Memory module, for instance, provides 4K of RAM (with battery backup to retain data when unplugged), 4K of addressable ROM, and 6K of GROM, including utilities for machine language programming.

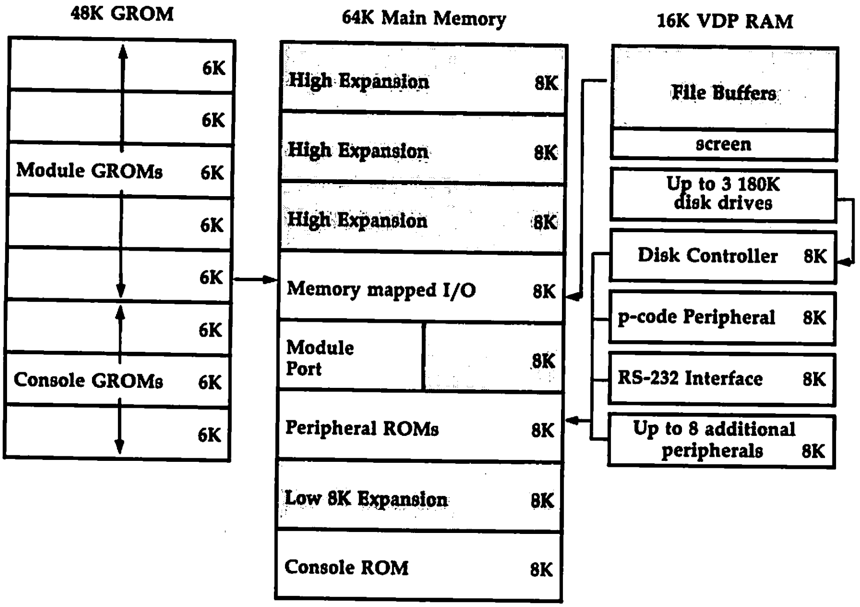
As the memory map shows, 8K of the main memory space is occupied by peripheral ROMs. The main processor has extremely flexible input/output capabilities and can actually assign several 8K peripheral ROMs to the same 8K address space without stealing any of the 48K user RAM. Each peripheral is controlled with separate instructions, so the system never becomes confused. Up to 11 peripherals, each with its own ROM, can be connected to the TI, giving the computer unusual expansion capabilities. The TI Peripheral Expansion System provides an eight-slot parallel expansion bus and auxiliary power supply to take advantage of this flexibility.

Expanded Memory Map

Compared with some other systems, the TI-99/4A has a relatively open architecture. Many different configurations (and capabilities) are possible, depending on what you plug into the system. Figure 2, an expanded TI memory map, illustrates this flexibility.

Like virtually every other computer, the TI does have some flaws. First, the unexpanded machine has no directly accessible user RAM (just VDP RAM, which can be accessed only secondhand through the VDP). Since there's no way to load and run machine language programs in RAM, machine language programming is impossible on an unexpanded system. This also explains why PEEK and POKE were omitted from TI BASIC: Since there's no memory you can access directly, the commands are meaningless. *(Note: The exception to all the machine language programming limitation is that you* can *create relatively primitive machine language programs on an unexpanded TI, as long as you have the Mini Memory module.)*

Figure 2. TI-99/4A Expanded Memory Map



Note: Shaded areas are possible user RAM.

The consequences for BASIC are even more serious. Since BASIC programs are stored in VDP RAM, every BASIC statement must be shuffled through two microprocessors instead of one. Remember, the main processor can't access VDP RAM directly. It can only send requests to the VDP, a second processor, which in turn accesses the memory. Worse still, the TI BASIC interpreter itself resides mostly in GPL-interpreted GROM, not in memory accessible to the main processor. As a result, every BASIC statement must be interpreted twice—

once by the GPL interpreter and again by the BASIC interpreter.

So much for the bad news. TI BASIC *is* slow—not as slow as its LIST and PRINT commands might suggest, but still slower than other personal computer BASICs. However, TI Extended BASIC is much faster than TI BASIC, and the system's flexibility partly compensates for its slowness.

The Heart of the TI-99/4A

The TMS9900 Microprocessor

William S. Miller

The TI-99/4A's microprocessor—the TMS9900—is a powerful chip. With the right software tools, such as the Editor/Assembler *or the Mini Memory command module, you can jump into assembly language programming. First, though, you need some background on the 9900, and the way it drives the TI-99/4A.*

Controlling the flow of information within any computer system is the central processing unit (CPU). Whether the computer is a huge mainframe, expensive mini, or personal micro, the CPU is responsible for moving information to and from memory locations (places where the machine stores its instructions and data), for transforming this information using built-in sequences of arithmetic and logical operations, and for transmitting this information to and from external devices for storage or communication with the outside world. The CPU within your TI-99/4A is a small microprocessor, the TMS9900.

These arithmetic, logical, and transfer operations make up the smallest software building blocks available to a programmer. Each operation has a corresponding code expressed as a group of ones and zeros, referred to as its *opcode*. As the microprocessor steps through memory, it recognizes these codes and executes the appropriate operations.

The set of available operations and their corresponding codes is called the *instruction set* of the CPU and comprises the *machine language* for that particular computer. Rather than fetch these codes or manipulate stored data one bit at a time along a single physical line (serially), all microcomputer systems transfer 8 or 16 bits simultaneously along 8 or 16 lines (in parallel). The microprocessor itself is capable of digesting data in corresponding 8- or 16-bit gulps. The width of the data path within the microprocessor, whether 8 or 16 bits, is what describes it as an *8-bit* or *16-bit* machine. The TI-99/4A's TMS9900 microprocessor transfers and manipulates data 16 bits at a time.

Some Important Facts

The TMS9900 microprocessor was developed in the mid-1970s by Texas Instruments as a single-chip implementation of the CPU architecture and instruction set of their 990 minicomputer series. The 9900 is a single large-scale integration integrated circuit in a big 3.2 × 0.9 inch 64-pin package. This pin set includes 16 data and 16 address lines—using 64 pins rather than the usual 40 permits each of these data and address lines to have a direct connection to system memory, enabling full 16-bit data and instruction transfers to and from the processor.

The power supply levels required by the 9900 are +5, −5, and +12 volts. The 9900 also requires timing pulses in a cycle of four phases from an external clock for the control and coordination of the processor's internal operation. In the TI-99/4A, this four-phase cycle is repeated 3,000,000 times per second for a clock speed of 3 megahertz. Finally, the 9900 must have access to at least 256 bytes of external RAM on which to base its unique memory-based register files. Take a look at Figure 1, which shows the TMS9900 with its pin assignments labeled.

The TMS9900 microprocessor dominates the TI-99/4A's console circuit board, as you can see from Figure 2. (Make a note of the placement of the 9901 programmable system interface chip, to the left of the the 9900. We'll discuss some of the 9901's features a bit later.)

In several aspects, the TMS9900 reflects innovative and unique design. This article looks at these four design aspects of the 9900 microprocessor:

- Its unusual register files
- Its instruction set
- Its flexible serial interface
- Its powerful interrupt structure

We'll also briefly outline the features of the *Editor/ Assembler* package from Texas Instruments—this powerful software and command module lets you utilize the advanced capabilities of the TMS9900 microprocessor.

Figure 1. TMS9900 Pin Assignments

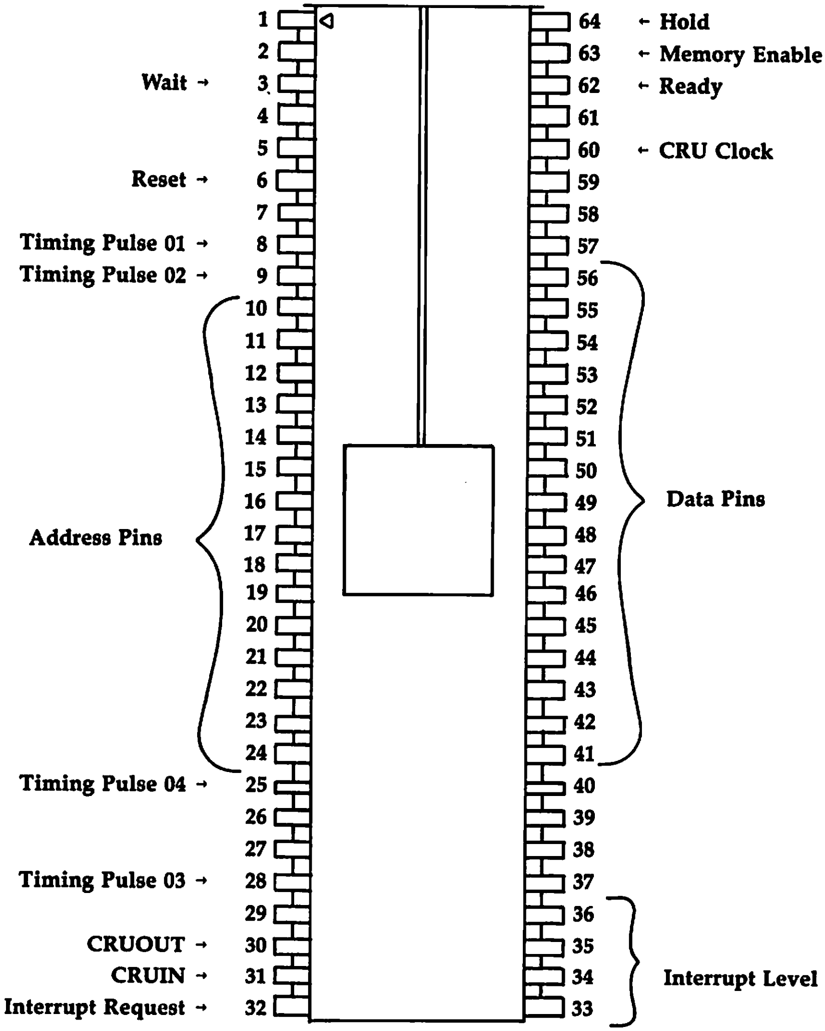
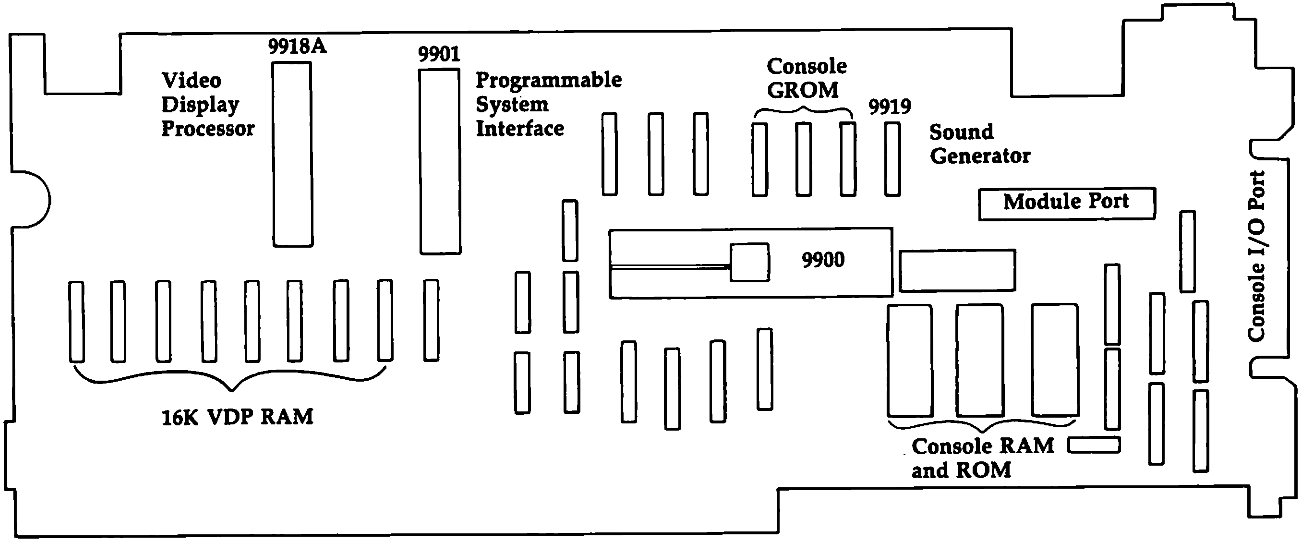


Figure 2. TI-99/4A Console Board Schematic



Registers on the TMS9900

The most unusual aspect of the 9900's design is its register sets. All microprocessors use a special set of memory locations into which required data and results from each machine instruction are placed. Two other locations are also needed—the *program counter,* for storing and incrementing the RAM address containing the next machine instruction, and the *status register,* which maintains a record of any special conditions that may have occurred during each instruction's execution.

These registers are typically built into the microprocessor chip itself, an arrangement that allows for very fast access to register contents, but limits the number of registers to between 3 and 16. Because of this limitation, whenever the microprocessor encounters an instruction causing a temporary branch to a subroutine (which in turn requires a new set of register contents), the programmer must provide a series of instructions to store the contents of these on-board registers in memory and load into the registers the values appropriate to the start of the new subroutine environment. When execution returns to the calling program, the new register contents must be saved and the old contents reloaded. This register saving and reloading can occupy almost 50 percent of the microprocessor's time when it's involved in a complex application, such as when using multiple subroutines and interrupts.

To avoid this, TI has taken an unusual approach to establishing register files on the TMS9900. The chip actually provides only three on-board registers—the program counter, status register, and a third, the *workspace pointer.* The latter is unique to the 9900 microprocessor family.

Rather than providing general-purpose registers on the chip, the 9900 establishes a set of 16 registers in consecutive RAM locations. The contents of the workspace pointer tell the microprocessor the address of the first of these 16 two-byte (or one-*word*) registers. As many register sets as desired may be set up in RAM. Each set is referred to as a *workspace* and contains registers numbered from R0 to R15. A new workspace may be used at any time simply by changing the contents of the workspace pointer. A workspace may be placed in any available RAM, even overlap with other workspaces.

It's easier to see this efficiency when the programmer is faced with a situation involving multiple interrupts and subroutine calls. Rather than saving and loading register contents before a branch or return, the programmer selects an instruction that executes what Texas Instruments calls a *context switch*. This single instruction performs several tasks before jumping from the current sequence of instructions to the sequence called for by the branch operation.

- First, the current program counter and workspace pointer values are saved in temporary storage locations.
- New values appropriate to the subroutine are then loaded into the program counter and workspace pointer.
- Finally, the old workspace pointer, old program counter, and status registers are saved to registers R13, R14, and R15 of the new workspace, effectively saving the state of the microprocessor at the moment prior to the jump.
- The microprocessor begins to execute instructions using the new workspace as its register set.
- A simple return instruction restores the old register values and workspace, enabling the microprocessor to continue with the calling program sequence.

This flexible register capability results in easier assembly language programming and faster program execution, and is unique to the 9900 family of microprocessors.

The 9900 Instruction Set

The Branch and Load Workspace Pointer (BLWP) and Return with Workspace Pointer (RWP) instructions described above are good examples of the convenience and speed provided by a well-designed instruction set. The TMS9900 offers 69 instructions using six addressing modes. These instructions fall into several categories (the numbers in parentheses indicate how many instructions exist in each category):

- Arithmetic instructions (13)
- Jump and Branch instructions (19)
- Compare instructions (5)
- Control and CRU instructions (10)
- Load and Move instructions (8)
- Logical instructions (10)
- Workspace Register Shift instructions (4)

The advanced assembly language features offered to the TI-99/4A assembly language programmer include the following.

Memory to memory operations. Many operations may be performed directly on the contents of memory locations in

RAM. For example, the ADD Words instruction may specify two RAM addresses containing *operands* (data to be manipulated). The operand found in the first address will be added to that in the second, and the sum left in the second address. There's no need to load registers. In contrast, virtually all 8-bit microprocessors require that a value be loaded into a specialized one-chip register known as the *accumulator* before an arithmetic operation may be performed using that value. The LDA (LoaD Accumulator) instruction is probably the most frequently used instruction of the 6502 microprocessor. On the 9900, any register or memory location may be used as an accumulator.

Byte and word operations. The 9900 may operate upon any 16-bit *word* (in other words, two 8-bit bytes) of memory, or on either byte of that word. In fact, the 9900 sees memory as 32,768 words rather than as 65,536 bytes. All operations have access to or transfer an entire word of data whether or not a single byte is manipulated.

Flexible addressing modes. *Addressing modes* refers to the strategies that you may select to specify where the operands required by an operation are to be found. For example, any register may contain an operand, the address where the operand may be found, or a value to which a constant is added to find the operand. The addressing modes available to the 9900 assembly language programmer include Workspace Register Addressing, Workspace Register Indirect Addressing, Workspace Register Indirect Auto Increment Addressing, Symbolic Memory Addressing, Indexed Memory Addressing, and Immediate Addressing. Most instructions may use any of these addressing modes in any combination—where appropriate.

Hardware multiply and divide. The TMS9900 can multiply two unsigned 16-bit values, producing a 32-bit integer product, or divide a 32-bit value by a 16-bit value, producing a 16-bit quotient, each with a single instruction.

The extended operation. The extended operation instruction permits the addition of up to 16 software designed instructions to the instruction set of the 9900, an advanced feature supported by the more recent TI-99/4A operating system, though not by the TI-99/4 or early TI-99/4As.

The Serial Interface and Interrupt Structure

Both of these functions are conducted with the help of a single auxiliary chip in the TI-99/4A, the 9901 programmable systems interface. Thus far, we've just talked about parallel data transfers, where entire 16-bit words are moved at once along the 16 data lines to or from the locations in the address space specified by signals present on the address bus. As can many microprocessors, the 9900 is able to send and receive information by means other than these data lines. *Serial data* may be moved into and out of the processor over two lines dedicated to the purpose, and the microprocessor may respond to combinations of signals placed on five lines dedicated to the receipt of *interrupts.*

Serial data consists of bits sent or received one at a time, while an interrupt does just that—it interrupts the transfer. When an *interrupt request* is received that the microprocessor has been programmed to acknowledge, it stops what it's doing, saves its state, branches through a context switch using parameters specified by the operating system, and performs the task called by the interrupt. When the job's done, the microprocessor state returns to normal.

While many processors offer a line for serial data I/O, the TMS9900 is capable of unusually flexible serial data transfers. This serial interface capability, along with any external support chips present which utilize this capability, is referred to as the Communications Register Unit, or CRU. This serial capacity is unusual in that it provides serial data with 4K bits of address space to specify sources and destinations for serial data. When an instruction is executed that sends or receives serial data along the two dedicated lines, the processor also places a 12-bit address specification on the address bus, determining one of 4096 sources or destinations for the data. This means that, in addition to the 64K byte address space available to the data lines, there's an additional serial address space with 4K of individually distinguishable single-bit addresses available as either source or destination for serial data.

In the TI-99/4A, the serial data lines are connected to the 9901 programmable systems interface, which receives information from 5 of the 12 address lines used to specify destinations for serial data and receives a *chip enable* signal to let the 9901 know when address information is intended for it. In turn, the 9901 provides 22 lines to the outside world that may be programmed for a variety of I/O tasks. When the appropriate address information and chip enable signal are sent, accompanied by from 1 to 16 bits of serial data, the 9901 interprets that data as destined for its registers used to program data transfer characteristics. In this way, you can initialize the 9901 to provide up to 16 of its 22 system lines for the receipt and disbursement of serial data to a number of sources and destinations, as well as program the 9901's flexible interrupt control capabilities.

The TMS9900 microprocessor provides several CRU instructions which can be used to control this serial interface and to transfer, test, and modify serial data as it's sent or received. In the TI-99/4A, the 9901 (as CRU) is used to address console devices such as the keyboard, joysticks, and cassette interface, as well as to address and activate external devices such as the disk controller, four ports of the RS-232 interface, the thermal printer, the p-code peripheral, and the appropriate bank-switched peripheral device ROM and RAM. A number of blocks of CRU address space have been reserved in the TI-99/4A operating system for future expansion.

The **interrupt** capabilities of the 9900 are just as flexible, and these too are significantly enhanced by the 9901. As already mentioned, the 9900 uses five lines to receive interrupts. One line is an *interrupt request line,* and is used whenever the processor is to be interrupted. The other four are used to specify the *level* of the interrupt request being made, an indication of the request's urgency. Using four lines gives the processor 16 levels of interrupt. To give you control over which interrupt levels will be acknowledged, the last 4 bits of the 16-bit status register may be used to select the levels of interrupt using the Load Interrupt Mask Immediate (LIMI) instruction. An LIMI 2 instruction *enables* interrupt levels 0, 1, and 2—all other interrupts will be ignored. The 9901 further augments this interrupt capability by providing up to 16 separate interrupt lines and lets you more selectively mask a few or even a single one of these lines while leaving others (both higher and lower priorities) unmasked. The improved operating system of the TI-99/4A provides interrupt handling routines that take advantage of this flexible structure.

The *Editor/Assembler* Package

The *Editor/Assembler* software and hardware from Texas Instruments is a powerful development environment, one derived from their 990 minicomputers, and provides features unusual for a microcomputer assembler. Just some of the package's features are listed here.

80-column Editor. The Editor employs three overlapping 40-column windows onto the 80-column text, which may be scrolled horizontally or vertically to view all source text. Powerful editing features include insertion and deletion by character, word, or line; find word; find and replace words; move block; copy block; file insert from disk; and list source to any device.

The Assembler. This produces relocatable, linkable code. Object code produced by the Assembler may contain unresolved references that are resolved by the loader provided during load from disk, permitting the loader to place programs anywhere in available RAM. External References and Definitions let you build a library of program modules of relocatable code that will be automatically loaded from disk and linked to a new program which contains the appropriate references as the new program is assembled. All external references and definitions are resolved by the loader. This also permits assembly language programs to be loaded and called by name rather than address from the Extended BASIC, TI BASIC, and *Editor/Assembler* environments. Parameters may be passed at this time, too.

System utilities. Utilities residing either in console ROM or the *Editor/Assembler* software may be used simply by loading registers with the required parameters and calling the routine. The Extended BASIC environment duplicates these utilities for assembly programs run from this BASIC. Utilities include write or read single or multiple bytes from VDP RAM or VDP registers; keyboard scan; link to GROM routines; link to ROM or RAM assembly routines; link to floating-point routines; PEEK, POKE, PEEKV, and POKEV from TI-BASIC; link to assembly routines from BASIC by name (rather than by address); get or modify BASIC numeric, string, or array variables for assembly language program use; and link to error routines.

TI Bug Object Code Debugger. Once the object code is produced, it can be debugged by setting breakpoints; inspecting or changing CPU, VDP, and GROM memory parameters; inspecting and changing CRU bits; finding words, bytes, and blocks; moving blocks; converting from decimal to hexadecimal or the reverse; and performing hexadecimal arithmetic. The optional Advanced Assembly Debugger is also available to further enhance these debugging tools.

The *Editor/Assembler* manual. A very complete, well-written guide to 9900 assembly language and the TI-99/4A architecture in general.

TI Character Graphics

Rick Rothstein

Printing redefined characters within strings is a fast and efficient method for creating custom graphics. By using undefined character codes, you can add custom characters without sacrificing any part of the standard character set. The technique works with either TI BASIC or Extended BASIC.

Many TI owners know that the CALL CHAR statement redefines graphic characters, and that these new characters can be assigned to any character code from 32 to 159 (only characters 32–143 in Extended BASIC). Once graphic characters have been redefined, they're often placed on the screen—one at a time—with CALL HCHAR or CALL VCHAR statements. But it's much more efficient to print them on the screen as strings.

Let's draw a smiling face on the screen to illustrate the basic technique. Enter and run Program 1. Lines 160–190 redefine the first 25 characters of the alphabet as graphic shapes, then lines 200–240 print these custom characters on the screen as strings. (If you have Extended BASIC, this same technique works with the DISPLAY AT statement.) Lines 1000–1050 contain the character redefinition data.

Program 1. Smiling Face

```
100 CALL CLEAR
110 CALL SCREEN(13)
120 CALL COLOR(5,11,1)
130 CALL COLOR(6,11,1)
140 CALL COLOR(7,2,11)
150 CALL COLOR(8,11,1)
160 FOR N=65 TO 89
170 READ CODE$
180 CALL CHAR(N,CODE$)
190 NEXT N
200 PRINT TAB(13);"ABCDE"
210 PRINT TAB(13);"FRSTG"
220 PRINT TAB(13);"HUIVJ"
230 PRINT TAB(13);"KWPQL"
240 PRINT TAB(13);"MNOXY": : : : : : : : :
250 GOTO 250
```

```
1000 DATA 0000000000000001,00000071F7FFFFF,0000FFF
     FFFFFFFFF,000000E0F8FEFFFF
1010 DATA 0000000000000080,0307070F0F1F1F1F,C0E0E0F
     0F0F8F8F8,3F3F3F3F3F3F3F3F
1020 DATA FFFFFFFFFFFFFFFF,FCFCFCFCFCFCFCFC,1F1F1F0
     F0F070703,F8F8F8F0F0E0E0C0
1030 DATA 0100000000000000,FFFF7F1F07000000,FFFFFFF
     FFFFF0000,000000C3FF7E0000
1040 DATA 3860C08000000000,0000000E1F3F3F3F,0000000
     000818181,00000070F8FCFCFC
1050 DATA 1F0E000000000206,F870000000004060,1C06030
     100000000,FFFFFEF8E0000000,8000000000000000
```

The disadvantage of redefining the standard characters is that the commonly used letters and numbers are no longer available. Since alphanumeric characters occupy the first 126 ASCII character codes, the usual solution is to put custom graphics in characters with codes above 126.

Undefined Character Codes

Characters with ASCII codes over 126 are unlike the alphanumerics. Since BASIC doesn't use them, they're considered *undefined,* and ordinarily are blanks. But you can still print them on the screen, usually with the CHR$ function. For instance, after characters 127, 128, and 129 are redefined, they can be printed with the statement PRINT CHR$(127); CHR$(128); CHR$ (129). Of course, if these characters have not been redefined, this statement simply prints three blanks.

Although the CHR$ function is fast, it's more cumbersome and less memory-efficient than printing an ordinary string enclosed in quotation marks (like PRINT "ABC"). Fortunately—though you may never have seen this fact documented—it's not necessary to use the CHR$ method. All the character codes above 126 can be generated directly from the keyboard by pressing the CTRL or FCTN keys along with alphanumeric keys. This allows you to enclose the undefined characters within quotation marks, just like ordinary alphanumerics.

Let's see how this is done. Press FCTN-= (QUIT) and reselect either TI BASIC or TI Extended BASIC (you'll see why this is necessary in a moment). Enter the following line in immediate mode (with no line number), *holding down the CTRL key* as you type the *A* and *B*:

PRINT *"AB"*

On the screen it will look as though nothing was printed. Actually, characters 129 and 130 were printed next to each other, but since neither character has been redefined, both are blanks. Enter the following program *without* erasing the spot where you printed the two special characters:

```
10 CALL CHAR(129,"0044447C44444400")
20 CALL CHAR(130,"0038101010103800")
```

When you run this program, watch what happens at the spot where the two undefined characters were printed earlier. The custom characters appear as soon as the program redefines character codes 129 and 130. Enter NEW and type CTRL-A (hold down the CTRL key and press A). Since you've redefined character 129, it's no longer blank.

Clearing Redefined Characters

There are only four ways to clear graphic definitions from characters with codes of 127 or higher: Press the QUIT key (FCTN-=), enter the BYE command, enter a new definition with a CALL CHAR statement, or shut off the computer.

Once again, press the QUIT key and reselect TI BASIC (do *not* select TI Extended BASIC if it's available). Now type in Program 1 again, substituting the following line:

```
160 FOR N=129 TO 153
```

When you enter lines 200–240, press the CTRL key when entering the letters between the quotation marks (remember, the undefined characters will look like blanks). When you run the program, it draws the same smiling face, using character codes which were previously thought unprintable without the CHR$ function. More significantly, while the first version of the program eliminated the first 25 characters of the alphabet, this version retains all the standard alphanumerics.

Table 1 shows all the character codes which can be redefined and printed on the screen, along with key combinations that generate them. For example, CTRL-A generates character code 129—to get this character, hold down the CTRL key and press A. Note that character code 127 is generated by the FCTN key rather than the CTRL key. Also, remember that only those characters with codes of 143 or less are available with Extended BASIC.

Table 1. Redefinable CTRL/FCTN Characters

Code	Keys	Code	Keys
127	FCTN-V	144	CTRL-P
128	CTRL-,	145	CTRL-Q
129	CTRL-A	146	CTRL-R
130	CTRL-B	147	CTRL-S
131	CTRL-C	148	CTRL-T
132	CTRL-D	149	CTRL-U
133	CTRL-E	150	CTRL-V
134	CTRL-F	151	CTRL-W
135	CTRL-G	152	CTRL-X
136	CTRL-H	153	CTRL-Y
137	CTRL-I	154	CTRL-Z
138	CTRL-J	155	CTRL-.
139	CTRL-K	156	CTRL-;
140	CTRL-L	157	CTRL-=
141	CTRL-M	158	CTRL-8
142	CTRL-N	159	CTRL-9
143	CTRL-O		

Character codes 127–159 are not the only ones which can be printed on the screen with CTRL or FCTN in standard BASIC. Table 2 lists the remaining undefined codes. The CALL CHAR statement, however, cannot redefine these characters. Hence, they can be printed on the screen only as blanks. Short of printing secret (invisible) codes on the screen, which can be read only by the CALL GCHAR statement, it's difficult to imagine a practical use for these codes. None of them can be used in Extended BASIC.

Table 2. Nonredefinable CTRL/FCTN Characters

Codes	Keys	Codes	Keys
176	CTRL-0	188	FCTN-0
177	CTRL-1	189	FCTN-;
178	CTRL-2	190	FCTN-B
179	CTRL-3	191	FCTN-H
180	CTRL-4	192	FCTN-J
181	CTRL-5	193	FCTN-K
182	CTRL-6	194	FCTN-L
183	CTRL-7	195	FCTN-M
184	FCTN-,	196	FCTN-N
185	FCTN-.	197	FCTN-Q
186	FCTN-/	198	FCTN-Y
187	CTRL-/		

Putting Custom Characters to Work

As you've already seen, it's not easy to type or read characters that all look blank. Program 2 (for regular BASIC) or Program 3 (for Extended BASIC) will help you incorporate these special characters into your own programs. Each program simply redefines the characters above 126 to make them display in reverse video (character 127 is displayed as the FCTN symbol to remind you to type it with FCTN instead of CTRL). Run the program before loading or typing in your own program. You'll find it much easier to enter the special characters and read your listings on the screen.

Program 2. Visible Characters for TI BASIC

```
100 CALL CHAR(127,"007040604A4A0A04")
110 FOR N=128 TO 159
120 READ A$
130 CALL CHAR(N,"FF"&A$)
140 NEXT N
1000 DATA FFFFFFFFCFEFDF,C7BBBB83BBBBBB,87DBDBC7DBD
     B87,C7BBBFBFBFBBC7
1010 DATA 87DBDBDBDBDB87,83BFBF87BFBF83,83BFBF87BFB
     FBF,C3BFBFA3BBBBC7
1020 DATA BBBBBB83BBBBBB,C7EFEFEFEFEFC7,FBFBFBFBFBB
     BC7,BBB7AF9FAFB7BB
1030 DATA BFBFBFBFBFBF83,BB93ABABBBBBBB,BB9B9BABB3B
     3BB,83BBBBBBBBBB83
1040 DATA 87BBBB87BFBFBF,C7BBBBBBABB7CB,87BBBB87AFB
     FBB,C7BBBFC7FBBBC7
1050 DATA 83EFEFEFEFEFEF,BBBBBBBBBBBBC7,BBBBBBD7D7E
     FEF,BBBBBBABABABD7
1060 DATA BBBBD7EFD7BBBB,BBBBD7EFEFEFEF,83FBF7EFDFB
     F83,FFFFFFFFFFCFCF
1070 DATA FFCFCFFFCFEFDF,FFFF83FF83FFFF,C7BBBBC7BBB
     BC7,C7BBBBC3FBF7CF
```

Program 3. Visible Characters for Extended BASIC

```
100 CALL CHAR(127,"007040604A4A0A04",128,"FFFFFFFFF
    FCFEFDF")
110 HEX$="0123456789ABCDEF"
120 FOR J=129 TO 143
130 CALL CHARPAT(J-64,A$)
140 B$=""
150 FOR K=1 TO 16
160 B$=B$&SEG$(HEX$,17-POS(HEX$,SEG$(A$,K,1),1),1)
170 NEXT K
180 CALL CHAR(J,B$)
190 NEXT J
```

Programs 2 and 3 make screen listings comprehensible, but it's impossible to do the same for printed listings. Printers interpret these special codes as true control codes, performing linefeeds, backspaces, tabs, or whatever, rather than printing the redefined characters.

Printing graphic characters on the screen is fast and memory-efficient. However, it does have one drawback. The PRINT statement (and DISPLAY AT statement in TI Extended BASIC) can put graphic characters into only 28 of the 32 available screen columns. Hence, any part of a picture which occupies columns 1, 2, 31, or 32 must still be put on the screen with a CALL HCHAR or CALL VCHAR statement *after* the rest of the picture has been PRINTed.

It's not a good idea to place any critical graphics in those columns anyway, since some TV sets cannot display them due to overscan. Many programmers simply blank out these columns with a solid block that's the same color as the border, thus creating a frame for the screen display. This can be done efficiently—taking advantage of screen wraparound—with the single statement CALL VCHAR(1,31, CHARNUM,96). Replace CHARNUM with the ASCII code of the character defined as a solid block.

TI-99/4A Character Definitions

Michael A. Covington

Ever needed to know the definitions for the TI-99/4A's character set? If you don't have Extended BASIC, it's impossible. Listed here are all the TI's characters, their ASCII values, and their definitions. Primarily for TI BASIC owners.

The accompanying table shows the graphic definition (along with the ASCII code value and its actual appearance) of every predefined character on the TI-99/4A. You'll need this information for several reasons:

- If you want to return a character to its original appearance after having changed it.
- If you want to have more than one copy of a given character or set of characters (in different colors, for instance).
- If you want to emulate the TI-99/4A's character set on the older TI-99/4.
- Or if you simply want to see how the character set works.

In TI BASIC, there's no way to make the computer give you the definitions of predefined characters. These tables were prepared using the CALL CHARPAT statement in Extended BASIC and dumped pixel by pixel onto an Epson graphics printer—what you see here is an exact picture of how the characters look on the screen.

The Character Set

```
32     "0000000000000000"
33  !  "0010101010100010"
34  "  "0028282800000000"
35  #  "0028287C287C2828"
36  $  "0038545038145438"
37  %  "0060640810204C0C"
38  &  "0020505020544834"
39  '  "0008081000000000"
40  (  "0008102020201008"
```

41	)	"0020100808081020"
42	*	"000028107C102800"
43	+	"000010107C101000"
44	,	"0000000000301020"
45	–	"000000007C000000"
46	.	"0000000000003030"
47	/	"0000040810204000"
48	0	"0038444444444438"
49	1	"0010301010101038"
50	2	"003844040810207C"
51	3	"0038440418044438"
52	4	"00081828487C0808"
53	5	"007C407804044438"
54	6	"0018204078444438"
55	7	"007C040810202020"
56	8	"0038444438444438"
57	9	"003844443C040830"
58	:	"0000303000303000"
59	;	"0000303000301020"
60	<	"0008102040201008"
61	=	"0000007C007C0000"
62	>	"0020100804081020"
63	?	"0038440408100010"
64	@	"0038445C545C4038"
65	A	"003844447C444444"
66	B	"0078242438242478"
67	C	"0038444040404438"
68	D	"0078242424242478"
69	E	"007C40407840407C"
70	F	"007C404078404040"
71	G	"003C40405C444438"
72	H	"004444447C444444"
73	I	"0038101010101038"
74	J	"0004040404044438"
75	K	"0044485060504844"

```
 76  L  "004040404040407C"
 77  M  "00446C5454444444"
 78  N  "00446464544C4C44"
 79  O  "007C44444444447C"
 80  P  "0078444478404040"
 81  Q  "0038444444544834"
 82  R  "0078444478504844"
 83  S  "0038444038044438"
 84  T  "007C101010101010"
 85  U  "0044444444444438"
 86  V  "0044444428281010"
 87  W  "0044444454545428"
 88  X  "0044442810284444"
 89  Y  "0044442810101010"
 90  Z  "007C04081020407C"
 91  [  "0038202020202038"
 92  \  "0000402010080400"
 93  ]  "0038080808080838"
 94  ^  "0000102844000000"
 95  _  "000000000000007C"
 96  `  "0000201008000000"
 97  a  "00000038447C4444"
 98  b  "0000007824382478"
 99  c  "0000003C4040403C"
100  d  "0000007824242478"
101  e  "0000007C4078407C"
102  f  "0000007C40784040"
103  g  "0000003C405C4438"
104  h  "00000044447C4444"
105  i  "0000003810101038"
106  j  "0000000808084830"
107  k  "0000002428302824"
108  l  "000000404040407C"
109  m  "000000446C544444"
110  n  "0000004464544C44"
```

111	o	"0000007C4444447C"
112	p	"0000007844784040"
113	q	"0000003844544834"
114	r	"0000007844784844"
115	s	"0000003C40380478"
116	t	"0000007C10101010"
117	u	"0000004444444438"
118	v	"0000004444282810"
119	w	"0000004444545428"
120	x	"0000004428102844"
121	y	"0000004428101010"
122	z	"0000007C0810207C"
123	{	"0018202040202018"
124	¦	"0010101000101010"
125	}	"0030080804080830"
126	~	"0000205408000000"

4

It's Educational

Oh, So Simple

Jeannie M. Watson

Even toddlers can benefit from the TI-99/4A computer. These four short programs, with simple graphics and synthesized sound, teach the alphabet, numbers, and colors. Extended BASIC and the Speech Synthesizer module required.

Personal computers can be a tremendous tool for everyone in the family, from parents who use it for correspondence and financial management to school-age children who use it to help with homework. Even preschoolers can make use of the TI-99/4A, as the four short programs here demonstrate.

The TI-99/4A offers an excellent opportunity for introducing very young children to computers, at the same time teaching them such things as the alphabet, numbers, and colors. Because of its sprites, Extended BASIC, and optional Speech Synthesizer, this machine can provide for the needs and interests of extremely young learners. A variety of programs can be developed to appeal to your toddler just by following a few simple rules.

At Their Level

You've got to keep some things in mind when very young children use a personal computer. Preschool children haven't yet developed abstract thinking abilities. Two-year-olds, for instance, function on a very concrete thinking level. They need simple tasks to perform, with immediate responses from the computer. Steps that a child must perform need to be short and precise.

All the Senses

The TI-99/4A screen can offer excellent images to children, who need to easily see and recognize visually not only what the screen is showing, but also which keys are which. Add the Speech Synthesizer and you've got auditory responses. This is a definite advantage for preschoolers, since they often don't yet know how to read and write. And the feel as fingers push keys offers tactile stimulation and fine motor control, another sensory input. All in all, the computer can give a well-rounded set of sense-oriented learning stimulations.

Keep It Simple

Programs need to provide young children with an activity they can easily perform. Success must be readily obtainable, and a response and reward should follow immediately. Start with one single keypress and work up to more than one as the child's readiness increases. It's important that your child can perform the task you've assigned and not experience failure.

Provide Uncluttered Motivation

Movement, color, vocal rewards, and similar interest-holding tactics are important for holding a young child's attention span. However, care must be taken to provide the child with easily recognizable symbols and sounds, without unnecessary distraction or interference. The little learner hasn't yet matured enough to always separate the important from the unimportant. For example, if you want to teach the letter *B*, put it on the screen, but don't fill the screen with flowers, graphics, and other distractions. An uncluttered, simple *B* moving across the screen while a Speech Synthesizer says *B* will produce better results than a *B* buried in the midst of a field of graphics.

Programming

With the TI-99/4A, its Speech Synthesizer, and Extended BASIC, the following programs are excellent beginnings for a young child who has watched sisters, brothers, and parents at the computer.

Teach the Letters A–Z

A toddler learns through repetition and modeling (copying what others do or say). This program should be accompanied by lots of praise and attention from the parents. The child simply needs to be shown how to press one letter key at a time. (Before using any of these programs, make sure the ALPHA LOCK key is pressed down.)

Program 1. Say the Letter

```
50 CALL MAGNIFY(2)
100 CALL CLEAR
200 CALL KEY(0,KEY,STATUS)
300 IF KEY<65 OR KEY>90 THEN 200
400 CALL SPRITE(#1,KEY,2,100,10,0,25)
500 CALL SAY(CHR$(KEY))
600 GOTO 200
```

Teach the Numbers 0–9

Show your toddler the top row of keys on the keyboard. Take the child's finger and press one number at a time. Don't worry about the other keys, like S or D, because this program doesn't recognize them. Notice that the program teaches only the visual and auditory symbol of the number. It *doesn't* teach one-to-one correspondence, which is the child's realization that 8 equals eight things or 2 equals two things. That's for a different program, one beyond the scope of this article.

Program 2. Say the Number

```
50 CALL MAGNIFY(2)
100 CALL CLEAR
200 CALL KEY(0,KEY,STATUS)
300 IF KEY<48 OR KEY>57 THEN 200
400 CALL SPRITE(#1,KEY,2,100,10,0,25)
500 CALL SAY(CHR$(KEY))
600 GOTO 200
```

All Together, Now

After your child has mastered the first two programs, a review is a good idea because repetition usually helps retention. Combining the letters and numbers is one step higher on the learning scale.

Program 3. Letters and Numbers

```
50 CALL MAGNIFY(2)
100 CALL CLEAR
200 CALL KEY(0,KEY,STATUS)
300 IF (KEY<48 OR KEY>90)OR(KEY>57 AND KEY<65)THEN
    200
400 CALL SPRITE(#1,KEY,2,100,10,0,25)
500 CALL SAY(CHR$(KEY))
600 GOTO 200
```

And Colors, Too

Teaching colors to your toddler with the computer is a delight. Children like color. Be sure your TI-99/4A is connected to a color monitor or color television for this program. Show your child how to press the space bar. Each time it's pressed the computer will say a color and show it on the screen.

If you want to change the colors and the words spoken, all you have to do is modify lines 240–340, changing the values in the COLOR statements and entering the appropriate word in the CALL SAY statements. Refer to your Extended BASIC manual for the various color values and to your Speech Synthesizer manual for the correct words. You may want to change the colors, but not the spoken words, so that the colors on your TV or monitor match better with the words spoken.

Program 4. Say the Colors

```
120 CALL SCREEN(2)
130 CALL CLEAR
140 CALL CHAR(128,"FFFFFFFFFFFFFFFF")
150 FOR C=8 TO 26
160 CALL VCHAR(4,C,128,18)
170 NEXT C
180 CALL KEY(0,KEY,STATUS)
190 IF KEY<>32 THEN 180
200 L=L+1
210 IF L<5 THEN 230
220 L=1
230 ON L GOTO 240,270,300,330
240 COLOR=5
250 CALL SAY("BLUE")
260 GOTO 360
270 COLOR=7
280 CALL SAY("RED")
290 GOTO 360
300 COLOR=11
310 CALL SAY("YELLOW")
320 GOTO 360
330 COLOR=13
340 CALL SAY("GREEN")
360 CALL COLOR(13,COLOR,2)
370 GOTO 180
```

Happy Face Arithmetic

Doug Hapeman

This delightful educational game puts Happy Face's fate in your child's hands. Correct answers to addition, subtraction, multiplication, and division problems move Happy Face to the top. Extended BASIC required.

Addition, subtraction, multiplication, and division. The four primary activities in arithmetic aren't always easy for children to learn (and are sometimes hard for adults to remember). Constant practice and drill is one of the best ways to master these skills and, once mastered, sharpen them. That's why young children often spend a lot of time time hunched over paper, pencils in hand, adding, subtracting, multiplying, and dividing numbers.

Your TI-99/4A can serve as an excellent arithmetic teaching tool, giving your child problem after problem to solve. "Happy Face Arithmetic," an entertaining educational program, puts this tool at your child's fingertips.

Happy Face, Sad Face

After you've typed in and saved the program, load it from tape or disk and enter RUN. Your child has four types of problems to choose from—addition, subtraction, multiplication, and division. Simply press the appropriate key.

There are even three difficulty levels in Happy Face Arithmetic. Depending on your child's age and ability, you can choose *Easy* (uses numbers from 1 to 9), *Medium* (1–25), or *Hard* (1–99). There aren't any fractional numbers in Happy Face Arithmetic, only integers. This applies to answers as well. In the division mode, all answers will be integers.

The program displays five questions at a time. The upper part of the screen is a series of five floors, with ladders connecting them. Each time all five questions are answered correctly, Happy Face climbs a ladder to the next level. If just one of the five is answered incorrectly, Happy Face remains on the same floor and the correct answers are shown. Of course, the object is to keep answering problems until Happy Face reaches the top level. Your child will delight in the reward—balloons launched and music playing. You'll also see the percentage of problems answered correctly.

Entering answers is easy. Just press the correct number keys and then hit the ENTER key. Mistakes can be corrected by using the FCTN and arrow keys to move the cursor left or right, then entering the right number. Once you press the ENTER key, however, you can't go back and change the answer.

Happy Face Arithmetic

```
100 REM HAPPY FACE MATH
110 REM
120 DIM W(5),X(5),Y(5),Z(5),C$(13)
130 DISPLAY AT(9,3)ERASE ALL:RPT$("*",23): :"  * H
    A P P Y - F A C E *": :"  * A R I T H M E T I C
     *": :"  ";RPT$("*",23)
140 REM *******DEFINE CHARACTERS AND COLORS*******
150 FOR I=10 TO 0 STEP -1 :: READ C,C$(I):: CALL CH
    AR(C,C$(I)):: DISPLAY AT(23,12)BEEP:I :: NEXT I
160 FOR I=0 TO 8 :: CALL COLOR(I,2,13):: NEXT I ::
    CALL COLOR(9,5,11):: CALL COLOR(10,15,13):: CAL
    L COLOR(12,2,13):: CALL SCREEN(15)
170 DATA 95,001818007E001818,97,FFFFFFFFFFFFFFFF,98
    ,FFFFFF,99,FFFFC0C0FFFFC0C0,100,FFFF0303FFFF030
    3,104,FFFFFFFF7F3E1C
180 DATA 112,030F3F7F7F3FFFFFFFFF3F7F7F37070100C0CC
    FEFEFCFFFFFFFFFCFEFEECE08
190 DATA 124,071820404C888081808884434020180 7E0180C
    0232110181011121C2020418E
200 DATA 128,071820404C888081808384484020180 7E0180C
    023211018101C12112020418E
210 DATA 132,071820404C888081809F904844231807E0180C
    023211018101F9091222C418E
220 DATA 136,030F1F3F3F3F1F0F07030101020408 80E0F0
    F8F8F8F8F0E0C08
230 REM *******OPTIONS MENUS*******
240 DISPLAY AT(23,3):"*PRESS ANY KEY TO BEGIN*" ::
    CALL KEY(0,KEY,S):: IF S=0 THEN 240
250 DISPLAY AT(5,1)ERASE ALL:"CHOOSE THE TYPE OF PR
    OBLEMS:": : :"PRESS   FOR": : :"  1  =  ADDITIO
    N": :"  2  =  SUBTRACTION"
260 DISPLAY AT(15,3)BEEP:"3  =  MULTIPLICATION": :"
      4  =  DIVISION"
270 CALL KEY(0,K,S):: IF S=0 OR K<49 OR K>52 THEN 2
    70 :: FLAG,ND,NR=0 :: M=2^(1/12)
280 DISPLAY AT(5,1)ERASE ALL BEEP:"CHOOSE LEVEL OF
    DIFFICULTY:": : :"PRESS   FOR": : :"  1  =  EAS
    Y PROBLEMS": :"  2  =  MEDIUM PROBLEMS"
290 DISPLAY AT(15,3):"3  =  DIFFICULT PROBLEMS"
```

```
300 CALL KEY(0,KK,S):: IF S=0 OR KK<49 OR KK>51 THE
    N 300 :: CALL CLEAR
310 REM *******PRINT SCREEN*******
320 CALL HCHAR(1,1,104,32)
330 FOR R=5 TO 14 STEP 3 :: CALL HCHAR(R,1,98,32)::
     ON (R-2)/3 GOSUB 340,350,360,350 :: NEXT R ::
    CALL HCHAR(17,1,97,256):: GOTO 380
340 FOR C=12 TO 20 STEP 8 :: GOSUB 370 :: NEXT C ::
     RETURN
350 FOR C=8 TO 24 STEP 8 :: GOSUB 370 :: NEXT C ::
    RETURN
360 FOR C=4 TO 28 STEP 8 :: GOSUB 370 :: NEXT C ::
    RETURN
370 CALL VCHAR(R,C,99,3):: CALL VCHAR(R,C+1,100,3):
    : RETURN
380 R=113 :: H=15 :: CALL SPRITE(#1,124,2,R,120,#2,
    124,2,17,120,#3,112,16,1,192,0,-1):: CALL MAGNI
    FY(3)
390 CALL MOTION(#1,0,H,#2,0,-H):: FOR I=18 TO 21 ::
     DISPLAY AT(I,1):RPT$(" ",28):: NEXT I :: CALL
     HCHAR(23,1,97,64)
400 REM *******DIFFICULTY LEVEL NUMBER LIMITS******
    *
410 ON KK-48 GOTO 420,430,440
420 N=9 :: GOTO 460
430 N925 :: GOTO 460
440 N=99
450 REM *******SELECT RANDOM NUMBERS*******
460 FOR I=1 TO 5 :: RANDOMIZE :: W(I)=INT(RND*N)+1
    :: NEXT I :: IF K-48>2 THEN N=9 ELSE N=N
470 FOR I=1 TO 5 :: RANDOMIZE :: X(I)=INT(RND*N)+1
    :: NEXT I
480 FOR I=1 TO 5 :: IF W(I)>=X(I)THEN 490 :: HOLD=W
    (I):: W(I)=X(I):: X(I)=HOLD
490 NEXT I
500 REM *******CALCULATE CORRECT ANSWERS*******
510 FOR I=1 TO 5 :: ON K-48 GOSUB 520,530,540,550 :
    : NEXT I :: GOTO 570
520 Z$="+" :: Z(I)=W(I)+X(I):: RETURN
530 Z$="-" :: Z(I)=W(I)-X(I):: RETURN
540 Z$="x" :: Z(I)=W(I)*X(I):: RETURN
550 Z$="_ " :: Z(I)=W(I)*X(I):: HOLD=W(I):: W(I)=Z(I
    ):: Z(I)=HOLD :: RETURN
560 REM *******PRINT PROBLEMS*******
570 COL=1 :: FOR I=1 TO 5 :: DISPLAY AT(18,COL):USI
    NG "####":W(I):: DISPLAY AT(19,COL):" ";Z$
580 DISPLAY AT(19,COL+2):USING "##":X(I):: DISPLAY
    AT(20,COL):" ---" :: COL=COL+
6 :: NEXT I
590 REM *******ACCEPT ANSWERS*******
```

```
600 COL=5 :: FOR I=1 TO 5 :: ACCEPT AT(21,COL-LEN(S
    TR$(Z(I))))VALIDATE(DIGIT)SIZE(3)BEEP:Y(I):: CO
    L=COL+6 :: NEXT I
610 REM *******PRINT CORRECT ANSWERS*******
620 J=0 :: COL=1 :: FOR I=1 TO 5 :: IF Y(I)<>Z(I)TH
    EN Z$="OOPS" ELSE Z$="GOOD" :: IF Y(I)<>Z(I)THE
    N 630 :: NR=NR+1 :: GOTO 640
630 J=1
640 DISPLAY AT(23,COL):Z$ :: DISPLAY AT(24,COL):USI
    NG "####":Z(I):: COL=COL+6 :: NEXT I :: IF J=0
    THEN 690
650 REM *******MUSIC FOR NOT FIVE RIGHT*******
660 P=128 :: GOSUB 670 :: FOR I=25 TO 1 STEP -1 ::
    CALL SOUND(-40,440*M^I,1,-3,2):: NEXT I :: P=12
    4 :: GOSUB 670 :: GOTO 790
670 CALL PATTERN(#1,P,#2,P):: RETURN
680 REM *******SPRITE MOVEMENT AND MUSIC FOR FIVE R
    IGHT*******
690 FLAG=FLAG+1 :: P=132 :: CALL POSITION(#1,R,C)::
     ON FLAG GOTO 700,720,700,740
700 FOR I=57 TO 185 STEP 64 :: IF C>I THEN 710 :: C
    =I :: GOTO 760
710 NEXT I :: C=57 :: GOTO 760
720 FOR I=217 TO 25 STEP -64 :: IF C<I THEN 730 ::
    C=I :: GOTO 760
730 NEXT I :: C=217 :: GOTO 760
740 FOR I=153 TO 89 STEP -64 :: IF C<I THEN 750 ::
    C=I :: GOTO 760
750 NEXT I :: C=153
760 GOSUB 780 :: CALL MOTION(#1,-5,0):: R=R-24 :: P
    =124
770 FOR I=4 TO 14 :: CALL COLOR(9,I,11):: CALL SOUN
    D(60,440*M^I,1):: NEXT I :: CALL SOUND(200,440*
    M^15,1):: GOSUB 780 :: H=-H :: GOTO 790
780 CALL COINC(#1,R,C,3,CH):: IF CH=0 THEN 780 ELSE
     CALL MOTION(#1,0,0,#2,0,0):: CALL LOCATE(#1,R,
    C):: CALL PATTERN(#1,P,#2,P):: RETURN
790 CALL COLOR(9,5,11):: ND=ND+5 :: IF R>17 THEN 39
    0
800 REM *******ROUTINE FOR END OF TEST*******
810 CALL LOCATE(#2,R,C-16):: CALL MOTION(#1,0,H,#2,
    0,H)
820 FOR I=5 TO 21 :: C=INT(RND*212)+16 :: CALL SPRI
    TE(#I,136,8,192,C,-20,5):: NEXT I
830 PER=INT(NR/ND*100+.5)
840 DISPLAY AT(18,1):" HAPPY FACES TOGETHER AGAIN":
     :" YOU HAD";NR;"RIGHT OUT OF";ND:" THAT'S";PER
    ;"PERCENT CORRECT!"
850 IF NR<ND THEN 880
860 REM *******MUSIC FOR TEST OVER*******
```

```
870 DISPLAY AT(23,1):" TERRIFIC, A PERFECT SCORE!":
    " KEEP UP THE EXCELLENT WORK" :: GOTO 890
880 DISPLAY AT(23,1):"NOW SEE IF YOU CAN REACH THE"
    :"  TOP WITH A PERFECT SCORE"
890 CALL KEY(0,K,S):: IF K=89 OR K=78 THEN 1240 ::
    T=80
900 CALL SOUND(T*3,311,0):: CALL PATTERN(#1,132,#2,
    132)
910 CALL SOUND(T,349,0)
920 CALL SOUND(T*4,392,0,311,4,156,2):: CALL KEY(0,
    K,S):: IF K=89 OR K=78 THEN 1240
930 CALL SOUND(T*3,392,0,156,2)
940 CALL SOUND(T,415,0,156,2)
950 CALL SOUND(T*3,392,0,233,2):: CALL KEY(0,K,S)::
     IF K=89 OR K=78 THEN 1240
960 CALL SOUND(T,370,0,233,2)
970 CALL SOUND(T*4,392,0,233,2):: CALL KEY(0,K,S)::
     IF K=89 OR K=78 THEN 1240
980 CALL SOUND(T*4,415,0,311,4,262,2):: CALL PATTER
    N(#1,124,#2,124)
990 CALL SOUND(T*4,415,0,311,4,247,2):: CALL KEY(0,
    K,S):: IF K=89 OR K=78 THEN 1240
1000 CALL SOUND(T*8,392,0,311,4,233,2):: CALL KEY(0
     ,K,S):: IF K=89 OR K=78 THEN 1240
1010 CALL SOUND(T*4,466,0,392,4,156,2):: CALL PATTE
     RN(#1,132,#2,132)
1020 CALL SOUND(T*3,466,0,156,2):: CALL KEY(0,K,S):
     : IF K=89 OR K=78 THEN 1240
1030 CALL SOUND(T,523,0,156,2)
1040 CALL SOUND(T*3,466,0,196,2)
1050 CALL SOUND(T,440,0,196,2)
1060 CALL SOUND(T*4,466,0,196,2):: CALL KEY(0,K,S):
     : IF K=89 OR K=78 THEN 1240
1070 CALL SOUND(T*8,523,0,311,4,208,2):: CALL PATTE
     RN(#1,124,#2,124):: CALL KEY(0,K,S):: IF K=89
     OR K=78 THEN 1240
1080 CALL SOUND(T*4,466,0,311,4,196,2)
1090 CALL SOUND(T*3,466,0):: CALL KEY(0,K,S):: IF K
     =89 OR K=78 THEN 1240
1100 CALL SOUND(T,494,0)
1110 CALL SOUND(T*3,523,0,208,2):: CALL PATTERN(#1,
     132,#2,132)
1120 CALL SOUND(T,494,0,208,2)
1130 CALL SOUND(T*2,523,0,208,2)
1140 CALL SOUND(T*10,622,0,415,4,262,2):: CALL KEY(
     0,K,S):: IF K=89 OR K=78 THEN 1240
1150 CALL SOUND(T*3,392,0,156,2):: CALL PATTERN(#1,
     124,#2,124)
1160 CALL SOUND(T,415,0,147,2)
1170 CALL SOUND(T*2,440,0,139,2)
```

```
1180 CALL SOUND(T*10,466,0,330,4,131,2):: CALL KEY(
     0,K,S):: IF K=89 OR K=78 THEN 1240
1190 CALL SOUND(T*8,311,0,220,4,175,2):: CALL PATTE
     RN(#1,132,#2,132):: CALL KEY(0,K,S):: IF K=89
     OR K=78 THEN 1240
1200 CALL SOUND(T*8,349,0,208,4,117,2):: CALL KEY(0
     ,K,S):: IF K=89 OR K=78 THEN 1240
1210 CALL SOUND(T*16,311,0,196,4,156,2):: CALL PATT
     ERN(#1,124,#2,124):: CALL KEY(0,K,S):: IF K=89
      OR K=78 THEN 1240
1220 DISPLAY AT(12,2):"*WISH TO PLAY AGAIN? (Y/N)"
     :: GOTO 900
1230 REM *******FINISH OR REPEAT SESSION*******
1240 CALL DELSPRITE(ALL):: IF K=89 THEN 240 :: DISP
     LAY AT(13,6)ERASE ALL:"HAVE A NICE DAY!" :: ST
     OP
```

Spelling Tutor

Jeannie M. Watson

Does your child need practice with a list of spelling words? This program, which requires the Speech Synthesizer and the Terminal Emulator II command module, speaks and spells the list you enter.

Practice makes perfect. That cliché is especially true of learning how to spell. School children spend hours practicing this week's spelling words— listening to the teacher pronounce the words, writing them down, and taking tests.

You can give your child an advantage—additional spelling practice—with "Spelling Tutor." The TI-99/4A can duplicate, to a large extent, the classroom environment, even speak the words when you have Texas Instrument's Speech Synthesizer connected. This program, which also requires the Terminal Emulator II command module, will provide your child with entertaining and educational spelling drill.

Drill and Practice

The key phrase is *repetition for retention.* Spelling words, to be learned, must be memorized and retained. This program can provide an unending drill until all the words are learned and spelled correctly on the test. This is accomplished by following a series of steps. You (or your child) enter a word list, one word at a time. Each entry consists of a correct spelling of the word, a comma, and the phonetic-like spelling required to make the Speech Synthesizer pronounce the word properly.

You can enter up to 20 words. For fewer than 20, simply type a comma and press ENTER when prompted for another word. The words are printed and spoken in a list form to verify that they've been typed in correctly. Type Y or N for yes or no. You're allowed to enter the word list again if a mistake has been made.

Spelling Tutor personalizes the child's learning by asking for the student's name, again followed by a comma and phonetic-like spelling of the name. Drill and practice then begins in earnest. The computer asks the child to spell each word in the list. Three chances are given to spell every word in the list correctly. When the third mistake is made, the program spells the word, one letter at a time, saying and printing each letter. The printed word stays on the screen for several seconds. Once the drill is finished, the child is given a test.

Learning from Mistakes

Children learn well from their mistakes if errors are treated as learning experiences, not as some sort of negative reinforcement. This program immediately reteaches incorrectly spelled words. If a child makes a spelling error, Spelling Tutor gives immediate visual and auditory feedback by displaying the word on the screen after the three chances are up. When an error is made, the TI beeps and prints INCORRECT. After the spelling test is given, the program tells the child the number of correct and incorrect words, and again displays the incorrectly spelled words as they *should* be spelled. Your child learns from mistakes in two ways: immediate feedback and immediate review.

When a right answer is entered, the computer displays a happy face graphic and says the child's name.

Using Spelling Tutor

Using this program is simple. The important thing to remember is to follow each word—including the student's name—with a comma and a phonetic-like spelling of the word to allow the Speech Synthesizer to say the word correctly. Again, if you have fewer than 20 words and want to proceed, type a comma when asked for the next word and the program will continue.

Once the spelling test has been taken, two things can happen. If the child spelled all the words in the list correctly during the test, the program returns to the name prompt so that another student can play. If the child missed one or more words, the list is again presented. Another test is given at its end.

To quit the program, press FCTN-4.

Controlling the Speech Synthesizer

Before introducing your child to this program, you may want to check the pronunciation, by the Speech Synthesizer, of the spelling word list. The TI Speech Synthesizer is a wonderful tool, but it recognizes only phonetic spellings. You'll have to enter (after the comma) the correct phonetic spelling so that the Synthesizer can read the word. Spelling Tutor includes a RUN 2000 option that lets you check the pronunciation of the words *before* your child hears them. Just type RUN 2000, press ENTER, and listen for pronunciation errors.

Phonetic spellings are listed in the dictionary, so if you aren't sure of one, be sure to look it up. Exceptions-to-the-rule words are given phonetically, too. An example is the word *COLONEL*—phonetically it should be spelled *KERN-L,* with the accent on the first syllable. The accent marks to use are ˜ or ˆ, or you may want to try ! or ?. A little experimenting should solve any problems.

Spelling Tutor

```
100 CALL CLEAR
110 CALL CHAR(154,"0F1020409E808C81")
120 CALL CHAR(155,"F008040279013181")
130 CALL CHAR(156,"919191884423100F")
140 CALL CHAR(157,"898989112 2C408F0")
150 CALL CHAR(152,"3C4299A1A99423C")
160 OPEN #1:"SPEECH",OUTPUT
170 CALL CLEAR
180 PRINT "SPELLING TUTOR":"FOR T.I. 99/4A WITH TEI
    I":"":CHR$(152);"JEANNIE WATSON     ": : : : :
    : :
190 PRINT #1:"SPELLING TUTOR BY JEANNIE WATSON"
200 PRINT "ENTER WORD LIST":"IN THIS FORMAT":"WORD,
     PHONICS SPELLING":"HIT COMMA TO ENTER":"SHORTE
    NED LIST": : : : :
210 WN=1
220 DIM A$(25)
230 DIM B$(25)
240 DIM C$(25)
250 DIM E$(25)
260 DIM P$(25)
270 PRINT "ENTER WORD NUMBER":WN
280 INPUT A$(WN),B$(WN)
290 IF A$(WN)="" THEN 330
300 IF WN=20 THEN 330
310 WN=WN+1
320 GOTO 270
330 CALL CLEAR
340 PRINT #1:"YOUR WORD LIST IS"
350 FOR W=1 TO WN-1
360 IF A$(W)="" THEN 410
370 PRINT A$(W):
380 PRINT #1:B$(W)
390 NEXT W
400 PRINT "": :
```

```
410 INPUT "IS WORD LIST CORRECT? (Y OR N)":V$
420 IF V$="Y" THEN 450
430 WN=1
440 GOTO 270
450 Z=WN-1
460 CALL CLEAR
470 PRINT "ENTER STUDENT'S NAME":"NAME, PHONIC NAME
    ":
480 INPUT N$,M$
490 IF N$="" THEN 200
500 PRINT #1:M$
510 CORRECT=0
520 FOR K=0 TO 20
530 E$(K)=""
540 P$(K)=""
550 NEXT K
560 CALL CLEAR
570 PRINT "PRESS SPACE BAR":"TO BEGIN": : : : : : :
     : : :
580 CALL KEY(3,K,S)
590 IF K<>32 THEN 580
600 CALL CLEAR
610 P=1
620 T=0
630 PRINT #1:"PLEASE SPELL":B$(P)
640 INPUT C$(P)
650 IF C$(P)=A$(P)THEN 900
660 CALL SOUND(100,500,5)
670 PRINT "INCORRECT": :
680 T=T+1
690 IF T>=3 THEN 760
700 GOTO 630
710 T=1
720 IF P=Z THEN 990
730 P=P+1
740 CALL SOUND(100,600,5)
750 GOTO 740
760 CALL CLEAR
770 PRINT #1:B$(P):"IS SPELLED"
780 FOR O=1 TO LEN(A$(P))
790 D$=SEG$(A$(P),O,1)
800 PRINT #1:D$
810 PRINT D$
820 NEXT O
830 PRINT "": : : : : : : : : : : :
840 FOR DELAY=1 TO 500
850 NEXT DELAY
860 IF P=Z THEN 990
870 P=P+1
880 CALL CLEAR
```

```
890 GOTO 620
900 PRINT #1:"YOU ARE CORRECT":M$
910 PRINT "YOU ARE CORRECT ";N$: : : : :
920 GOSUB 1340
930 FOR DELAY=1 TO 500
940 NEXT DELAY
950 IF P=Z THEN 990
960 P=P+1
970 CALL CLEAR
980 GOTO 630
990 CALL CLEAR
1000 REM SPELLING TEST
1010 PRINT "SPELLING TEST": : : : : : : : : : : :
1020 CORRECT=0
1030 FOR Y=1 TO Z
1040 PRINT #1:M$:"PLEASE SPELL THE WORD":B$(Y)
1050 INPUT L$
1060 IF L$=A$(Y)THEN 1110
1070 E$(R)=A$(Y)
1080 P$(R)=B$(Y)
1090 R=R+1
1100 GOTO 1120
1110 CORRECT=CORRECT+1
1120 NEXT Y
1130 CALL CLEAR
1140 IF CORRECT=Z THEJ 1250
1150 PRINT #1:M$:"YOU SCORED":CORRECT:"OUT OF A TOT
     AL OF":Z
1160 PRINT N$;" YOU SCORED":CORRECT;"RIGHT":(Z-CORR
     ECT);"WRONG": : : :
1170 PRINT "THESE WORDS NEED":"MORE STUDY:": : : :
     : :
1180 FOR U=0 TO R
1190 PRINT E$(U): :
1200 PRINT #1:P$(U)
1210 FOR DEL=1 TO 1000
1220 NEXT DEL
1230 NEXT U
1240 GOTO 600
1250 CALL CLEAR
1260 PRINT #1:"YOU ARE A GOOD STUDENT":M$:"YOU GOT
     THEM ALL RIGHT"
1270 PRINT N$;" IS A GOOD SPELLER!!!!!":
1280 GOSUB 1340
1290 FOR DEL=1 TO 1000
1300 NEXT DEL
1310 CALL CLEAR
1320 GOTO 470
1330 REM HAPPY FACE SUBROUTINE
1340 RO=12
```

```
1350 CO=20
1360 FOR CH=154 TO 157
1370 CALL HCHAR(RO,CO,CH)
1380 IF CO<21 THEN 1420
1390 CO=20
1400 RO=RO+1
1410 GOTO 1430
1420 CO=CO+1
1430 NEXT CH
1440 RETURN
1450 OPEN #2:"SPEECH",OUTPUT
1460 CALL CLEAR
1470 INPUT "ENTER WORD ":W$
1480 IF W$="" THEN 100
1490 PRINT #2:W$
1500 GOTO 1470
2000 GOTO 1450
2010 END
```

5

Assembly Language

TI-99/4A Memory and Assembly Language Tips

Lee M. Marsh

This informative article outlines the TI-99/4A's memory and offers some interesting tips about assembly language programming.

The TI-99/4A is an interesting and complex computer that uses three processors. The main processor is the TMS9900, which handles all numeric and data manipulation. The second is the TMS9918A Video Display Processor, responsible for keeping the data displayed on the screen and for the automatic motion of sprites. The third microprocessor is the TMS9919 sound generator controller, with three sound generators that can be independently set for frequency and volume.

Before writing assembly language programs on the TI-99/4A, you must know where things are located in memory. What follows are some of the tidbits and techniques I've learned from using the *Editor/Assembler.* They may be of help to you as well.

Memory Complications

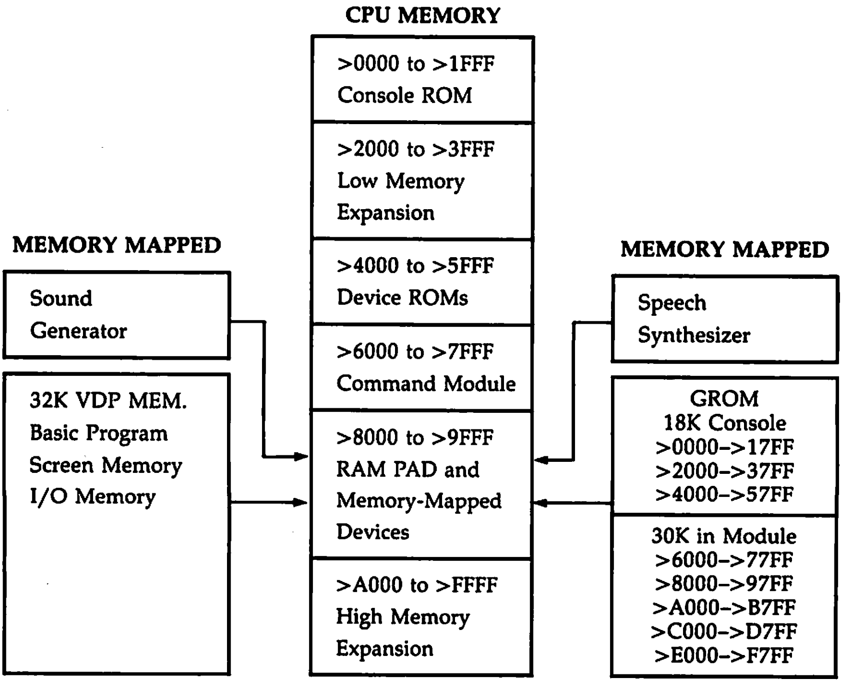
Three different command modules for the TI-99/4A allow you to load and run assembly language programs—the *Editor/Assembler,* Mini Memory, and Extended BASIC. The *Editor/Assembler* and Mini Memory allow you to assemble and run assembly language programs. Extended BASIC lets you run assembly language programs, though not create them. We'll get into its shortcomings in a moment.

The TI-99/4A uses a fairly complicated memory scheme. The TMS9900 microprocessor has 16 address lines and is capable of directly accessing 65,535 bytes (64K) of memory. (Though there can be more than 64K of memory in the system, it's something left for later.) For now, let's see how the memory that's directly addressable is divided.

The first 8K, from >0000 to >1FFF (the > symbol indicates that the number is in hexadecimal format), is built into the TI's console and contains some of the BASIC interpreter and utility programs such as SCAN, which checks to see if a key's been pressed and returns its value. Some of these routines can be used by your assembly language programs.

The next block, from >2000 to >3FFF, is called Low Memory Expansion. This 8K is part of the 32K Memory Expansion card—how it's used is determined by the module you've got plugged into the console. This block usually has utility routines, user assembly programs, and a REFerence/DEFinition (REF/DEF) table which lets you name your programs so that they can call each other and be accessed by the CALL LINK statement in Extended BASIC. This is one of the areas where the *Editor/Assembler,* Mini Memory, and Extended BASIC differ.

The TI-99/4A's Memory



Differences

As an example, consider how you'd access a routine in this 8K block. Assume you're trying to call the VSBW utility routine. VSBW writes a single character to screen memory.

The *Editor/Assembler* uses the statement **BLWP VSBW**. The Assembler would resolve the REFerence to VSBW when compiling. (BLWP—Branch and Load Workspace Pointers—is like a GOSUB in BASIC.)

Because Extended BASIC cannot resolve external REFerences, you must look them up on page 415 of the *Editor/ Assembler* manual. For instance, in Extended BASIC you'd have to write **BLWP >2020** instead. If you used a REFerence, you'd get a *Bad Tag* error when you loaded the program in Extended BASIC. (Another note concerning Extended BASIC: It doesn't support DSRLNK—Device Service Routine Link—which means that you cannot use peripherals from your assembly language programs. You either have to do it from BASIC or try to write your own DSRLNK routine.)

When you're using the Mini Memory, you also have to look up the hexadecimal address of such a routine. However, since the routines are in a different area of memory (>6000 to >6FFF), you need to refer to the Mini Memory's own manual. Sticking with the same example, you'd enter **BLWP @6024** if you were using the Mini Memory module.

As you can see, calling utilities is one of the major things to watch when you're converting a program to run under another command module.

Peripherals and Modules

The 8K of memory from >4000 to >5FFF is located in the peripheral devices connected to the system (the disk, RS-232, or p-code cards). These are selected one at a time through Communications Register Unit (CRU) addressing. CRU addressing provides for economical single-bit I/O processing.

The cassette recorder motors, for example, are turned on and off by CRU instructions. Have a disk controller card? Inspect addresses >4000 to >5FFF and you'll find all zeros—but turn on CRU bit >1100 and look again—now the disk ROM program is there.

To turn on the ROMs with the Debugger, type **C1100,1**, press ENTER, type **1**, and press ENTER. **C1100,1** ENTER **0** ENTER deselects the ROMs when you're finished. Make a note that all peripheral ROMs share the same memory space.

The 8K from >6000 to >7FFF is used by the command modules which plug into the console port. This memory can be either RAM or ROM, depending on the module. (As far as I know, Mini Memory is the only one with RAM.)

What's Left

The next 8K, from >8000 to >9FFF, is very important because it contains the RAM PAD and all memory-mapped devices. The RAM PAD is located at >8300 to >83FF. This 256-byte area is the nerve center of the system, and is used to pass parameters and status from program to program.

The final block is large, from >A000 to >FFFF. This is the last 24K of the 32K Memory Expansion card and is at your disposal when using the *Editor/Assembler* or Mini Memory module. You're confined somewhat when in Extended BASIC. From >A000 to the address contained in >8336–37 is free—the memory above that address is used by Extended BASIC.

Memory Mapping

Memory mapping allows the TI-99/4A to exceed 64K of memory. Several devices are memory mapped, including the sound generator, the Speech Synthesizer, Graphic Programming Language ROM (GROM), and Video Display Memory. Memory mapping in the TI-99/4A means that you write the address you want to read/write from into a specific memory address, then read/write the data into a different memory address. You can think of memory mapping as a window through which these devices can exchange data.

- GROM contains utility routines which you can access with BLWP GPLLNK. A maximum of 48K of GROM can reside in the system, divided into eight segments of 6K each. Three are in the TI's console, and the other five can be inside the command modules. The 18K of GROM in the TI console contains the rest of the BASIC interpreter.
- You'll notice that no mention has been made of directly addressable console memory for your BASIC programs. The 16K of memory in the unexpanded TI-99/4A is controlled by the TMS9918A Video Display Processor and isn't in the TMS9900's directly addressable space. BASIC programs are stored in VDP memory. (I think that's one reason why TI BASIC is not as fast as it could be. BASIC programs and data

must be pushed through an eight-bit window to and from VDP memory.)
- Sound is another memory-mapped device, and to use it you put a sound table in VDP memory which gives information that the three sound generators need (frequency, duration, and attenuation), then tell the sound generator to process it.

One Last Tip

Here's one last tip to help you poke around inside Extended BASIC. If you replace all the external REF statements in an assembly language source listing with their hex equivalent in EQU statements (VSBW EQU >2020, for instance), you can recompile it and run it from Extended BASIC. To load and run, use the following procedure:

1. Get the ready prompt in Extended BASIC.
2. Type **CALL INIT** and press ENTER.
3. Type **CALL LOAD ("DSK1.DEBUG")**, assuming you are using DSK1. and you recompiled it under the filename DEBUG.
4. Type **CALL LINK("DEBUG")**, again assuming that you left the DEF statement in the source program as DEBUG.
5. Press U to change the screen offset so that the characters show up.
6. Use the Debugger as you would with the *Editor/Assembler*, but now you can inspect Extended BASIC ROM and GROM.

References

I find the following information invaluable when writing assembly language programs. The numbers in italics are page numbers in the *Editor/Assembler* manual.

Address	**Feature or Function**
0000–1FFF	8K Console ROM *399*
000E	Keyboard scan *250*
2000–2001	ID Code >A55A *411*
2000–3FFF	Low memory expansion *411*
200A–2019	Argument identifier *278*
2022	UTLTAB *264*
2022–20FF	UTLTAB data *411*
2024	First free high memory *264*
2026	Last free high memory *264*

Address	Feature or Function
2028	First free low memory *264*
202A	Last free low memory *264*
202C	Checksum *264*
202E	PAB flag pointer *264*
2030	GPL return address *264*
2032	CRU address save *264*
2034	Save DSR/SUB address *264*
2036	Save DEV/SUB name length *264*
2038	Save DEV/SUB name pointer *264*
203A	UTILWS (utility workspace) *246*
20bA	USRWSP (user workspace) *246*
2100–2128	Utility vectors *411*
2128–26FF	Utility programs *411*
2700–3F37	Assembly programs *411*
3f38–3FFF	REF/DEF table *246, 307*
4000–5FFF	Peripheral ROM *399*
6000–7FFF	Command module *399*
8000–9FFF	RAM PAD memory-mapped devices *399*
8300–83FF	RAM PAD *404*
8310	Value stack pointer *278*
8312	Number of arguments *278*
831A	First free VDP memory *253*
833C	Pointer to PAB in VDP memory *302*
834A	Start of floating-point accumulator *252*
8354	Math errors *254*
8354–8355	Length of name in cassette call *253*
8356	Address of character after CS1 in cassette call *253*
8356–8357	Pointer to name length in PAB *262*
835C	Arguments *252*
836E	VSPTR *259*
8370–8370	Highest VDP memory *405*
8372	Least significant byte of data stack pointer Most significant byte=83 *404*
8373	Least significant byte of subroutine stack pointer Most significant byte=83 *404*
8374	Keyboard number to scan *250*
8375	ASCII key code returned *250*
8376	Joystick Y position *250*
8377	Joystick X position *250*
8378	Random number generator *405*
8379	VDP interrupt timer 405
837A	Maximum number of sprites *405*
837B	Copy of VDP status *405*
837C	GPL status byte *250, 405*
837D	VDP charactor buffer *405*

Address	Feature or Function
837E	Current screen row *405*
837F	Current screen column *405*
83C0–84FG	Interpreter workspace *405*
83C0	Random number seed *406*
83C2	Interrupt control flag *406*
83C4	User interrupt address *406*
83CA	Console keyboard debounce *406*
83CC	Sound list pointer *312, 406*
83CE	>01 to trigger sound generator *312, 406*
83D0	>0000 during cassette call *253*
83D0–83D4	GROM/ROM search pointers *406*
83D4	Copy of VDP register #1 *326*
83D6–83D7	Screen time out counter *406*
83D8	Scan return address *406*
83DA	Scan player number *406*
83FD	Least significant bit=1 for VDP sound table *312*

This is a list of "windows" to the memory-mapped devices.

Address	Feature or Function
8400	Sound generator *317*
8800	VDP read data *267*
8802	VDP read status register *269*
8C00	VDP write data *268*
8C02	VDP write address *266*
9000	Speech read data *351*
9400	Speech write data *351*
9800	GROM read data *271*
9802	GROM read address *270*
9C00	GROM write data *271*
9C02	GROM write address *270*

TI FastSearch

Thomas W. Kirk

Looking for a fast way to search through hundreds of array elements? "TI FastSearch" is a machine language subprogram which you can call and link to with Extended BASIC. Editor/Assembler, *Extended BASIC, disk drive, and Memory Expansion unit necessary.*

Searching through ten records, looking for just the right one, is certainly possible in TI BASIC or Extended BASIC. But multiply the number of records by ten, or a hundred, and the task begins to look impossible. Even Extended BASIC is just too slow. Only machine language, the language the computer uses when it communicates with itself, will do.

I discovered this when a friend told me of a business which had a TI-99/4A on display—the business wanted to use it to keep track of sales documents. I wrote an Extended BASIC program to do what was needed, but it was just too slow. One of the problems was the search routine, which examined the contents of an array containing pointers to a random access file. After I came to the conclusion that machine language was required, I distilled the problem into two parts: finding all the array elements in turn and comparing them with the value sought.

Finding Array Elements and Simple Variables

The BASIC Support Utility subprograms contained in the Extended BASIC command module provide easy access to arrays created in Extended BASIC. The *Editor/Assembler* manual describes the setup required to invoke the routines.

NUMREF, one of these utility subprograms, is used to recover a numeric array element which was created in Extended BASIC and stored in floating-point format. The linkage from Extended BASIC follows this format:

CALL LINK*("NAME",K,N(x))*

where *NAME* is the defined name of the assembly language subprogram you're calling, *K* is the first parameter, and *N(x)* the second parameter. Execution of this command transfers control of the computer to the called assembly language program and passes information concerning the particulars of each argument to a value stack in the VDP memory pointed to by contents of the PAD at locations >8300 to >8315 (the > symbol indicates hexadecimal notation).

However, the easiest way to locate a particular argument value is through assembly language:

```
LI R0,1          Element number or 0 if argument is a simple
                 variable
LI R1,1          Parameter number
BLWP @NUMREF     Invoke NUMREF
```

NUMREF finds the specified argument and places it at eight consecutive byte locations starting at location >834A. The value is stored in radix 100 format.

STRREF, another utility subprogram, is used to recover a string element which was created in Extended BASIC and which is stored in string format—06667979717369 where *06* is string length in bytes, and where *66, 79, 79, 71, 73,* and *69* are ASCII codes. STRREF places the recovered string in a user-designated buffer. The syntax, in assembly language, is

```
LI R0,1          Element number or 0 if argument is a simple
                 variable
LI R1,1          Parameter number
LI R1,BUF        Buffer location—the first byte must contain
                 buffer length. Attempting to load a string which
                 exceeds specified value will result in an error.
BLWP @STRREF     Invoke STRREF
```

Entering TI FastSearch

Your first job, then, is to enter the assembly language listing (Program 1) at the end of the article. Using the *Editor/ Assembler,* you need to enter, assemble, and debug the program. It's not long (not as long as some other machine language programs in this book, at least), and when you're satisfied that it's working correctly, save it to disk. Save it in a FIXED 80 format. If you're planning to use the demonstration program (Program 2), make sure you save the file as **SORTA.**

Once you have the program on disk, you can load it into memory and use the CALL LINK format to access its two subroutines.

Comparing Values

A search is done simply and quickly by comparing the bytes of the passed argument with the bytes of the array element fetched by the BASIC Support Utility. The position is tracked by a counter. When a match is detected, the counter is stored in a predetermined memory location, and control is returned to Extended BASIC. The BASIC program then PEEKs at the agreed memory location and captures the required array position.

Program 2 is an Extended BASIC program which shows you how easy it is to link to assembly language subroutines from TI BASIC and Extended BASIC. It creates a 1000-element array, asks you to specify one of those elements, and then locates that array for you.

FIND and AGAIN

The file you've created, SORTA, contains two subprograms: FIND and AGAIN.

FIND. This finds the *first* occurrence of any number *K*, passed from Extended BASIC in array *N(x)*. To call this subprogram, use the format

CALL LINK("FIND",*K*,*N(x)*)

To return control to Extended BASIC, use the format

CALL PEEK(−24560,HI,LO)

where the specified element number equals (HI*256)+LO.

Thus, if the element number was 32967, for instance, HI would equal 128 and LO would equal 199 (128*256=32768; 32768+199=32967).

AGAIN. This subprogram finds *any* occurrence of any number *K*, passed from Extended BASIC in array *N(x)*. Its format is much the same as FIND:

CALL LINK("AGAIN",*K*,*N(x)*)

where *K* is the number and *N(x)* is the array. Control is passed back to Extended BASIC by the same procedure as with FIND:

CALL PEEK(−24560,HI,LO)

where the element number equals (HI*256)+LO.

For either FIND or AGAIN, array *N(x)* must be created in Extended BASIC before the utility can be used.

Note in Program 2, the Extended BASIC demonstration, that both FIND and AGAIN are used. This insures that all instances of an array element are found. FIND locates the *first* instance of the element, while AGAIN locates *any* instance.

Program 1. TI FastSearch

```
 DEF FIND,AGAIN
NUMREF EQU >200C
TARGET EQU >A000
PLACE  EQU >A010
STATUS  EQU >837C
SAVE DATA >0000
FAC EQU >834A
GPLWS EQU >83E0
MYWS BSS >20
MYWS1 BSS >20
MYWS2 BSS >20
MSWS3 BSS >20
FIND
 LWPI MYWS
 LI R0,0
 LI R1,1
 BLWP @NUMREF  LOAD LINKED VALUE IN FAC
 LI R2,8
 LI R3,FAC
 LI R4,TARGET
BACK MOV *R3+,*R4+    PUT IT AT TARGET
 DECT R2
 JNE BACK
 LI R1,2
NEXT BLWP @NUMREF    LOAD ARRAY VALUE IN FAC
 LI R2,8
 LI R3,TARGET
 LI R4,>834B
COMP C *R3+,*R4+  COMPARE TARGET WITH  FAC
 JNE ADD1         JUMP NOT EQ TO ADD1
 DECT R2
  CI R2,0
 JNE COMP
 JMP MATCH
ADD1 INC R0
 CI R0,>03E8
 JLT NEXT
 JMP NO
MATCH MOV R0,@PLACE
 BLWP @RETURN
NO LI R0,>FFFF
 MOV R0,@PLACE
 BLWP @RETURN

******** AGAIN ***************

AGAIN
 LWPI MYWS
 MOV @PLACE,R0
 INC RO
 LI R1,2
NEXT1 BLWP @NUMREF    LOAD ARRAY VALUE IN FAC
 LI R2,8
 LI R3,TARGET
 LI R4,>834B
COMP1 C *R3+,*R4+  COMPARE TARGET WITH  FAC
 JNE ADD2         JUMP NOT EQ TO ADD1
 DECT R2
  CI R2,0
 JNE COMP1
 JMP MATCH1
ADD2 INC RO
 CI RO,>03E8
 JLT NEXT1
 JMP NO1
MATCH1 MOV RO,@PLACE
 BLWP @RETURN
NO1 LI RO,>FFFF
 MOV RO,@PLACE
 BLWP @RETURN

**** RETURN *************

RETURN DATA MYWS1,RET1
RET1 CLR RO
 MOVB RO,@STATUS *CLEAR STATUS
 LWPI GPLWS      *BRANCH TO GLWP
 B @>0070        *LOAD RETURN
 RT              *RETURN
 END
```

Program 2. Extended BASIC Search Demonstration

```
100 CALL CLEAR :: PRINT "LOADING ASSEMBLY SUBPROGRA
    M.":"ONE MOMENT PLEASE..."
110 REM THIS EXTENDED BASIC PROGRAM ILLUSTRATES THE
     USE OF THE MACHINE LANGUAGE SUBPROGRAMS CONTAI
    NED IN "SORTA".
120 CALL INIT :: REM INITIALIZE SYSTEM
130 CALL LOAD("DSK1.SORTA")
140 DIM ARRAY(1000)
150 CALL CLEAR :: PRINT "BUILDING A 1000 ELEMENT
      ARRAY.": :"ONE MOMENT PLEASE..."
160 FOR X=1 TO 1000 :: ARRAY(X)=X+3 :: NEXT X :: RE
    M BUILDS AN ARRAY
170 DISPLAY AT(3,2)ERASE ALL:"INPUT AN INTEGER BETW
    EEN  0 AND 1000: " :: ACCEPT AT(5,11)BEEP VALID
    ATE(NUMERIC):K
180 CALL CLEAR :: PRINT "SEARCHING.": :"ONE MOMENT
    PLEASE..."
```

```
190 CALL LINK("FIND",K,ARRAY())
200 CALL CLEAR
210 CALL PEEK(-24560,HI,LO):: CALL CONVERT(HI,LO,VA
    LUE):: IF VALUE=65535 THEN 250
220 PRINT "ARRAY POSITION IS: ";VALUE: :"N(";STR$(V
    ALUE);")=";K: :
230 CALL LINK("AGAIN",K,ARRAY())
240 GOTO 210
250 PRINT "THAT'S ALL": :"PRESS ENTER TO CONTINUE..
    ."
252 CALL KEY(3,KE,ST):: IF ST=0 THEN 252 ELSE 170
260 END
270 SUB CONVERT(HI,LO,VALUE)
280 VALUE=HI*256+LO
290 SUBEND
```

Expand TI BASIC with Mini Memory

Christopher Flynn

Seven assembly language routines add power and flexibility to TI BASIC. You don't have to know anything about assembly language programming to enter or use these utilities. Mini Memory command module and disk drive needed.

You've probably heard of Texas Instrument's Mini Memory command module, but you may not know what it does, or even what it can do, for your TI-99/4A. Though the module is no longer available retail (not much associated with the TI-99/4A is, unfortunately), there's still a good chance you'll be able to find one. TI-99/4A user groups, many of which still exist, are an excellent place to start. And there are various organizations, even a few companies, that have made it their job to distribute whatever TI hardware and software they come across.

What we'll do in this article is see what the Mini Memory module is, and some of the things you can do with it. We'll focus on how to use Mini Memory to add new commands to TI BASIC.

What It Is

First of all, the Mini Memory is a TI command module. This means that you can plug it right into the TI-99/4A console, just like *Parsec* or *Munchman.* You don't need any additional hardware to use the Mini Memory.

Inside the cartridge, there are three things:

- A 4K Read Only Memory (ROM) chip
- A 6K Graphics Read Only Memory (GROM) chip
- And 4K of Random Access Memory (RAM), with battery backup

The ROM and GROM chips contain programs written by Texas Instruments. These programs are quite different from those usually found in a command module.

Since the module contains 4K of RAM (the type of memory into which the computer can store information and from which it can read that information), you can store data or programs in the Mini Memory. Notice that the 4K RAM also has a built-in battery backup. Why? Well, whatever you store in the RAM will remain there—even when you turn off your computer. This gives you some interesting possibilities.

Also included with the Mini Memory is a cassette tape containing the *Line-by-Line* assembler. This is a program which lets you experiment with assembly language programming. Don't forget that your TI-99/4A contains a 16-bit CPU. Its assembly language capabilities are outstanding.

But What's It For?

There are a number of uses for the Mini Memory cartridge. Here are just a few:

- Storage for BASIC programs or files
- Learning about assembly language programming
- Running assembly language programs

What we'll do is briefly discuss these applications. This should give you a better idea of what the Mini Memory really can do.

Just another device. First of all, you can use the Mini Memory as though it were another device. In other words, you can make it work like a cassette tape recorder or even a disk drive. You can save a program in the cartridge (if the program isn't too long) with the SAVE command. You can get the program back again with the OLD command. Since there's a battery backup, the program remains in the cartridge's memory even after you turn off your machine.

The 4K RAM can also be used for saving data. You can use any type of file structure supported by TI BASIC. The PRINT command will write data into the Mini Memory. The INPUT command will read the data back. Even relative files work. These capabilities allow you to do things that you just couldn't do with a tape recorder.

Learning assembly. Another use for the Mini Memory cartridge is as an aid to learning assembly language. The *Line-by-Line* assembler is useful, but it's not a full-scale development system. The *Line-by-Line* assembler takes up most of the 4K of Mini Memory RAM. It has some other deficiencies as well. There's no provision for saving the source code for programs, and debugging can be a very tedious job. But at least you can try out assembly language to see if you like it. (If you decide to jump into assembly language at full speed, you'll need to find a copy of Texas Instrument's *Editor/Assembler.* With it, however, you'll have to have an expanded TI-99/4A, with a disk drive.)

Running assembly language programs. The Mini Memory module can be used to run assembly language programs written by others. People who have fully expanded TI-99/4A computers can develop programs you can use. In fact, there are several utility programs included in this article and other assembly language programs in this book that you can enter into the module.

TI BASIC Utilities

If you could expand TI BASIC, what features would you add? Subroutines such as CALL COLOR or CALL SOUND might immediately come to mind.

How about something different? We've developed assembly language utilities for:

- Sorting
- Screen formatting
- String-number conversion

You'll probably find these very useful. These utilities will come in handy when you're developing your own BASIC programs.

Before you continue, you need the following hardware and software.

- TI-99/4A console
- Disk drive
- Mini Memory command module
- Disk containing one or more of the utilities in object code form

The first three items are standard—you probably already have them. A disk which includes the utilities is something you don't have—at least not yet. You can, however, put these utilities on disk, with some typing.

Disk Utilities

Though most people probably use a cassette tape to store Mini Memory–created assembly language programs and routines, it is possible to load them from disk. Normally, only files created by Texas Instruments' *Editor/Assembler* package can be loaded from disk through the Mini Memory option of the command module. However, we've managed to duplicate the object code format that the *Editor/Assembler* uses. As long as you follow the directions below, you'll be able to load one or more of these utilities directly from disk.

- In TI BASIC, type in Program 1, "Binload," and save it to disk. This is your master object code maker, to which you'll add DATA statements specific to each utility.
- Load Program 1 from disk and list it. The last line, line 340, should be near the bottom of the screen. Select the appropriate set of DATA statements (Programs 2–9) and type them in. When you've finished entering one complete set (the number of DATA statement lines depends on the utility), save the program to disk, using the correct filename. These are

Program 2	SORTN/BAS
Program 3	SORT$/BAS
Program 4	DISP$/BAS
Program 5	ACCPT$/BAS
Program 6	SETWID/BAS
Program 7	TOSTR$/BAS
Program 8	TONUM$/BAS
Program 9	KERNAL/BAS

- Once you have these BASIC programs on disk, all you need to do to create machine language object files is to run each of them in turn. When you run SORTN/BAS, for instance, it will ask you for a filename. Simply enter SORTN and press ENTER, and the TI will automatically create an object file called SORTN on the disk in drive 1.

  Use the following filenames when creating object code files for these utilities. *It is important that you use these names.*

BASIC Filename	**Object Filename**
SORTN/BAS	SORTN
SORT$/BAS	SORT$
DISP$/BAS	DISP$
ACCPT$/BAS	ACCPT$
SETWID/BAS	SETWID
TOSTR$/BAS	TOSTR$
TONUM/BAS	TONUM
KERNAL/BAS	KERNAL

It's at this point that any typing errors in the DATA statements will show up. The last few numbers of each

DATA statement are a *checksum*. If there is an error in a particular DATA statement, a message to that effect will appear. Check that line and make any necessary corrections.

- Continue to run the BASIC programs until you have all the object files on disk.

Loading the Utilities

Before you can use the utilities, they must be stored in the Mini Memory module. Just follow these steps:

- Turn off the computer. If you have the Memory Expansion unit installed, remove it. Insert the Mini Memory module. Turn the computer back on and wait for the title screen to appear.
- Press any key to obtain the main menu. Select the Mini Memory option by pressing the 3 key.
- Select the *Re-Initialize* option from this menu by pressing the 3 key. This insures that the Mini Memory is cleared of any old programs or data. Hit FCTN-6 (PROC'D) if so instructed.
- You should see the Mini Memory menu on your screen again. Select the *Load and Run* option by pressing the 1 key.
- A prompt appears, asking for a filename. Type DSK1.*filename*, where *filename* is one of the object code filenames listed above. (Make sure that *filename* is on the disk currently in drive 1.)
- Continue to load and run the utilities, one at a time, until all you have on disk are placed in the Mini Memory's RAM.
- Press the ENTER key instead of typing in a filename. Mini Memory will ask you for a *Program Name*. Ignore this, and simply press FCTN-= (QUIT).

Note: KERNAL is not strictly a utility, but *it's vital that it be loaded into Mini Memory whenever you're using any or all of the utilities.*

Now you're ready to use the utilities in the Mini Memory module. The utilities will stay in the Mini Memory until they're erased (or until the battery runs out). Keep the disk with the object files in a safe place.

Using the Utilities

If you're used to including CALL SOUND or CALL COLOR in your programs, you already know how to use these utilities. In general, a program in the Mini Memory is called with a program statement like:

CALL LINK(*"program name"*,parameter list)

LINK may be a new BASIC word to you. Think of it as the way you tell BASIC to use a Mini Memory program.

The particular program that you want is given by *"program name"*. The program name can be up to six characters long. BASIC insists that the program name be enclosed in quotation marks.

Computers will hardly ever do anything useful until they're given some data or information. The same is true of the Mini Memory utilities. We want to send something from our BASIC program to Mini Memory, and we expect some results back. That's the purpose of the parameter list. It's nothing more than a fancy name for a list of variables or constants.

Notice how easy this is. You don't need to worry about the memory location of any machine language programs. You don't have to do any bothersome PEEKs and POKEs. All you do is go about the business of solving the particular problem you're working on. The TI-99/4A helps you out instead of getting in your way.

The following sections describe the utilities in general terms. Near the end of this article you'll find a reference section which contains a complete list of all the utilities as well as the specifics on each of the parameter lists.

Sorting

Many applications need some kind of sorting routine. Perhaps you want to alphabetize a list of names or compile a list of numbers. There are two programs to do this: **SORT$** sorts lists of strings, while **SORTN** sorts numbers.

These two sorts are in-memory sorts. That means that all the data to be sorted must be in BASIC's memory at the same time. Both of these sorts use the shell sort (named after a mathematician) algorithm. Fairly good sorting speed is obtained.

The "Utilities Reference" at the end of the article lists all the details. Let's look at a few examples of SORTN and SORT$.

SORTN. Sort, in ascending order, the first 100 entries in the array EXPENSES:

```
CALL LINK("SORTN",100,"A",EXPENSES( ))
```

Look at how the array is made available to Mini Memory. The parentheses after EXPENSES indicate that EXPENSES is an array. Furthermore, the entire array is being passed to SORTN. Both SORTN and SORT$ require a left and right parentheses pair right after the array name.

How do you get the answers back to BASIC? The sorted results are returned in EXPENSES. This means that the entries in EXPENSES have been reordered. The original sequence of data has been lost.

SORT$. NAMES$ contains a list of last names of those to whom you'll send Christmas cards. Each string in NAMES$ is ten characters long. Sort the first 50 entries in the list, in ascending order, on all ten characters:

```
CALL LINK("SORT$",50,10,1,10,"A",NAMES$( ))
```

You have to include four numbers in the parameter list:

50	number of entries
10	length of each string
1	starting position of the sort key subfield
10	length of the sort key subfield

SORT$. Suppose each string was 20 characters long. The last name starts in position 11 this time. Now the sort statement would look like this:

```
CALL LINK("SORT$",50,20,11,10,"A",NAMES$( ))
```

You've included the information to tell SORT$ that you want to sort on a subfield rather than on the entire string.

These sorts are very easy to use. Be careful of the way you type the array name in the CALL LINK statement. *Remember that SORT$ requires each string to be of the same length. Pad strings on the right with blanks if necessary.*

Screen Formatting

One of the facilities absent from TI BASIC is a way to PRINT or INPUT from a given row and column on the screen. TI BASIC is a scroller. It prints and accepts data from line 24. Everything above that is scrolled up to make room.

Isn't that good enough? For many applications it is. But sometimes you may want to design programs with a bit of graphics so that the screen resembles a paper form. (This is particularly true of data collection programs.) A scrolling display doesn't lend itself to this type of program design.

Many programmers have resorted to HCHAR or VCHAR to address the screen by row and column. Unfortunately, these commands can display only one character from a source string at a time. They're also a bit on the slow side.

With the right utilities in Mini Memory, you have a set of three subroutines designed to overcome these restrictions:

DISP$	display a string at a given row and column
ACCPT$	accept a string from a given row and column
SETWID	set screen width

DISP$ and ACCPT$ work only with strings. Moreover, they're line-oriented. DISP$ displays only one screen line of data at a time. Similarly, you can only type up to one line of data at a time with ACCPT$.

Before getting into details, let's take a quick look at SETWID. It will set the screen width to either 28 or 32 characters per line. You can use 32 characters per line only if you're using a monitor or television set that doesn't cut off the edges of your screen. SETWID works with ACCPT$ and DISP$. The line length for BASIC's PRINT, INPUT, and LIST statements is not affected.

The "Utilities Reference" section at the end of the article lists the details for ACCPT$ and DISP$. In both cases ROW, COL, and SIZE are part of the parameter list. ROW and COL are used to address the position on the screen where you want either to display or accept a string. ROW may range in value from 1 to 24. COL may range from 1 to either 28 or 32, depending on SETWID (the default is 28).

The top-left corner of the screen is always row 1, column 1. This is true regardless of what screen width is being used. The utility programs automatically adjust for the 28 or 32 character line length.

SIZE is a parameter that limits either the amount of data displayed or typed in. A few examples will clarify these subroutines.

**DISP$**. Display up to ten characters of ADDRESS$ at row 2, column 10:

```
CALL LINK("DISP$",2,10,10,ADDRESS$)
```

Refer to "Utilities Reference." SIZE is positive (10). Thus, ten characters will be erased from the screen beginning at row 2, column 10. Next, up to ten characters of ADDRESS$ will be displayed. Any characters to the right of ADDRESS$ on row 2 will not be disturbed.

DISP$. Do the same thing, but erase the rest of row 2 from column 10 to the end of the line:

```
CALL LINK("DISP$",2,10,-10,ADDRESS$)
```

If SIZE is negative, the remainder of the row is erased prior to displaying data.

That's all there is to DISP$. You could have used numeric variables in place of 2 and 10 for ROW and COL. Also, you could have used a string literal in the parameter list.

ACCPT$ is very similar to BASIC's INPUT statement. ACCPT$ is used to read information from the keyboard. However, there's no prompt option associated with ACCPT$. A message should be displayed with DISP$ first.

ACCPT$ has some unique features. There's only one editing key available—the left arrow (FCTN-S). It serves as a simple backspace key. Characters that you backspace over are erased. Second, ACCPT$ has an automatic ENTER feature. SIZE indicates the number of characters that ACCPT$ wants you to type. If you type in exactly that many, you don't have to press ENTER—ACCPT$ does it automatically. Third, ACCPT$ can read characters from the screen that you previously typed or that were put on the screen by an earlier DISP$ statement. Thus, you don't have to type frequently used responses over and over. Finally, ACCPT$ uses the underline as its cursor. This is to remind you that ACCPT$, not BASIC, is working.

ACCEPT$. Get a two-character state code from row 6, column 1:

```
CALL LINK("ACCPT$",6,1,2,STATE$)
```

Two characters will be erased from the screen starting at row 6, column 1. ACCPT$ will wait for you to type in a response. If you just hit ENTER, STATE$ will be set to the null string. If you type two characters, say, MD, ACCPT$ will supply the ENTER for you. STATE$ will then contain "MD".

ACCEPT$. What if you want VA most of the time, but you don't want to type it in every time?

```
100 CALL LINK("DISP$",6,1,2,"VA")
  .
  .
200 CALL LINK("ACCPT$",6,1,-2,STATE$)
  .
  .
300 GOTO 100
```

First, you put VA on the screen with DISP$. In the ACCPT$ statement, you use a negative number for SIZE. This tells ACCPT$ not to erase the field from the screen first—leave whatever's there alone. Looking at the screen, you'd see VA. If this is what you wanted, you'd just type ENTER. If you wanted a different state code, you'd type over VA and then hit ENTER.

From these simple examples, you can probably get a good idea of what ACCPT$ and DISP$ can do. Experiment with these two subroutines. They take a little time to get to know.

String/Number Conversion

As you've seen, ACCPT$ and DISP$ work very well for character strings. But what about numbers? Surely there are times when you'll want to display a number on the screen or when you'll want to input a number.

What you end up doing is using string variables for all of your input/output (I/O), then converting the strings to numbers and vice versa as the program requires.

TI BASIC has built-in functions to handle much of the work. STR$ converts a number to a string. VAL does the reverse, converting a string to a number.

Sometimes, though, you'll want more flexibility. For example, if you're displaying a column of numbers, you'd like to round the numbers and also make sure that the decimal points line up. You *can* do this in BASIC, but so you don't have to write such a program over and over again, TOSTR$ is included in the utility package.

BASIC's VAL function has a nasty quirk. It's designed to convert a character string, containing a series of digits, to a number. It does this just fine. However, if there's a nonnumeric character in the character string, VAL chokes and the program stops. TONUM$ is a subroutine which helps alleviate this problem. TONUM$ extracts the numeric portion of any character string that it is fed. This numeric portion can then be acted upon by VAL without any difficulty.

Look to "Utilities Reference" for all the details on these subroutines. First, though, let's go through a few examples.

TOSTR$. Convert the contents of RESULT to a character string rounded to two places after the decimal point:

```
CALL LINK("TOSTR$",RESULT,2,T$)
```

T$ will contain the character string. You could have used any string variable. The name does not matter.

Here are some typical results.

RESULT	**T$**	**LEN(T$)**
1.232	1.23	4
2.345	2.35	4
4	4.00	4
38.91299	38.91	5
0	.00	4
−3	−3.00	5
−2.345	−2.34	5
−1.232	−1.22	5

Notice that negative numbers are rounded toward zero.

If you wanted to print these numbers in a column, you'd pad them on the left with spaces. Then, you'd take the rightmost *N* characters with the SEG$ function.

TONUM$. The easiest subroutine to use. It has only one variable in its parameter list. Give TONUM$ a string, and it will return the numeric part, if any, of the string in the same variable:

CALL LINK("TONUM$",A$)

A$ Before	**A$ After**
1234	1234
1.78ZS6	1.78
−1.89.23	−1.89
#$!	0

Although it may be difficult to see from the example, TONUM$ skips over leading spaces.

TONUM$ should be used with a bit of caution. Suppose someone, in a payroll system, intended to type in 20000 for an annual salary. Due to some kind of error, the number was typed as Z0000. As the example shows, TONUM$ would convert this to a 0 annual salary. Be sure to include the necessary editing checks in your programs.

Sample Program

A sample program is shown in Program 10. It's designed to use most of the subroutines described in this article. Try out this program after you've loaded all eight subroutines into Mini Memory.

Utilities Reference

ACCPT$

Accept String At

CALL LINK("ACCPT$",ROW,COL,SIZE,A$)

ROW	Row number (1–24) on which the field will be displayed
COL	Column number (1–28 or 1–32)
SIZE	Size (1–28 or 1–32) of the field to be entered

If SIZE>0 then the field will be erased on the screen. If SIZE<0 then the screen will not be erased, thus making it possible to read the prior field contents. When SIZE characters have been typed, ACCPT$ returns control to BASIC. It's not necessary to press ENTER. The left arrow (FCTN-S) is the only active editing key; it functions as a simple backspace key.

DISP$

Display String At

CALL LINK("DISP$",ROW,COL,SIZE,A$)

ROW	Number of the row (1–24) on which the field is to be displayed
COL	Column number (1–28 or 1–32) of the field
SIZE	Size (1–28 or 1–32) of the field to be displayed

If SIZE>0 then the field will be erased prior to display. If SIZE<0 then the entire row from COL onward will be erased.

SETWID

Set Screen Width

CALL LINK("SETWID",N)

N	Set the screen width to N

N may be either 28 or 32, representing the number of characters per line. The TV or monitor must be capable of displaying 32 characters if this option is chosen. SETWID also affects the COL and SIZE parameters of ACCPT$ and DISP$.

SORT$

Sort String Array

CALL LINK("SORT$",N1,N2,N3,T$,A$( ))

N1	Number of records to sort
N2	Record length (must be fixed)
N3	Starting position (relative to 1) of sort key

N4	Length of sort key
T$	Type of sort—"A" is ascending, "D" is descending
A#$( )	Name of string array containing the records

Performs a shell sort of the records (or strings) in the array A$( ). The sorted records are returned in A$( ).

SORTN

Sort Numeric Array

CALL LINK("SORTN",N1,T$,A( ))

N1	Number of elements to sort
T$	Type of sort—"A" is ascending, "D" is descending
A( )	Name of the array containing elements to be sorted

Performs a shell sort of the elements in A( ). The sorted elements are returned in A( ).

TONUM$

Edit a String for Numeric Value

CALL LINK("TONUM$",A$)

A$	The string to be edited

Edits A$ for a numeric value. Removes leading blanks; returns all digits up to the first nonnumeric character. Returns the result in A$. Use the VAL function if the result is to be placed in a numeric variable.

TOSTR

Convert a Number to a String

CALL LINK("TOSTR",N1,N2,A$)

N1	The number to be converted
N2	The number of digits (0–10) to be included after the decimal point
A$	The result string

The number (N1) is rounded according to the number of digits after the decimal point desired (N2). The number is converted into a string and returned in A$. If N2 is zero, N1 is converted to an integer. If N2 is nonzero, trailing zeros (up to N2) are supplied.

Program 1. Binload

```
100 INPUT "NAME OF PROGRAM:":N$
110 OPEN #1:"DSK1."&N$,DISPLAY ,OUTPUT,FIXED 80
120 READ A$
```

```
130 IF A$="END" THEN 280
140 A=0
150 FOR I=1 TO LEN(A$)-5
160 A=A+ASC(SEG$(A$,I,1))
170 NEXT I
180 D=65536-A
190 H$=""
200 FOR I=1 TO 4
210 H=D-16*INT(D/16)
220 D=INT(D/16)
230 H$=SEG$("0123456789ABCDEF",H+1,1)&H$
240 NEXT I
250 IF H$<>SEG$(A$,LEN(A$)-4,4)THEN 310
260 PRINT #1:A$
270 GOTO 120
280 PRINT #1:":        99/4 AS"
290 CLOSE #1
300 END
310 PRINT "BAD DATA LINE:"
320 PRINT A$
330 CLOSE #1:DELETE
340 END
```

Program 2. SORTN/BAS

```
1000 DATA 0017A        974B6BC80BB713CB0201B0001B06
     A0B7118BC1E0B834AB04C07F2BBF
1010 DATA 974C8B0201B0002BD820B734FB714DB0202B714DB
     0420B604CB04E0B71427F2BFF
1020 DATA 974DEB9820B714EB7350B1602B0720B7142B0202B
     0001B0A12B81C2B11FD7F2C0F
1030 DATA 974F4B13FCBC802B7140BC0A0B7140B0812B1343B
     C802B7140BC187B61827F29CF
1040 DATA 9750AB0208B0001BC248BC2A0B7140BA289BC00AB
     0201B0003B0420B60447F2DDF
1050 DATA 97520B0201B0008BD861B8349B835BB0601B16FBB
     C009B0201B0003B04207F2E6F
1060 DATA 97536B6044B0420B601CB0A00BD060B837CBC820B
     7142B7142B1604B20607F2DBF
1070 DATA 9754CB7146B1604B1017B2060B7146B1614BC00AB
     0201B0003B0420B60407F311F
1080 DATA 97562B0201B0008BD861B835BB8349B0601B16FBB
     C009B0201B0003B04207F2E0F
1090 DATA 97578B6040B6260B7140B15C8B0588B0606B16C4B
     10B9B04C0BD800B837C7F2A7F
2000 DATA 9758EBC2E0B713CB045BBC80BB713EB0201B0001B
     06A0B7118B0200B00017F2B2F
2010 DATA 975A4B0201B0018B06A0B7126BC260B834AB0609B
     0201B0002B06A0B71187F2EFF
```

```
2020 DATA 975BABC820B714AB714AB1607B0200B0001B0201B
     0020B06A0B7126B10067F2F0F
2030 DATA 975D0B0200B0001B0201B001CB06A0B7126BC220B
     834AB0608B0201B00037F30EF
2040 DATA 975E6B06A0B7118B04C7BC1A0B834AB1501B0707B
     0746BC289B0A5ABA2887F272F
2050 DATA 975FCBC820B714AB714AB1301B05CABC820B714AB
     714AB1603B0205B00207F296F
2060 DATA 97612B1002B0205B001CB6148BC2E0B713EB045BB
     B060B735EB0420B60247F2C6F
2070 DATA 97628B0580B0602B15FBB045B7FA53F
2080 DATA 674B6SORTN 7FCFAF
2090 DATA END
```

Program 3. SORT$/BAS

```
1000 DATA 0013A        9737CBC80BB713CB0201B0001B06
     A0B7118BC1E0B834AB02017F2D2F
1010 DATA 9738EB0002B06A0B7118B0200B0001B0201B00FFB
     06A0B7126BC160B834A7F2D6F
1020 DATA 973A4B0201B0003B06A0B7118B0200B0001BC045B
     06A0B7126BC120B834A7F2F1F
1030 DATA 973BAB0604B0201B0004B06A0B7118B0200B0001B
     C045B06A0B7126BC0E07F2E6F
1040 DATA 973D0B834AB04C0B0201B0005BD820B734FB714DB
     0202B714DB0420B604C7F2B4F
1050 DATA 973E6B04E0B7142B9820B714EB7350B1602B0720B
     7142B0202B0001B0A127F2F2F
1060 DATA 973FCB81C2B11FDB13FCBC802B7140BC0A0B7140B
     0812B134EBC802B71407F266F
1070 DATA 97412BC187B6182B0208B0001BC248BC2A0B7140B
     A289BC009B0201B00067F2D1F
1080 DATA 97428B0202B714DBC805B714CB0420B604CBC00AB
     0201B0006B0202B724F7F2CBF
1090 DATA 9743EBC805B724EB0420B604CBC084BA083B0602B
     C003B04C1BD322B714E7F285F
2000 DATA 97454B2060B7144B1302B7320B734FB7322B7250B
     02C1B0602B0600B16F37F2F8F
2010 DATA 9746ABC820B7142B7142B1604B2060B7144B1615B
     1003B2060B7144B13117F30DF
2020 DATA 97480BC009B0201B0006B0202B724FB0420B6048B
     C00AB0201B0006B02027F312F
2030 DATA 97496B714DB0420B6048B6260B7140B15BDB0588B
     0606B16B9B10AEB04C07F2A1F
2040 DATA 974ACBD800B837CBC2E0B713CB045B7F8F3F
2050 DATA 6737CSORT$ 7FD23F
2060 DATA END
```

Program 4. DISP$/BAS

```
1000 DATA 00158         97630BC80BB713CB06A0B7594B04
     C0B0201B0004B0202B714D7F2F3F
1010 DATA 97642B0203B00FFBC803B714CB0420B604CBC1C7B
     1307BC00ABC085B02017F296F
1020 DATA 97658B2000B06A0B7620B1009BC086B8085B1501B
     C085BC00AB0201B20007F2E8F
1030 DATA 9766EB06A0B7620BC0A0B714CB8086B1501BC086B
     8085B1501BC085B02037F2AAF
1040 DATA 97684B714EBC00ABD073BB060B735EB0420B6024B
     0580B0602B15F8BC2E07F28DF
1050 DATA 9769AB713CB045BBC80BB713EB04C3B06A0B7700B
     9801B7360B13FBB98017F26EF
1060 DATA 976B0B7356B131FB9801B7355B130FB9801B7357B
     1AF2B9801B735FB1BEF7F264F
1070 DATA 976C6BB060B735EB0420B6024B0580B0583B80C2B
     15E7B1011BD060B73577F2B7F
1080 DATA 976DCBB060B735EB0420B6024B0600B0603B15DDB
     13DCB0580B0583B10D97F295F
1090 DATA 976F2BD060B7352B0420B6024BC2E0B713EB045BB
     0420B602CBD801B73527F2A4F
2000 DATA 97708BD820B7353B8374BD060B7351B0420B6024B
     0204B0200B0420B60207F30CF
2010 DATA 9771EBD060B837CB2060B7148B1312B0604B16F7B
     D060B7352B0420B60247F2C9F
2020 DATA 97734B0204B0200B0420B6020BD060B837CB2060B
     7148B1303B0604B16F77F2FFF
2030 DATA 9774AB10E1BD060B8375BD820B7353B837CB045BB
     C802B714CB1314B11137F294F
2040 DATA 97760B0201B714EB0420B6030BC0C2B78E0B735EB
     714DB0603B16FBBC0C27F283F
2050 DATA 97776B9823B714DB7357B1604B0620B714CB0603B
     16F8B045B7F4DCF
2060 DATA 67630DISP$ 7FD4FF
2070 DATA END
```

Program 5. ACCPT$/BAS

```
1000 DATA 00060         97788BC80BB713CB06A0B7594BC0
     86B8085B1501BC085BC2427F2B7F
1010 DATA 9779ABC1C7B1606BC00AB0201B2000B06A0B7620B
     1003BC00AB06A0B77587F2AEF
1020 DATA 977B0BC00ABC089B06A0B769EBC0C3B1605BC1C7B
     1607B04E0B714CB10047F262F
1030 DATA 977C6BC00ABC083B06A0B7758B04C0B0201B0004B
     0202B714DBD820B73537F2ADF
1040 DATA 977DCB837CB0420B6048BC2E0B713CB045B7F7EFF
1050 DATA 67788ACCPT$7FD26F
1060 DATA END
```

Program 6. SETWID/BAS

```
1000 DATA 00014         9797CBC80BB713CB0201B0001B06
     A0B7118BC060B834AB04E07F2D6F
1010 DATA 9798EB714A7FD84F
1020 DATA 6797CSETWID7FCD9F
1030 DATA END
```

Program 7. TOSTR$/BAS

```
1000 DATA 000EA         97892BC80BB713CB0201B0002B06
     A0B7118B0200B0000B02017F311F
1010 DATA 978A4B000AB06A0B7126BC220B834ABC048B0A11B
     C821B7368B835CB04E07F283F
1020 DATA 978BAB835EB04E0B8360B04E0B8362B04C0B0201B
     0001B0420B6044B04207F2D8F
1030 DATA 978D0B601CB0600BD820B7353B8355B0420B6018B
     0014BD820B7353B837C7F2C3F
1040 DATA 978E6B04C6BD1A0B8356B06C6B04C9BD260B8355B
     06C9B0229B8300BC0867F26CF
1050 DATA 978FCB0201B714EBDC79B0602B16FDBC086B0201B
     714EB9C60B735DB132B7F24CF
1060 DATA 97912B0602B16FBB0202B000FB0201B714EBA046B
     DC60B735BB0602B16FC7F291F
1070 DATA 97928B04E0B714CBC086B0201B714EB9C60B735AB
     1306B05A0B714CB06027F29AF
1080 DATA 9793EB16F9BD460B735AB0202B714DBA808B714CB
     C208B1302B05A0B714C7F272F
1090 DATA 97954B9820B714DB734FB1608B05A0B714CBD820B
     735BB714FB1002B02027F29EF
2000 DATA 9796AB7361B04C0B0201B0003B0420B6048BC2E0B
     713CB045B7F4EEF
2010 DATA 67892TOSTR$7FCF9F
2020 DATA END
```

Program 8. TONUM$/BAS

```
1000 DATA 000AA         977E8BC80BB713CB04C0B0201B00
     01B0202B00FFBC802B714C7F2AFF
1010 DATA 977FAB0202B714DB0420B604CBC160B714CB1333B
     04C6B04E0B714CB04C77F288F
1020 DATA 97810B0201B714EBD091B9802B7357B1604B0581B
     0605B16F9B1025BC0C17F2C6F
1030 DATA 97826B9802B7358B1303B9802B7359B1606B0605B
     131CB05A0B714CB05817F2D4F
1040 DATA 9783CBD091B9802B735AB1604BC186B160EB0586B
     1007B9802B735BB1A097F291F
1050 DATA 97852B9802B735CB1B06B0587B05A0B714CB0581B
     0605B16ECB0603BD4E07F28FF
```

```
1060 DATA 97868B714DBC1C7B1609B0200B0001BC800B714CB
     D820B735BB714EB02037F29DF
1070 DATA 9787EB714DB04C0B0201B0001BC083B0420B6048B
     C2E0B713CB045B7F3BDF
1080 DATA 677E8TONUM$7FCF1F
1090 DATA END
```

Program 9. KERNAL/BAS

```
1000 DATA 00264          97118B04C0B0420B6044B0420B60
     1CB1200B045BB8020B834A7F31EF
1010 DATA 9712AB1104B8801B834AB1101B045BB0200B1300B
     0420B6050B609CB77B87F2E9F
1020 DATA 97140B0001B0000B1000B4000B2000B0001B0001B
     534EB5445B5220B594F7F340F
1030 DATA 97156B5552B2053B454CB4543B5449B4F4EB2020B
     5330B3030B3000B00007F2FBF
1040 DATA 9716CB0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F399F
1050 DATA 97182B0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F3A8F
1060 DATA 97198B0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F3A1F
1070 DATA 971AEB0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F38CF
1080 DATA 971C4B0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F39BF
1090 DATA 971DAB0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F38DF
1100 DATA 971F0B0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F39CF
1110 DATA 97206B0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F3ABF
1120 DATA 9721CB0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F39DF
1130 DATA 97232B0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F3ACF
1140 DATA 97248B0000B0000B0000B000AB584BB4554B4248B
     434FB4E41B0000B00007F30EF
1150 DATA 9725EB0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F397F
1160 DATA 97274B0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F3A6F
1170 DATA 9728AB0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F398F
1180 DATA 972A0B0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F3A0F
1190 DATA 972B6B0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F399F
```

```
1200 DATA 972CCB0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F38BF
1210 DATA 972E2B0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F39AF
1220 DATA 972F8B0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F393F
1230 DATA 9730EB0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F39BF
1240 DATA 97324B0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00007F3AAF
1250 DATA 9733AB0000B0000B0000B0000B0000B0000B0000B
     0000B0000B0000B00017F39BF
1260 DATA 97350B44BFBBF00B0508B0D20B2B2DB2E30B3945B
     607FBFF05B2A2AB2A2A7F237F
1270 DATA 97366B2A00B3F32B3F05B3E32B3E05B3D32B3D05B
     3C32B3C05B3B32B3B057F279F
1280 DATA 67118KERNAL7FD05F
1290 DATA END
```

Program 10. Utility Demonstration

```
130 DIM SORTLIST(100),SORTLIST$(100)
140 CALL CLEAR
150 CALL LINK("DISP$",12,7,28,"MINI MEMORY DEMO")
160 FOR I=1 TO 1000
170 NEXT I
180 CALL CLEAR
190 CALL LINK("DISP$",2,1,28,"DO YOU WANT TO SORT:"
    )
200 CALL LINK("DISP$",4,4,28,"S  STRINGS")
210 CALL LINK("DISP$",6,4,28,"N  NUMBERS")
220 CALL LINK("DISP$",8,1,28,"ENTER YOUR SELECTION
     S")
230 REM
240 REM GET TYPE OF SORT
250 REM
260 CALL LINK("ACCPT$",8,23,-1,TYPE$)
270 IF TYPE$="S" THEN 300
280 IF TYPE$="N" THEN 300
290 GOTO 260
300 REM
310 REM GET NUMBER OF ITEMS
320 REM
330 CALL LINK("DISP$",12,1,28,"HOW MANY ITEMS TO SO
    RT  50")
340 CALL LINK("ACCPT$",12,25,-2,ITEMS$)
350 CALL LINK("TONUM$",ITEMS$)
360 ITEMS=VAL(ITEMS$)
370 IF ITEMS<3 THEN 330
380 IF ITEMS>99 THEN 330
```

```
390 REM
400 REM BUILD LIST OF ITEMS TO SORT
410 REM
420 CALL CLEAR
430 CALL LINK("DISP$",12,1,28,"WORKING ...")
440 IF TYPE$="N" THEN 540
450 FOR I=1 TO ITEMS
460 CALL LINK("DISP$",24,1,28,STR$(I))
470 T$=""
480 FOR J=1 TO 10
490 T$=T$&CHR$(25*RND+65)
500 NEXT J
510 SORTLIST$(I)=T$
520 NEXT I
530 GOTO 580
540 FOR I=1 TO ITEMS
550 CALL LINK("DISP$",24,1,28,STR$(I))
560 SORTLIST(I)=100*RND
570 NEXT I
580 REM
590 REM SORT THE LIST
600 REM
610 CALL CLEAR
620 CALL LINK("DISP$",12,1,28,"BEGIN SORT ...")
630 CALL SOUND(50,440,0)
640 IF TYPE$="N" THEN 670
650 CALL LINK("SORT$",ITEMS,10,1,10,"A",SORTLIST$()
    )
660 GOTO 680
670 CALL LINK("SORTN",ITEMS,"A",SORTLIST())
680 REM
690 REM DISPLAY THE RESULTS
700 REM
710 CALL SOUND(50,440,0)
720 IF TYPE$="N" THEN 770
730 FOR I=1 TO ITEMS
740 PRINT SORTLIST$(I)
750 NEXT I
760 GOTO 830
770 FOR I=1 TO ITEMS
780 CALL LINK("TOSTR$",SORTLIST(I),2,T$)
790 T$="          "&T$
800 T$=SEG$(T$,LEN(T$)-6,7)
810 PRINT T$
820 NEXT I
830 REM
840 REM ALL DONE
850 REM
860 FOR I=1 TO 2000
870 NEXT I
```

```
880 CALL CLEAR
890 CALL LINK("DISP$",12,1,28,"NEAT, HUH?")
900 FOR I=1 TO 1000
910 NEXT I
920 GOTO 100
```

TI File Management

Thomas W. Kirk

Assembly language provides power to almost any task. This article and its accompanying programs offer the building blocks you need to create impressive file management programs. There's even an example included, an automatic disk file cataloger. Extended BASIC, Memory Expansion, disk drive, and Editor/Assembler *required.*

Like many people, I've never been wildly enthusiastic about being neat. I have a hard time keeping track of my programs. And a super fast program doesn't save much time if it takes all night to find it. Trying to get organized, I decided to create a fast autodisk cataloger.

What I came up with is a set of general-purpose assembly language subprograms. These subprograms greatly enhance the powerful Extended BASIC language by making file access easy. In the process, I've managed to construct a working demonstration of these routines. By coincidence, this demo automatically catalogs files on any number of disks.

What We Need

All file management packages have several things in common. They need to create files, save files, load previously created (old) files, modify files, sort files, and find records. Commercial-quality file management systems are quite versatile and laden with features, but other than perfecting and expanding on these simple concepts, they're not much different from what's presented here.

Before making a file management program work reasonably fast on the TI-99/4A, you have to understand the machine's limitations. TI Extended BASIC is great for repeating an operation a few times, but it seems slow when it tries to repeat something numerous times. Loading, saving, and sorting files, and searching arrays are four such repetitive tasks. The subprograms I've developed use the flexibility of TI Extended BASIC to negate some of the language's limitations.

The first task—creating a file—is a repetitive process, but not one that's time critical. The major consideration is that the process is simple, concise, and, most important, foolproof. TI Extended BASIC excels in this.

The second and third tasks—saving and loading files—are another matter. The processes are totally repetitive and require no interaction. A slow routine in these areas is noticeable and frustrating. Assembly language, with the help of TI's utilities, offers the perfect solution.

Fast Loading and Saving

File is a term used to describe a grouping of *records*. For our purpose, we'll limit the discussion to records targeted for disk storage. In BASIC there are three steps to file access: (1) open the file, (2) input record from or print records to the file, and (3) close the file.

The TI Extended BASIC command to open a file looks like this:

```
OPEN #1:"DKS1.EXAMPLE",SEQUENTIAL, VARIABLE 80,
  DISPLAY, UPDATE
```

You'll notice that the statement includes information defining file parameters or attributes. These definitions are used by the operating system to determine which set of subroutines is used to create the file. The BASIC interpreter translates the attributes into a list of numbers and stores them in a Peripheral Access Block (PAB). Active PABs are stored in Video Display Processor Random Access Memory (VDP RAM). In assembly language, you cut out the middleman, the interpreter, and directly access the operating system by placing a PAB in VDP RAM, telling the computer where the PAB is located, and then branching to the Device Service Routine (DSR). The DSR reads the contents of the PBA, turns on the Read Only Memory (ROM) chips contained in the disk controller card, and then branches to the ROM subroutines required to execute the operation indicated in the PAB.

The PRINT command is used to *write* a record to an opened file, and the INPUT command is used to *read* a record from an opened file. Both commands modify byte 1 of the PAB. PRINT places a 3 in byte 1, while INPUT inserts a 2. Execution of a PRINT command entails creating a buffer in the VDP RAM, transferring target data to the buffer, then branching to the instructions contained in the disk controller card for the actual write-to-disk operation. Executing an INPUT command uses the same actions as a PRINT operation, except the

DSR calls the read-from-disk routines contained in the disk controller. Each of these commands is implemented in assembly language by placing the appropriate operation code in byte 1 of the PAB and invoking the DSR. The TI *Editor/Assembler* package contains a detailed description of how to access the file management system of the TI-99/4A.

The CLOSE instruction is identical to the OPEN operation, except PAB byte 1 contains the CLOSE code of 1. Of course, subsequent to setting the PAB for the close operation, the DSR is invoked.

Now that you understand how to get data to or from a disk, it's important to determine how to reference and assign data to variables created in BASIC. TI thoughtfully provided several utilities specifically designed to do this job. The STRREF, STRASG, NUMREF, and NUMASG (see "TI Fast-Search," elsewhere in this collection) allow capture of data from or data assignment to variables created in BASIC.

Combining the OPEN, repetitive variable reference, write, and CLOSE subtasks together makes a complete fast print-to-disk subroutine. While putting the OPEN, repetitive read and variable value assignment, and CLOSE subtasks together yields a fast read-from-disk subroutine.

Modifying a File

No one is perfect, and that's precisely why files often need some modification. TI Extended BASIC is perfectly suited for changing files. For the most part, the task is nonrepetitive, and the lavish editing functions available with the ACCEPT AT statement make writing and editing programs easy. In general, the Extended BASIC edit section of the program is self-explanatory. A full description of the operations procedure is included toward the end of this article.

Sorting Array Elements

If waiting for the disk operating system to load data is frustrating, then waiting for an interpreted language to sort an array of elements is boring. Only one thing is more boring than waiting for a sort routine to work its magic—reading about how it works. Let's just say that this sorting routine works. It *is* important to note, however, that this routine works much faster on a scrambled array than on a sorted one. Not only is it unnecessary to sort a sorted array, this routine will take up to three times as long to run the second time through. Appending additional scrambled elements to the array will in effect partially scramble it and therefore, to some degree, lessen run time.

Searching the Array

Searching an array (again, see "TI FastSearch" in this book) is a simple matter in Extended BASIC, but it's very time-consuming. An equivalent search in assembly language can execute in less than one fifth the time. The linkage to BASIC is also quite simple, making the implementation all the more attractive.

The search subprogram compares a parameter passed from BASIC with the contents of an array. Just a serial comparison. A counter tracks the element number of the most current comparison. Upon detection of a match, the counter value is passed back to BASIC via a PEEK command.

Building the Program

Type in Program 1, "QDEMO." It's written in TI Extended BASIC, so make sure you have that command module plugged into the console. Save it to disk using the filename QDEMO (you *can* assign it a different name).

Entering the three assembly language subprograms (the source code for these subprograms is listed in Programs 2, 3, and 4) requires more structure. Each subprogram is independent, though you'll need all three if you're using QDEMO. If you need (in creating your own file management program, for instance) only one or two of the subprograms, then by all means feel free to enter only those you'll use.

Of course, you need the *Editor/Assembler* package to enter, edit, assemble, and debug these subprograms. Save them to disk (the same disk which contains QDEMO) using these filenames.

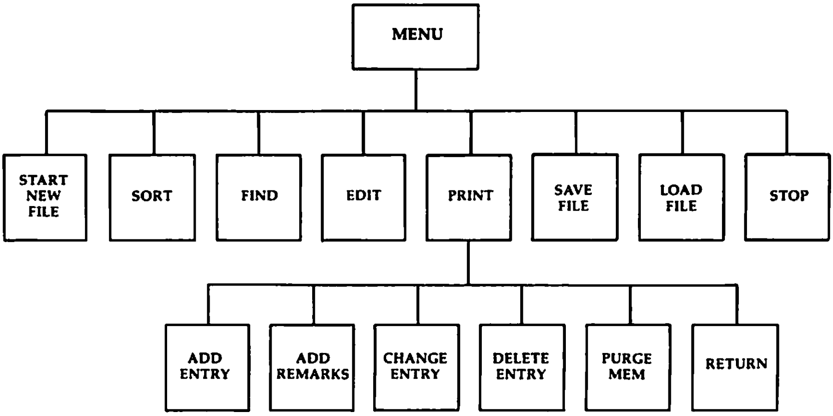
Program 2	QSORTA
Program 3	QSHA
Program 4	QLOADA

If you want QDEMO to load your assembly language subprograms, the object code must be assembled to the proper filename.

Operational Instructions

The first step in understanding how to use a relatively complex grouping of assembly language subprograms is to understand how they're organized. The figure shows the organization and logical flow of the file management program we're using to demonstrate the use of these assembly language file loader/saver, sort, and search subprograms.

Autocataloger Overview



Obviously, the program is menu driven, so the top block is the major select subroutine. Immediately following initialization, you'll be presented with a selection of operations to choose from:

1) Start new file
2) Sort file
3) Find entry
4) Edit file
5) Display file
6) Print file
7) Save file
8) Load file
9) Terminate session

Start new file. Choosing this selection allows you to quickly load the indexes of your disk library. Two important actions take place in this section. First, the subroutine assigns your disk indexes to array positions, and, second, a selective counter assigns a pointer value for access to the REMARKS relative file to each array element. The actions are fully automatic. Merely follow the prompts.

The *Start new file* section is also used to append existing files which have been loaded using option 8. In either circumstance, the limit is 500 records. Five hundred records just about exhaust the TI-99/4A's string memory capability.

Sort file. This section does just what it says—it sorts the file. After selection, you'll be asked the question *# OF CHARACTERS: 6.* Your answer determines the resolution of the sort routine. Responding with 1 means that the sort will look at only the first character of the target string, while an entry of 15 forces the sort routine to look at more character positions. Theoretically, the sort should go faster the lower the resolution. I find that the default value of 6 is adequate for this application.

Find entry. Pressing 3 yields access to the *Find entry* section of the program. Normally, you should go through the *Edit file* section prior to entry of this subroutine. I recommend this because, even though an array entry is created in the *Start new file* section, the REMARKS file has yet to be created. Attempting to access a nonexistent file causes a file error. This error is trapped, and so doesn't stop the program. But it's always best to enter the *Edit* section, add remarks, then return to *Find entry.*

To find an entry, just follow the prompts. The machine calls the search routine, then recovers the relative record pointed to by the captured array element.

Edit file. This selection offers you several additional options, all presented on the screen:

1) Add entry
2) Add remark
3) Change entry
4) Delete entry
5) Return to main menu

Add entry allows individual records to be added. Use this section to append your file to include newly acquired programs added to a disk index currently in the file.

Add remark lets you add remarks either in sequential order, starting at the beginning of the file, or to a particular title you've specified. In either case, the remark is in FIXED 40, IN-

TERNAL format. Because of this, the remark is limited to 28 characters. Ample prompts are provided.

Change entry. Both the title entry and the remark can be changed if *Change entry* is selected. Normally, this section should not be used to change a title that has no remark entered with it.

Delete entry deletes a specified entry and replaces it with a *!!*num* tag. This tag insures that the relative record number is not wasted. The next add or append action will assign the specified title the relative record number contained in the tag.

Return to main menu returns you to the master menu.

Display file. If selection 5 is chosen, the file will print to the screen. Stop and start the listing by pressing any key. Pressing FCTN-9 returns you to the main menu.

Print file. Prints to the specified device the contents of the file and its remark. There is no autopaging provided.

Save file. The fast write-to-disk subroutine described earlier. A SEQUENTIAL, FIXED 40, DISPLAY file is created. This file contains the title (variable length), relative record number (three digits), and disk name (variable length), as read from J$( ). The format of the record is

"TITLE*NUM*DISKNAME"

The number of records saved is stored in the first record of this sequential file. When this selection is chosen, you'll be asked to designate a target filename. The routine overwrites existing files of the same name.

Load old file. Causes a designated FIXED 40, DISPLAY, SEQUENTIAL file to be loaded. The number of records to be read is stored in record 1 of the target file. The read-in data is stored in J$( ). You're asked for a target file immediately after entering this section.

End this session. Stops the program. All data not saved to disk is lost.

Dry Run

We've covered the particulars of the subprogram set, and discussed the organization of the demonstration program and considerations for entering the code. Now it's time to try it out.

- Turn the computer on with the Extended BASIC module plugged in.
- Place your program disk in drive 1.

- Type RUN "DSK1.QDEMO".
- The TI-99/4A will come to life and say it's initializing. The program is loading the object files.
- The main menu appears. Take your program disk out of drive 1 and put it away.
- Select menu item 1 (*Start new listing*).
- Designate a target drive.
- Place a disk you wish to catalog in the designated drive. Follow the prompts. The program will load the index, and ask if you want to load another. Continue repeating this until you've entered all your disks or reach the 500-record limit.

  Note: I suggest you start off loading just one index, then run through the rest of the procedures to insure that the program works. This will save the frustration of loading 400 records, only to find that you made one mistake typing in QDEMO.
- Select N to return to the main menu when you're finished loading indexes.
- Select 2 from the main menu (*Sort file*).
- Follow the prompts. The TI-99/4A will sort the file, then return to the main menu.
- Select 7 on the main menu (*Save file*).
- Insert a new, initialized disk in the drive of your choice.
- Answer the prompts. Your file will be saved and the program will return to the main menu.
- Select 4 on the main menu (*Edit*).
- Select 2 on the submenu (*Add remarks*).
- Answer the prompt to cause the remarks to begin at entry 1.
- Enter your remarks. The program will return to the *Edit* submenu automatically when all records are remarked.
- Select 5 on the submenu (*Return to main menu*).
- You're on your own.

You have the utility routines and a useful autocatalog program. I hope you find these tools useful. I believe they're one implementation of the fundamental building blocks needed to create any reasonably fast TI Extended BASIC file management program.

Acknowledgments

The DSR subroutine used in this article was provided courtesy of Texas Instruments, Incorporated.

Assembly Language Routines Documentation

In order to use these assembly language subprograms to create your own file management programs in Extended BASIC, you need to know even more about them. What follows should give you an idea of what each of the three subprograms does, how it does it, and what you can use it for.

Assign/Reference

Object filename: QLOADA

Contains the routines:

PULL1. This routine reads records off a disk and assigns them to a targeted set of array elements. Its calling linkage format is

CALL LINK("PULL1",*"DSK−.NAME"*,N$( ),J)

where *DSK−.NAME* is the target drive and program name string, N$( ) is the array to which elements will be assigned, and J is the number of elements to read.

This is an autoreturn routine. No data is returned to Extended BASIC.

PUSH1. Writes a designated number of records, captured from a targeted array, to disk. Its calling format is

CALL LINK("PUSH1",*"DSK−.NAME"*,N$( ),J)

where *DSK−.NAME* is the target drive and filename, N$( ) is the designated array, and J is the number of elements to write. As with PULL1, this returns automatically to BASIC, with no data returned.

Before calling either of these routines, the target array must be dimensioned in TI Extended BASIC.

Other considerations are that the number of records to be transferred to memory must be within the limits of the DIM statement. The size limits of the VDP RAM must also be observed. The file attributes are SEQUENTIAL FIXED 40, DISPLAY, and UPDATE.

This subprogram (QLOADA) is designed for general-purpose application. It can be linked to any TI Extended BASIC program which needs a fast array load/save subroutine.

String Search

Object filename: QSHA

Contains the routines:

FIND. Sequentially searches the target array for the *first* occurrence of a designated string argument. The calling format for FIND is

CALL LINK ("FIND","*STRING*",N$( ))

where *STRING* is the designated string, and N$( ) is the target array.

The array position is returned at >A000 (decimal −24560). CALL PEEK(−24560,HI,LO)::VAL=HI*256+LO renders the proper array position available to the TI Extended BASIC caller.

AGAIN. Finds the *second and all subsequent* occurrences of a designated string argument in a target array. Its calling linkage format is

CALL LINK ("AGAIN","*STRING*",N$( ))

where *STRING* is the designated string and N$( ) is the target array.

The return linkage is identical to FIND's.

The target array must be dimensioned in TI Extended BASIC.

This subprogam (QSHA) is callable only from TI Extended BASIC. With appropriate modifications this subprogram is callable from TI BASIC.

The search routine is tailored to the specific application contained in QDEMO. Additional applications may require array elements to be terminated with an "*". The "*" serves to decouple the several front characters of a string from the trailing characters of the same string. An example might be

"FRONT*TRAILINGCHARACTERS"

The search routine will find "FRONT" and ignore the remaining characters.

Sort Routine

Object filename: EQSORTA

Contains the routines:

CLEAR. This routine sequentially clears array elements by installing null strings. The clearing action starts at element

1 of a specified string and ends at a specified element. The calling format is

CALL LINK("CLEAR",N$( ),*LENGTH*,1)

where N$( ) is the target array, *LENGTH* is the number of elements to be cleared, and 1 is a *required* filler. This is an autoreturn routine, with no data returned to Extended BASIC.

FILTER. Strips a specified number of string array elements of all null string elements. Call it with

CALL LINK("FILTER",N$( ),*LENGTH*,1)

where N$( ) is the target array, *LENGTH* is the number of elements the routine will filter, and 1 is a required filler. Again, no data returned.

QSORT. Sorts a specified number of string array elements. The order is ascending. The resolution of the sort is selectable. Its format is

CALL LINK("QSORT",N$( ),*LENGTH*,*RES*)

where N$( ) is the target array, *LENGTH* the number of elements to sort, and *RES* the number of characters that will be included in the sort. Autoreturn routine, no data returned.

The target array must be dimensioned in TI Extended BASIC prior to call. If elements have been deleted, use FILTER to remove any null strings. If null strings are not removed, the sort routine will place them at the beginning of the array.

This sort routine works reasonably fast on a scrambled array. Sorting a sorted array takes about three times as long as the same sort on a scrambled array. It must be called only from TI Extended BASIC. Minor modifications are required to link to TI BASIC.

Program 1. QDEMO

Enter this program in Extended BASIC.

```
100 CALL CLEAR :: PRINT : : : :"***DISK DIRECTORY D
    EMO***": : :
110 PRINT "INITIALIZING.":"ONE MOMENT PLEASE..."
120 CALL INIT :: CALL LOAD("DSK1.QSORTA"):: CALL LO
    AD("DSK1.QLOADA"):: CALL LOAD("DSK1.QSHA")
130 GOTO 170 :: DIM J$(520):: A$,AK$,B$,DIS$,DM$,DO
    $,FI$,FLEN$,JTAG$,JTAG1$,N1$,NAME$,NM$,PN$,RE$
140 A,C,D,E,F,HI,IC,IC1,J,J1,JE,JJ,K,KK,L,LO,LS1,LS
    2,MSB,NONE,P,PLA,S,SK,SK1,TK,X
150 CALL CLEAR :: CALL CONVERT :: CALL ERR :: CALL
    HOLD :: CALL INIT :: CALL KEY :: CALL LINK :: C
    ALL LOAD :: CALL PEEK
```

```
160 !@P-
170 REM START PROGRAM TARGET
180 ON ERROR 2010 :: J=1 :: OPEN #1:"MEMINIT",FIXED
     10 :: CLOSE #1 :: ON WARNING NEXT :: REM ON ER
    ROR 2010
190 DIM J$(750)
200 ON ERROR 2010
210 CALL CLEAR :: PRINT "***********MENU**********
    **": : :"1. START NEW LISTING.":"2. SORT LISTIN
    G.":"3. FIND ENTRY.":"4. EDIT LISTING."
220 PRINT "5. DISPLAY LIST.":"6. PRINT LIST.":"7. S
    AVE LIST.":"8. LOAD LIST.":"9. TERMINATE THIS S
    ESSION."
230 SK=0 :: PLA=0 :: A=1
240 PRINT : :"ENTER THE NUMBER FOR YOUR   SELECTION
     PLEASE."
250 CALL KEY(3,K,S)
260 IF K=49 THEN CALL CLEAR :: GOTO 430
270 IF K=50 THEN CALL CLEAR :: GOTO 380
280 IF K=51 THEN CALL CLEAR :: GOTO 360
290 IF K=52 THEN CALL CLEAR :: GOTO 370
300 IF K=53 THEN CALL CLEAR :: GOTO 390
310 IF K=54 THEN CALL CLEAR :: GOTO 400
320 IF K=55 THEN CALL CLEAR :: GOTO 410
330 IF K=56 THEN CALL CLEAR :: GOTO 420
340 IF K=57 THEN CALL CLEAR :: PRINT "*END FILE DIR
    ECTORY PROGRAM*": : : :"GOOD DAY!" :: END
350 GOTO 250
360 GOSUB 810 :: CLOSE #1 :: GOTO 210 :: REM POINTE
    R TO FIND J$() ELEMENT
370 GOSUB 1130 :: GOTO 210
380 GOSUB 680 :: GOTO 210
390 GOSUB 760 :: GOTO 210
400 GOSUB 1890 :: GOTO 210
410 GOSUB 1030 :: GOTO 210
420 GOSUB 1070 :: GOTO 210
430 IF J<>1 THEN J=J-1
440 DISPLAY AT(1,1):"**START NEW DIRECTORY FILE**":
     : :"MASTER DISK[1-3]?";A :: ACCEPT AT(4,19)VAL
    IDATE(DIGIT)SIZE(-1):A
450 DISPLAY AT(7,1):"PLACE IN DRIVE #";A;"A DISK YO
    U WISH TO INCLUDE  IN THE DIRECTORY.": : :"
     ELEMENTS LOADED": :"THE LIMIT IS 500."
460 DISPLAY AT(24,1):"PRESS ENTER TO CONTINUE."
470 CALL KEY(3,K,S):: IF K<>13 THEN 470
480 IF A<1 THEN 440
490 IF A>3 THEN 440
500 OPEN #2:"DSK"&STR$(A)&".",INPUT ,RELATIVE,INTER
    NAL
510 INPUT #2:A$
```

```
520 DISPLAY AT(23(1):"LOADING DSK";STR (A);".";A$;"
     INDEX.":"ONE MOMENT PLEASE..."
530 GOSUB 2040
540 IF J=500 THEN DISPLAY AT(12,1):"¡¡¡BUFFER FULL¡
    ¡¡" :: COPO 610
550 INPUT #2:B$
120 DISPLAY AT(12,1)SIZE(4):J
570 IF LEN(B$)=0 THEN 610
580 IF PLA<1000 THEN GOSUB 2090
590 J$(JJ)=B$&"*"&JTAG1$&"*"&A$
600 GOTO 530
610 CLOSE #2
620 DISPLAY AT(23,1):"ENTER ANOTHER DISK? Y/N":"
                                            "
630 CALL KEY(3,K,S)
640 IF K=89 THEN 440
650 IF K=78 THEN CALL LINK("FILTER",J$(),J,0):: CAL
    L PEEK(-24576,HI,LO):: J=(HI*256+LO):: J$(J+1)
    ="ZZZZZ" :: GOTO 210
660 GOTO 630
670 GOTO 210
680 MSB=6 :: DISPLAY AT(1,4)ERASE ALL:"****SORTING
    SECTION****" :: J$(J+1)="ZZZZZ"
690 DISPLAY AT(6,1):"ENTER":"# OF CHARACTERS:";MSB
    :: ACCEPT AT(7,18)BEEP SIZE(-2):MSB
700 DISPLAY AT(20,1):"SORTING";J;: :"THE PROCESS SH
    OULD TAKE":"ABOUT";J*.15;"SECONDS."
710 CALL LINK("FILTER",J$(),J,0)
720 CALL LINK("FIND","ZZZZZ",J$()):: CALL CONVERT(P
    LA):: J=PLA
730 CALL LINK("QSORT",J$(),J,MSB)
740 RETURN
750 REM ***DISPLAY SECTION***
760 KK=J :: FOR X=1 TO KK
770 CALL KEY(5,K,S):: IF K=15 THEN RETURN
780 IF S<>0 AND S<>-1 THEN CALL HOLD
790 PRINT J$(X)
800 NEXT X :: CALL HOLD :: GOTO 210 :: REM FINISH D
    ISPLAY SECTION
810 DISPLAY AT(1,1)ERASE ALL:"  *FIND ENTRY SECTION
    *" :: GOSUB 2180
820 GOSUB 890
830 GOSUB 2150 :: INPUT #1,REC TK:RE$ :: DISPLAY AT
    (14,1):"REMARKS:":RE$
840 DISPLAY AT(24,1):"FIND ANOTHER? Y/N"
850 CALL KEY(3,K,S)
860 IF K=89 THEN 820
870 IF K=78 THEN RETURN
880 GOTO 850
890 DISPLAY AT(11,1):"ENTER YOUR PROGRAM:" :: ACCEP
    T AT(11,20):NAME$
```

```
900 DO$=NAME$&"*"
910 CALL LINK("FIND",DO$,J$())
920 CALL CONVERT(PLA):: IF PLA>1000 THEN DISPLAY AT
    (11,1):"NO MATCH" :: GOTO 980
930 DISPLAY AT(11,1):J$(PLA)
940 DISPLAY AT(24,1):"FIND DUPLICATE? Y/N":
950 CALL KEY(3,K,S):: IF K=78 THEN GOTO 980
960 IF K=89 THEN CALL LINK("AGAIN",DO$,J$()):: GOTO
     920
970 GOTO 950
980 DISPLAY AT(24,1):"FIND ANOTHER? Y/N."
990 CALL KEY(3,K,S)
1000 IF K=78 THEN RETURN
1010 IF K=89 THEN GOTO 890
1020 GOTO 990
1030 PRINT "  ***SAVE FILE SECTION***": : : : :: PR
     INT "ENTER TARGET FILE NAME:" :: INPUT N1$ ::
     REM THIS IS LOAD FILE SUBROUTINE
1040 CALL LINK("FIND","ZZZZZ",J$()):: CALL CONVERT(
     PLA):: J=PLA :: PRINT "SAVING FILE:";N1$: :J;"
     ELEMENTS.": :"ONE MOMENT PLEASE..."
1050 J$(1)=STR$(J):: CALL LINK("PUSH1",N1$,J$(),J):
     : J$(1)=""
1060 RETURN
1070 PRINT "  ***LOAD FILE SECTION***": : :: PRINT
     "ENTER TARGET FILE NAME:" :: INPUT N1$ :: REM
     THIS IS LOAD FILE SECTION
1080 OPEN #1:N1$,FIXED 40 :: INPUT #1:FLEN$
1090 IF ASC(FLEN$)<48 OR ASC(FLEN$)>57 THEN FLEN$=S
     EG$(FLEN$,2,LEN(FLEN$)-1)
1100 J=VAL(FLEN$):: CLOSE #1 :: PRINT "LOADING ";J;
     "ELEMENTS.": :"ONE MOMENT PLEASE..."
1110 CALL LINK("PULL1",N1$,J$(),J):: J$(1)="" :: JJ
     =J
1120 PRINT : : :"FINISHED" :: J$(J)="ZZZZZ" :: RETU
     RN
1130 CALL CLEAR :: PRINT "***EDIT LIST SECTION***":
      : : : :"      *****MENU*****": : :"1. ADD ENTR
     Y":"2. ADD REMARKS":"3. CHANGE ENTRY"
1140 PRINT "4. DELETE ENTRY.":"5. PURGE MEMORY.":"6
     . MAIN MENU": :"PRESS NUMBER FOR SELECTION  PL
     EASE."
1150 CALL KEY(3,K,S)
1160 IF K=49 THEN CALL CLEAR :: GOTO 1230
1170 IF K=50 THEN CALL CLEAR :: GOTO 1380
1180 IF K=51 THEN CALL CLEAR :: GOTO 1560
1190 IF K=52 THEN CALL CLEAR :: GOTO 1680
1200 IF K=53 THEN CALL CLEAR :: GOTO 1830
1210 IF K=54 THEN CALL CLEAR :: RETURN
1220 GOTO 1150
```

```
1230 DISPLAY AT(1,1)ERASE ALL:"****ADD ENTRY SECTIO
     N****"
1240 GOSUB 2180
1250 GOSUB 2090 :: REM GET ENTRY NUMBER IF AVAILABL
     E
1260 DISPLAY AT(11,1):"ENTER PROGRAM NAME:": :"ENTE
     R DISK NAME:": :"ENTER REMARK:"
1270 ACCEPT AT(11,20):NM$ :: ACCEPT AT(13,17):DM$ :
     : ACCEPT AT(16,1):RE$
1280 IF PLA>1000 THEN JE=J :: GOTO 1300
1290 IF PLA<1000 THEN JE=PLA :: GOTO 1310
1300 GOSUB 2040 :: REM GET JTAG1$
1310 J$(JE)=NM$&"*"&JTAG1$&"*"&DM$ :: DISPLAY AT(23
     ,1):J$(JE);" ADDED":"ADD ANOTHER? Y/N"
1320 PLA=JE :: GOSUB 2150 :: PRINT #1,REC TK:RE$
1330 CALL KEY(3,K,S)
1340 IF K=89 THEN 1250
1350 IF K=78 THEN CLOSE #1 :: J$(J+1)="ZZZZZ" :: RE
     TURN
1360 GOTO 1330
1370 RETURN
1380 DISPLAY AT(1,1)ERASE ALL:" ***ADD REMARKS***"
1390 GOSUB 2180 :: REM CALL CLEAR
1400 DISPLAY AT(24,1):"START AT 1ST RECORD? Y/N"
1410 CALL KEY(3,K,S):: IF K=89 THEN PLA=2 :: GOTO 1
     450
1420 IF K=78 THEN DISPLAY AT(24,1):"ENTER START TIT
     LE:" :: ACCEPT AT(24,19):NAME$ :: CALL LINK("F
     IND",NAME$,J$()):: CALL CONVERT(PLA):: GOTO 14
     40
1430 GOTO 1410
1440 IF PLA>1000 THEN DISPLAY AT(24,1):"NO MATCH. T
     RY AGAIN!" :: GOTO 1420
1450 FOR X=PLA TO JJ-1
1460 DISPLAY AT(3,1):"RECORDS LEFT:";(JJ-1)-PLA
1470 PLA=X :: GOSUB 2150
1480 DISPLAY AT(10,1): :"ENTER REMARK FOR:": :J$(PL
     A):: ACCEPT AT(15,1)SIZE(-80):RE$
1490 PRINT #1,REC TK:RE$
1500 DISPLAY AT(24,1):"ADD ANOTHER REMARK? Y/N"
1510 CALL KEY(3,K,S):: IF K=78 THEN X=JJ :: GOTO 15
     40
1520 IF K=89 THEN 1540
1530 GOTO 1510
1540 NEXT X :: CLOSE #1
1550 RETURN
1560 REM CHANGE ENTRY
1570 GOSUB 890 :: GOSUB 2150 :: CALL CLEAR :: GOSUB
      2180
1580 INPUT #1,REC TK:RE$ :: CLOSE #1
```

```
1590 DISPLAY AT(12,1):"MAKE YOUR CHANGE.": : :J$(PL
     A): :"REMARKS:":RE$
1600 ACCEPT AT(15,1)SIZE(-80):J$(PLA):: ACCEPT AT(1
     8,1)SIZE(-80):RE$
1610 DISPLAY AT(24,1):NONE :: GOSUB 2180 :: PRINT #
     1,REC TK:RE$ :: CLOSE #1
1620 DISPLAY AT(24,1):"CHANGE ANOTHER? Y/N"
1630 CALL KEY(3,K,S)
1640 IF K=89 THEN 1570
1650 IF K=78 THEN 1130
1660 GOTO 1630
1670 GOTO 210
1680 DISPLAY AT(1,1)ERASE ALL:"**DELETE ENTRY SECTI
     ON**" :: DISPLAY AT(11,1):"PROGRAM NAME? " ::
     ACCEPT AT(11,15):NAME$
1690 P=0 :: L=0 :: DO$=NAME$&"*"
1700 CALL LINK("FIND",DO$,J$())
1710 CALL PEEK(-24560,HI,LO):: PLA=HI*256+LO :: IF
     PLA>1000 THEN DISPLAY AT(11,1):"NO MATCH" :: G
     OTO 1790
1720 DISPLAY AT(11,1):J$(PLA):: DISPLAY AT(24,1):"A
     RE YOU SURE? Y/N"
1730 CALL KEY(3,K,S):: IF K=89 THEN GOTO 1750
1740 IF K=78 THEN GOTO 210 ELSE GOTO 1730
1750 DISPLAY AT(12,1):"DELETED" :: P=POS(J$(PLA),"*
     ",2):: L=LEN(J$(PLA)):: J$(PLA)="!!"&SEG$(J$(P
     LA),P,4)
1760 PLA=0
1770 GOTO 1790
1780 GOTO 1730
1790 DISPLAY AT(24,1):"DELETE ANOTHER? Y/N":
1800 CALL KEY(3,K,S):: IF K=89 THEN 1680
1810 IF K=78 THEN 1130
1820 GOTO 1800
1830 DISPLAY AT(1,1):"****PURGE FILE****": : :"ARE
     YOU SURE?  Y/N"
1840 CALL KEY(3,K,S):: IF K=89 THEN 1870
1850 IF K=78 THEN 1130
1860 GOTO 1840
1870 CALL LINK("CLEAR",J$(),J+1,1):: J=1 :: GOTO 11
     30
1880 REM END PURGE SECTION
1890 DISPLAY AT(1,1):"***PRINT FILE SECTION***"
1900 IC=0
1905 DISPLAY AT(23,1):"ENTER PRINTER NAME:":"RS232.
     BA=1200.TW" :: ACCEPT AT(24,1)SIZE(-20):PN$
1910 OPEN #2:PN$ :: PRINT #2:TAB(10);"FILE NAME
      DISK NAME":TAB(10);"----------------------"
     :: PRINT #2:
1920 GOSUB 2180
1930 FOR X=2 TO J :: IF LEN(J$(X))=0 THEN 1990
```

```
1940 PLA=X :: GOSUB 2150 :: INPUT #1,REC TK:RE$
1950 D=POS(J$(X),"*",1):: DIS$=SEG$(J$(X),1,LEN(J$(
     X))+(D-LEN(J$(X))))
1960 FI$=RPT$("*",15-LEN(DIS$)):: LS1=LEN(J$(X))::
     LS2=LEN(DIS$)
1970 PRINT #2:TAB(10);DIS$&FI$&SEG$(J$(X),LS2+4,LS1
     -(LS2+2))&"**********"&RE$
1980 IC1=IC+1 :: IC=IC1 :: IF IC=60 THEN PRINT #1:
     : : : : : :
1990 NEXT X :: CLOSE #1 :: CLOSE #2
2000 RETURN
2010 CALL ERR(C,D,E,F):: ON ERROR 2020 :: CLOSE #1
2020 PRINT :"FILE ERROR FILE ERROR!!" :: ON ERROR 2
     025 :: CLOSE #2 :: FOR X=1 TO 200 :: NEXT X
2025 RETURN 200
2030 REM  ROUTINE TO SET ARRAY NUMBER
2040 J1=J+1 :: J=J1 :: JJ=J
2050 JTAG$=STR$(JJ)
2060 IF LEN(JTAG$)=1 THEN JTAG$="00"&JTAG$
2070 IF LEN(JTAG$)=2 THEN JTAG$="0"&JTAG$
2080 JTAG1$=JTAG$ :: RETURN
2090 REM ARRAY FILL SUBROUTINE
2100 JTAG1$=JTAG$ :: SK1=SK+1 :: SK=SK1 :: IF SK>1
     THEN 2130
2110 CALL LINK("FIND","!!",J$()):: CALL CONVERT(PLA
     ):: IF PLA>1000 THEN RETURN
2120 JTAG1$=SEG$(J$(PLA),4,3):: JJ=PLA :: RETURN
2130 CALL LINK("AGAIN","!!",J$()):: CALL CONVERT(PL
     A):: IF PLA>1000 THEN RETURN
2140 JTAG1$=SEG$(J$(PLA),4,3):: JJ=PLA :: RETURN
2150 IF PLA>1000 THEN RETURN :: REM GET RECORD NUMB
     ER
2160 IF LEN(J$(PLA))=0 THEN RETURN :: REM GET RECOR
     D NUMBER
2170 AK$=SEG$(J$(PLA),POS(J$(PLA),"*",1)+1,3):: TK=
     VAL(AK$):: RETURN :: REM GET RECORD NUMBER
2180 DISPLAY AT(5,1):"ENTER FILE NAME:";N1$ :: ACCE
     PT AT(5,17)SIZE(-15):N1$
2190 DISPLAY AT(7,1):"FILE ";N1$;"REM":"TO BE ACCES
     SED":"PRESS ENTER TO CONTINUE" :: CALL HOLD ::
      OPEN #1:N1$&"REM",RELATIVE 500,INTERNAL
2200 RETURN
2210 !@P+
2220 SUB HOLD
2230 DISPLAY AT(24,1):"PRESS ANY KEY TO CONTINUE"
2240 CALL KEY(3,K,S):: IF S=0 OR S=-1 THEN 2240
2250 SUBEND
2260 SUB CONVERT(PLA)
2270 CALL PEEK(-24560,HI,LO)
2280 PLA=HI*256+LO
2290 SUBEND
```

Program 2. QSORTA

Enter this, and the following two source code listings, with the Editor/Assembler.

```
*****EQSORT*****

 DEF CLEAR,FILTER,QSORT

******EQUATES***********

CFI    EQU >12B8
FAC    EQU >834A
GPLWS  EQU >83E0
NUMREF EQU >200C
STATUS EQU >837C
STRASG EQU >2010
STRREF EQU >2014
TGT    EQU >A000

******DATA**************

ARG1  DATA >0
ARG2  DATA >0
ARG3  DATA >0
BLN   DATA >0020,>2020,>2020
COUNT DATA >0
CT1   DATA >0
FF    DATA >FF
I     DATA >0
J     DATA >0
KK    DATA >0
LB    DATA >0
N     DATA >0
ONE   DATA >1
P     DATA >0
RB    DATA >0
X     DATA >0
ZERO  DATA >0

******BLOCKS************

ABUF  BSS >100
ABUF1 BSS >100
LP    BSS >100
MYWS  BSS >20
RP    BSS >100
RTWS  BSS >20
SWS1  BSS >20
SWS2  BSS >20
SWS3  BSS >20
SWS4  BSS >20
```

```
SWS5   BSS >20
TBUF   BSS >100

******BLWPS***********


EXIJ    DATA SWS1,EXIJ1
EXJI    DATA SWS2,EXJI1
GABJ    DATA SWS3,GABJ1
GABI    DATA SWS3,GABI1
INIT    DATA SWS3,INIT1
SREF    DATA SWS4,SREF1
SASI    DATA SWS5,SASI1
SASJ    DATA SWS5,SASJ1
RETURN DATA RTWS,RET1
XML     DATA GPLWS,XML1

*********************

CLEAR
 BLWP  @INIT
CL1     MOV @I,R0
 LI     R1,1
 LI     R2,BLN
 BLWP  @STRASG
 INC   @I
 C      @I,@ARG2
 JNE   CL1
 BLWP  @RETURN

*********************

FILTER
 LWPI  MYWS
 BLWP  @INIT
 MOV   @ONE,@I
 MOV   @ONE,@KK
 LI     R1,1

REPT   MOV @I,R0
 BLWP  @SREF
 CB     @TBUF,@ZERO
 JEQ   F548

 INC   @KK
 MOV   @KK,R0
 LI     R2,TBUF
 BLWP  @STRASG

F548   INC @I
```

```
 C      @I,@ARG2
 JLT    REPT

 MOV    @KK,@TGT

 BLWP   @RETURN

*********************

QSORT
 LWPI   MYWS
 BLWP   @INIT
 LI     R8,>2
 MOV    R8,@P
*
 MOV    @ONE,@LP(R8)
*
 MOV    @ARG2,@RP(R8)
*
S230    CI R8,0
 JLT    HOME
 JEQ    HOME
 JMP    S240
HOME    BLWP @RETURN
S240    MOV @LP(R8),@LB
*
 MOV    @RP(R8),@RB
*
 DECT   R8
S270    C @RB,@LB
 JLT    S230
 JEQ    S230
*
 MOV    @LB,@I
*
 MOV    @RB,@J
*
 BLWP   @SREF
*
S310    C @J,@ONE
 JLT    S350
*
**************************************************

S320    LI R2,TBUF+1
 BLWP   @GABJ
 MOV    @ARG3,R5
 LI     R3,ABUF+1
M1      CB *R2+,*R3+
 JGT    S350
 JNE    CON
 DEC    R5
 CI     R5,0
 JEQ    S350
 JMP    M1

*****************************************************

CON     DEC @J

 JMP    S310
*
S350    C @J,@I
 JGT    S380
 BLWP   @SASI
*

        JMP S500
*
S380    BLWP @EXIJ
*
 INC    @I
*
S400    C @I,@ARG2
 JGT    S440
*
*****************************************************

 LI     R2,TBUF+1
 BLWP   @GABI
 MOV    @ARG3,R5
 LI     R3,ABUF+1
M2      CB *R3+,*R2+
 JGT    S440
 JNE    S420
 DEC    R5
 CI     R5,0
 JEQ    S420

 JMP    M2

*****************************************************

S420    INC @I
*
 JMP    S400
*
S440    C @J,@I
 JLT    S480
 JEQ    S480
```

```
*
 BLWP   @EXJI
*
 DEC    @J
*
 JMP    S320
*
S480    BLWP @SASJ
*
 MOV    @J,@I
*
S500    INCT R8
*
 MOV    @RB,R1
 MOV    @I,R4
 MOV    @I,R3
 MOV    @LB,R2
 S      @R3,@R1
 S      @R2,@R4
 C      @R4,@R1
 JGT    S560
 JEQ    S560
*
 MOV    @I,R1
 INC    R1
 MOV    R1,@LP(R8)


 MOV    @RB,@RP(R8)
*
 MOV    @I,@RB
 DEC    @RB
*
S550    B @S270
*
S560    MOV @LB,@LP(R8)
*
 MOV    @I,R1
 DEC    R1
 MOV    R1,@RP(R8)
*
 MOV    @I,@LB
 INC    @LB
*
 B      @S270


*******SUBROUTINES***************
```

```
**INIT**

INIT1
 MOV    @ONE,@COUNT
 CLR    @LB
 CLR    @RB
 CLR    @TGT
 MOV    @ONE,@I
 MOV    @ONE,@J
 LI     R2,LP
 BL     @CLRA
 LI     R2,RP
 BL     @CLRA
 LI     R0,0
 LI     R1,2
 BLWP   @NUMREF
 LWPI   GPLWS
 BL     @GSV
 BL     @CFI
 MOV    @FAC,@ARG2
 BL     @GPOL
 LWPI   SWS3

 LI     R1,3
 BLWP   @NUMREF
 LWPI   GPLWS
 BL     @GSV
 BL     @CFI
 MOV    @FAC,@ARG3
 BL     @GPOL
 LWPI   SWS3

 RTWP

**SREF**

SREF1
 LI     R2,TBUF
 BL     @CLRT
 LI     R5,>FFFF
 MOV    @I,R0
 LI     R1,1
 LI     R2,TBUF
 MOV    R5,*R2
 BLWP   @STRREF
 RTWP

*****GABJ

GABJ1
```

```
 LI     R2,ABUF
 BL     @CLRA
 LI     R5,>FFFF
 MOV    @J,R0
 LI     R1,1
 LI     R2,ABUF
 MOV    R5,*R2
 BLWP   @STRREF
 RTWP

****GABI

GABI1
 LI     R2,ABUF
 BL     @CLRA
 LI     R5,>FFFF
 MOV    @I,R0
 LI     R1,1
 LI     R2,ABUF
 MOV    R5,*R2
 BLWP   @STRREF
 RTWP

**SASA**

SASI1
 MOV    @I,R0
 LI     R1,1
 LI     R2,TBUF
 BLWP   @STRASG
 RTWP

SASJ1
 MOV    @J,R0
 LI     R1,1
 LI     R2,TBUF
 BLWP   @STRASG
 RTWP

************************

**EXIJ**

EXIJ1
 LI     R5,>FFFF
 MOV    @J,R0
 LI     R1,1
 LI     R2,ABUF1
 MOV    R5,*R2
 BLWP   @STRREF
 MOV    @I,R0
 BLWP   @STRASG
 RTWP

**EXJI**

EXJI1
 LI     R5,>FFFF
 MOV    @I,R0
 LI     R1,1
 LI     R2,ABUF1
 MOV    R5,*R2
 BLWP   @STRREF
 MOV    @J,R0
 BLWP   @STRASG
 RTWP

********CLRT**************

CLRT
 LI     R1,>100
 LI     R3,>FFFF
CLRT1   MOV R3,*R2+
 DECT   R1
 CI     R1,0
 JNE    CLRT1
 RT

*****

CLRA
 LI     R1,>100
 LI     R3,>0
CLRA1   MOV R3,*R2+
 DECT   R1
 CI     R1,0
 JNE    CLRA1
 RT

***XML***************

XML1    BL @>12B8
 RTWP

**RETURN*************

RET1
 CLR    R0
 MOVB   R0,@STATUS
 LWPI   GPLWS
 B      @>0070

*********GPL PAD SAVE**********

GSV     LI R1,>100
 LI     R2,ABUF1
 LI     R3,>8300
AG      MOV *R3+,*R2+
 DECT   R1
 JNE    AG
 RT

*********GPL PAD RET ***********

GPOL    LI R1,>100
 LI     R2,ABUF1
 LI     R3,>8300
AG1     MOV *R2+,*R3+
 DECT   R1
 JNE    AG1
 RT

********************************

 END
```

Program 3. QSHA

```
 DEF FIND,AGAIN
*
*DATA STATEMENTS*****
*
COUNT     DATA >0000
N188      DATA >0200
ONE       DATA >0001
SAVE      DATA >0000
SIZE      DATA >0000
*
*EQUATES***********
*
BUF       EQU >A012
BUF1      EQU >A100
GPLWS     EQU >83E0
NUMREF    EQU >200C
PLACE     EQU >A010
SAVE1     EQU >A200
SAVE2     EQU >A202
STATUS    EQU >837C
STRREF    EQU >2014
TARGET    EQU >A000
*
*MEMORY BLOCKS******
*
MYWS      BSS >20
MYWS1     BSS >20
MYWS2     BSS >20
MSWS3     BSS >20
*
*****FIND**********
```

```
*
FIND
         LWPI MYWS
         CLR @COUNT
AG1      INC @COUNT
         CLR R1
         CLR R2
         CLR R3
         CLR R4
         MOV R1,@BUF+2
         MOV R1,@BUF1+2
         LI R2,>FF00
         MOVB R2,@BUF          LOADS STRING LENGTH INTO FIRST BYTE OF BUFFER
         LI R2,BUF             LOADS BUFFER ADDRESS IN R2
         LI R0,0
         LI R1,1
         BLWP @STRREF          LOAD LINKED VALUE IN BUF
         LI R1,2
NEXT     MOV @COUNT,R0
         LI R2,>FF00
         MOVB R2,@BUF1
         LI R2,BUF1
         BLWP @STRREF          LOAD ARRAY VALUE IN BUF1
         MOVB @BUF,R2          LOAD STRING LENGTH IN R2
         SWPB R2               PUT THE VALUE IN THE LSB
         LI R3,BUF+1
         LI R4,BUF1+1
COMP     CB *R3+,*R4+          COMPARE TARGET WITH  FAC
         JNE ADD1              JUMP NOT EQ TO ADD1
         DEC R2
         CI R2,0
         JNE COMP
         JMP MATCH
ADD1     INC @COUNT
         C @COUNT,@N188
         JLT NEXT
         JMP NO
MATCH    MOV R0,@PLACE
         BLWP @RETURN
NO       LI R0,>FFFF
         MOV R0,@PLACE
         BLWP @RETURN

********AGAIN*************

AGAIN    LWPI MYWS
         BL @AG1

********RETURN************

RETURN   DATA MYWS1,RET1
RET1     CLR R0
 MOVB    R0,@STATUS            *CLEAR STATUS
 LWPI    GPLWS                 *BRANCH TO GLWP
 B       @>0070                *LOAD RETURN
 RT                            *RETURN
 END
```

Program 4. QLOADA

```
 DEF PUSH1,PULL1
*******************
ABUF1   BSS >100
ARK     BSS >20          LEVEL 1 WS
ARK1    BSS >20          LEVEL 2 WS
```

```
ARK2    BSS >20              LEVEL 3 WS
RTWS    BSS >20
ARG1    BSS >2
ARG2    BSS >2
ARG3    BSS >2
BUFF    BSS >100             BUFFER FOR VDP DATA
BUF1    BSS >100
OPEN    DATA >0000
CLOSE   DATA >0100
READ    DATA >0200
WRITE   DATA >0300
INPUT   DATA >0000           1C00
OUTPUT  DATA >0000           1A00
CODE    DATA >0000           HOLDER FOR OPCODE
MODE    DATA >0000
FILL    DATA >FFFF
BSZ     DATA >0000
START   DATA >0000
VSTAR   DATA >0000

*******************

PABBUF  DATA >0000
PAB     DATA >0F80
KEEP    DATA >0000
CFI     EQU >12B8
DIR     EQU >8310
FAC     EQU >834A
NUMREF  EQU >200C
STATUS  EQU >837C
STRASG  EQU >2010
STRREF  EQU >2014

********************

SCLEN   EQU >8355
SCNAME  EQU >8356
CRULST  EQU >83D0
SADDR   EQU >83D2
GPLWS   EQU >83E0
FLGPTR  DATA 0
SVGPRT  DATA 0
SAVCRU  DATA 0
SAVENT  DATA 0
SAVLEN  DATA 0
SAVPAB  DATA 0
SAVVER  DATA 0
DLNKWS  DATA 0,0,0,0,0
TYPE    DATA 0,0,0,0,0,0,0,0,0,0,0
C100    DATA 100
```

```
H20    EQU $
H2000  DATA >2000
DECMAL TEXT '.'
HAA    BYTE >AA
SAVRTN DATA 0
VMBW   EQU >2024
VMBR   EQU >202C
VSBW   EQU >2020
VSBR   EQU >2028
KSCAN  EQU >201C
NAMBUF BSS >15

*********************
*            0 1  2 3  4 5
PDATA  DATA >0000,>0022,>2828
       DATA >0000,>000F
 TEXT  'DSK1.NAME1234567890'
 EVEN
MYREG  BSS >20
BUFFER BSS >100

******BLWPS****

RETURN DATA RTWS,RET1
INIT   DATA ARK1,INIT1
OPER   DATA ARK1,OPER1
ROOM   DATA ARK,ROOM1
RROOM  DATA ARK,RROOM1
FILE   DATA ARK1,FILE1

***WRITE ARRAY TO DISK****

PUSH1
 LWPI   MYREG
 BLWP   @INIT
 MOV    @OUTPUT,@MODE
 BLWP   @ROOM             ROUTINE TO GET SPACE AT >1000
 MOV    @WRITE,@CODE
 BLWP   @OPER
 LI     R0,1
 MOV    R0,@BSZ
PU1     BL @CLRA          CLEAR
 LI     R2,BUF1           SET BUF ADD FOR STR
 MOV    @FILL,*R2
 LI     R1,2
 MOV    @BSZ,R0
 BLWP   @STRREF
 LI     R0,>0022
 LI     R1,BUF1
 LI     R2,>28            40 CHARACTERS
```

```
BLWP    @VMBW
BLWP    @FILE
INC     @BSZ
C       @BSZ,@ARG3
JNE     PU1
BLWP    @RETURN            RETURN TO BASIC

*******READ RECORD TO ARRAY*******

PULL1
 LWPI    MYREG
 BLWP    @INIT              CALL INIT
 MOV     @INPUT,@MODE
 BLWP    @ROOM
 MOV     @READ,@CODE
 BLWP    @OPER
 LI      R0,1
 MOV     R0,@BSZ
P1       BLWP @FILE         READ FILE
 MOV     R1,@BUF1
 LI      R0,>0022
 LI      R1,BUF1
 LI      R2,>28             40 CHARACTERS
 BLWP    @VMBR              EXCHANGE FILE VALUE
 MOV     @BSZ,R0
 LI      R1,2
 LI      R2,BUF1
 BLWP    @STRASG
 INC     @BSZ
 C       @BSZ,@ARG3
 JNE     P1
 LI      R1,2
 BLWP    @RETURN

****INITIALIZE*********

INIT1
 CLR     @MODE
 CLR     @CODE
 CLR     @BSZ
 CLR     @ARG1
 CLR     @ARG2
 CLR     @ARG3
 CLR     R0
 BL      @STR
 LI      R0,0
 LI      R1,3
 BLWP    @NUMREF
 LWPI    GPLWS
 BL      @GSV
 BL     @CFI
 MOV    @FAC,@ARG3
 BL     @GPOL
 LWPI   ARK1
 RTWP

****PUT LINKED STR IN PAB*******

STR
 LI     R0,0             SIMPLE VARIABLE
 LI     R1,1             FIRST VARIABLE
 LI     R4,>FFFF
 LI     R2,PDATA+9       PUT IT IN PAB
 MOVB   R4,*R2           SET BUFF LENGTH
 BLWP   @STRREF          DO IT
 RT

*****RETURN*****************************************

RET1    BLWP @RROOM
 CLR    R0
 MOVB   R0,@STATUS
 LWPI   GPLWS
 B      @>0070
 RT

*******LOAD OP CODE*******

OPER1   LI R0,>0000
 MOV    @CODE,R1
 BLWP   @VSBW
 LI     R0,>0001
 MOV    @MODE,R1
 BLWP   @VSBW
 RTWP

***********ROOM*********

ROOM1   LI R0,>0000      this routine
 LI     R1,BUFF          gets a space
 LI     R2,>100          vdp ram @1000
 BLWP   @VMBR
 LI     R0,>0000
 LI     R1,PDATA         LOAD PAB
 LI     R2,>20
 BLWP   @VMBW            ****************
 MOV    @OPEN,@CODE      DO OPEN
 BLWP   @OPER
 BLWP   @FILE
 RTWP
```

```
******PUT DATA BACK IN VDP****

RROOM1   MOV @CLOSE,@CODE ********
 BLWP    @OPER            PUT CLOSE IN PAB
 BLWP    @FILE            ********
 LI      R0,>0000
 LI      R1,BUFF
 LI      R2,>100
 BLWP    @VMBW
 RTWP

*******EXECUTE*FILE*OPERATION****

FILE1    LI R6,>0009      POINTER TO NAME
 MOV     R6,@SCNAME
 BLWP    @DSRLNK
 DATA    8
 RTWP

*********GPL PAD SAVE**********

GSV      LI R1,>100
 LI      R2,ABUF1
 LI      R3,>8300
AG       MOV *R3+,*R2+
 DECT    R1
 JNE     AG
 RT

*********GPL PAD RET ***********

GPOL     LI R1,>100
 LI      R2,ABUF1
 LI      R3,>8300
AG1      MOV *R2+,*R3+
 DECT    R1
 JNE     AG1
 RT

***********CLRA*****************

CLRA     LI R3,>100
 CLR     R0
 LI      R1,BUF1
CLAR1    MOV R0,*R1+
 DECT    R3
 CI      R3,0
 JNE     CLAR1
 RT

**********************************
```

```
****DSRLNK***DSRLNK***DSRLNK***DSRLNK***

DSRLNK   DATA DLNKWS,DLENTR
DLENTR   MOV *R14+,R5
 SZCB    @H20,R15
 MOV     @SCNAME,R0
 MOV     R0,R9
 AI      R9,-8
 BLWP    @VSBR
 MOVB    R1,R3
 SRL     R3,8
 SETO    R4
 LI      R2,NAMBUF
LNK$LP   INC R0
 INC     R4
 C       R4,R3
 JEQ     LNK$LN
 BLWP    @VSBR
 MOVB    R1,*R2+
 CB      R1,@DECMAL
 JNE     LNK$LP
LNK$LN   MOV R4,R4
 JEQ     LNKERR
 CI      R4,7
 JGT     LNKERR
 CLR     @CRULST
 MOV     R4,@SCLEN-1
 MOV     R4,@SAVLEN
 INC     R4
 A       R4,@SCNAME
 MOV     @SCNAME,@SAVPAB

*******SEARCH ROMS**************

SROM     LWPI GPLWS
 CLR     R1
 LI      R12,>0F00
NOROM    MOV R12,R12
 JEQ     NOOFF
 SBZ     0
NOOFF    AI R12,>0100
 CLR     @CRULST
 CI      R12,>2000
 JEQ     NODSR
 MOV     R12,@CRULST
 SBO     0
 LI      R2,>4000
 CB      *R2,@HAA
 JNE     NOROM
 A       @TYPE,R2
 JMP     SGO2
SGO      MOV @SADDR,R2
 SBO     0
SGO2     MOV *R2,R2
 JEQ     NOROM
 MOV     R2,@SADDR
 INCT    R2
 MOV     *R2+,R9
 MOVB    @SCLEN,R5
 JEQ     NAME2
 CB      R5,*R2+
 JNE     SGO
 SRL     R5,8
 LI      R6,NAMBUF
NAME1    CB *R6+,*R2+
 JNE     SGO
 DEC     R5
 JNE     NAME1
NAME2    INC R1
 MOV     R1,@SAVVER
 MOV     R9,@SAVENT
 MOV     R12,@SAVCRU
 BL      *R9
 JMP     SGO
 SBZ     0
 LWPI    DLNKWS
 MOV     R9,R0
 BLWP    @VSBR
 SRL     R1,13
 JNE     IOERR
 RTWP
NODSR    LWPI DLNKWS
LNKERR   CLR R1
IOERR    SWPB R1
 MOVB    R1,*R13
 SOCB    @H20,R15
 RTWP

*********************************

 END
```

Appendix

A Beginner's Guide to Typing In Programs

What Is a Program?

A computer cannot perform any task by itself. Like a car without gas, a computer has *potential*, but without a program, it isn't going anywhere. The programs published in this book are written in a computer language called BASIC. BASIC is easy to learn and is built into the TI.

BASIC Programs

Computers can be picky. Unlike the English language, which is full of ambiguities, BASIC usually has only one right way of stating something. Every letter, character, or number is significant. A common mistake is substituting a letter such as *O* for the numeral *0*, a lowercase *l* for the numeral *1*, or an uppercase *B* for the numeral *8*. Also, you must enter all punctuation such as colons and commas just as they appear in the book. Spacing can be important. To be safe, type in the listings *exactly* as they appear. Enter all programs with the ALPHA LOCK on (in the down position). Release the ALPHA LOCK to enter lowercase text.

About DATA Statements

Some programs contain a section or sections of DATA statements. These lines provide information needed by the program; they are especially sensitive to errors.

If a single number in any one DATA statement is mistyped, your machine could lock up, or crash. The keyboard may seem dead, and the screen may go blank. Don't panic—no damage is done. To regain control, you have to turn off your computer, then turn it back on. This will erase whatever program was in memory, *so always save a copy of your program before you run it.* If your computer crashes, you can load the program and look for your mistake.

Sometimes a mistyped DATA statement will cause an error message when the program is run. The error message may refer to the program line that READs the data. *The error is still in the DATA statements, though.*

Get to Know Your Machine

You should familiarize yourself with your computer before attempting to type in a program. Learn the statements you use to store and retrieve programs from tape or disk. You'll want to save a copy of your program so that you won't have to type it in every time you want to use it. Learn to use your machine's editing functions. How do you change a line if you made a mistake? You can always retype the line, but you at least need to know how to backspace. It's all explained in your owner's manual.

A Quick Review

1. Type in the program, a line at a time, in order. Press ENTER at the end of each line.
2. Check the line you've typed against the line in the book. You can check the entire program again if you get an error when you run the program.
3. Make sure you've typed all the DATA statements and CALL CHAR statements correctly.

Index

To order your copy of *TI Collection, Vol. 2 Disk*, call our toll-free US order line: 1-800-346-6767 (in NY 212-887-8525) or send your prepaid order to:

TI Collection, Vol. 2 Disk
COMPUTE! Publications
P.O. Box 5038
F.D.R. Station
New York, NY 10150

Send _____ copies of *TI Collection, Vol. 2 Disk* at $12.95 per copy.

All orders must be prepaid (check, charge, or money order). NC residents add 4.5% sales tax.

Subtotal $__________

Shipping and Handling: $2.00/disk $__________

Sales tax (if applicable) $__________

Total payment enclosed $__________

All payments must be in U.S. funds.

☐ Payment enclosed
☐ Charge ☐ Visa ☐ MasterCard ☐ American Express

Acct. No. ______________________________ Exp. Date ________
(Required)

Signature ____

Name ____

Address ____

City __________________________ State ________ Zip ________

Please allow 4-5 weeks for delivery.

46103623

All New TI Software

There are fewer and fewer places where you can find useful software for your TI-99/4A. Commercial software is almost nonexistent for the computer, and most magazines no longer cover the machine.

COMPUTE! Publications is different. We continue to publish advanced programs and insightful articles on the TI-99/4A. This book, *COMPUTE!'s TI Collection, Volume Two,* is our latest TI-specific publication.

Filled with more than two dozen never-before-published programs and articles, *COMPUTE!'s TI Collection, Volume Two* contains a wealth of software for every TI user. Applications, tutorials, inside information, games, and unique utilities make this book a special value.

Just some of the things you'll find inside include:

- "Memo," a simple word processor which doesn't need a disk drive or use long tape files for storage
- "MitiCalc," an easy-to-use spreadsheet, perfect for your personal accounting and financial management
- "Expand TI BASIC," a combination tutorial and utility which adds powerful new capabilities to your computer with assembly language subroutines
- "Labyrinth," a mind-bending puzzle where you find yourself in a 125-room maze
- "Spelling Tutor," an educational tool that teaches spelling through computer-generated speech
- "TI File Management," an impressive collection of assembly language routines which you can use to create your own powerful file management program
- And much more

All the programs in this book are ready to type in and use. The writing is clear, concise, and easy to understand.

Do you want new software for your TI-99/4A? You've got a great collection right in your hands.

All the programs in this book are available on a companion disk. See the coupon in the back for details.

ISBN 0-87455-036-X