COMPUTE!'s TI Collection

VOLUME ONE

With over 30 TI-99/4A games, applications, utilities, and tutorials—most never before published—this anthology contains the best from COMPUTE! Publications. Arcade-style games, data base management, a sophisticated character editor, and much more provide something for every TI user.

A **COMPUTE! Books** Publication $12.95

COMPUTE!'s TI COLLECTION

VOLUME ONE

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

The following article was originally published in *COMPUTE!* magazine, copyright 1982, Small Systems Services, Inc.: "All Sorts of BASIC Sorts" (December).

The following articles were originally published in *COMPUTE!* magazine, copyright 1983, Small Systems Services, Inc.: "Programming the TI" (January); "Writing Your Own Games" (February); "Easy Editing" (March); "TI Graphics Made Easy" (March); "TI BASIC One-Liners" (May); "Using a Printer with the TI-99/4A" (June).

The following articles were originally published in *COMPUTE!* magazine, copyright 1983, COMPUTE! Publications, Inc.: "TI Mailing List" (July); "Sprite Editor for the TI" (September); "Runway 180: Using Sprites in TI Extended BASIC" (October); "All About the TI Character Set" (November); "TI Word Processor" (December).

The following articles were originally published in *COMPUTE!* magazine, copyright 1984, COMPUTE! Publications, Inc.: "The Mozart Machine" (January); "Sound Shaper" (March); "Worm of Bemer" (April); "Statistics for Nonstatisticians" (July).

The following articles were originally published in *COMPUTE!'s Gazette* magazine, copyright 1983, COMPUTE! Publications, Inc.: "Thinking" (December); "Bowling Champ" (December).

Printed in the United States of America

ISBN 0-942386-71-X

10 9 8 7 6 5 4 3 2

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-99/4 and TI-99/4A are trademarks of Texas Instruments, Inc.

Contents

Foreword

Why did you buy a computer? Was it to play games? Or were you more interested in home applications? Maybe you hoped that your children would learn BASIC programming. Whatever your reason, you'll be pleased with what you find between the covers of *COMPUTE!'s TI Collection, Volume 1.*

COMPUTE! Publications has been supporting the TI-99/4A since columnist C. Regena first appeared in *COMPUTE!* magazine in January 1983. Since then, through continuous coverage in *COMPUTE!* magazine and the publication of seven books, TI owners have recognized the high-quality programs and tutorials published by COMPUTE!. *COMPUTE!'s TI Collection, Volume 1* continues that tradition, presenting over 30 programs and articles in clear and easy-to-understand language.

This anthology of games, applications, utilities, and tutorials for the TI-99/4A contains many never before published. "SuperFont" is an exceptionally powerful and simple-to-use character editor. "Sprite Editor" and "Sound Shaper" make graphics and sound programming easy. Games like "Worm of Bemer" and "Bowling Champ" will provide hours of fun. "Thinking," a game that tests your memory and reasoning skills, can be played by the youngest learner, yet challenges even the most experienced game player. Need to organize your Christmas card files? "Mailing List" fits the bill.

And if all this weren't enough, we've included articles that show you how to use sprites in your own programs, utilities that help you organize your diskettes, an electronic spreadsheet, a word processor, and much more.

1
Getting Started

1
TI Features

C. Regena

The TI has some very powerful features. This overview of hardware, software, and miscellaneous resources will give you an idea of just what the TI can do.

Welcome to the world of the TI-99/4A computer. For home, personal, and educational applications, the TI-99/4A computer is a very powerful machine. This article will discuss some of the features unique to this microcomputer.

Extraordinary Graphics and Sound

Graphics. You may easily define your own high-resolution (detailed) graphics characters. There are 16 colors, and you may use all 16 on the screen *at the same time* in high-resolution graphics (unlike other computers). You may also use text anywhere on the screen at the same time you use high-resolution graphics. Most other microcomputers are limited when combining text with graphics.

Music. You may play up to three notes and one noise for a specified time using *one* statement. The music is specified by a number which represents a frequency of 110 Hz to 44733 Hz, tones from low A on the bass clef to beyond human hearing range. The tone may be between regular musical notes. An example which plays a three-note, C-major chord for three seconds is:

```
CALL SOUND(3000,262,6,330,4,440,2)
```

The first number is the duration in milliseconds, in this case 3000. The next numbers are frequency and loudness for each note. You may also add a "frequency" of −1 through −8 and a loudness for the noise generator. You may combine tones and noises for all kinds of sounds—everything from classical music to sound effects from outer space.

Combining music and graphics. "Computer choreography" is possible because other statements (including graphics) may be executed while music is played. You may illustrate a song, for example. Or if you have a game program, you may make calculations while you are making a noise. The computer will play music and execute statements until the duration runs out or until the program comes to another CALL

SOUND statement with a positive duration. A negative number for the duration will start that CALL SOUND statement even if the first duration has not finished. Try using a FOR–NEXT loop to vary any of the parameters for special effects. Here is a sample using just one tone:

```
100 FOR N=500 TO 880 STEP 20
110 CALL SOUND(-99,N,2)
120 NEXT N
130 FOR N=880 TO 500 STEP -20
140 CALL SOUND(-99,N,2)
```

Noises. Using negative durations and combinations of music and noise numbers for frequency, you can make all sorts of synthesized noises. Quite often with noises you will want to use a FOR–NEXT loop and vary the loudness parameter.

Built-in BASIC. The programming language of TI BASIC is built into the main console—nothing extra to buy. The TI BASIC language is an excellent language for learning how to program, yet it is powerful enough for an experienced mathematician because of the built-in functions.

String manipulations. String (non-number) manipulations are also very powerful. Here is a sample program to print a phrase A$ on the screen starting at row R and column C:

```
100 FOR I=1 TO LEN(A$)
110 CALL HCHAR(R,C+I-1,ASC(SEG$(A$,I,1)))
120 NEXT I
```

The loop will go from 1 to the LENgth of the phrase A$. String variable names must always end with a dollar sign. SEG$ takes a SEGment of the phrase. In this case we are starting at the left side and taking one letter at a time. ASC gets the ASCII character code value of the character in the phrase. CALL HCHAR uses a graphic method to place the character on the screen at a certain row and column.

No Variable Name Worries

Variable naming. In your own programming on the TI-99/4A you may use meaningful variable names, although in many microcomputers the BASIC language recognizes only two characters for a variable name. For example, if you have a program with the variable name BLUE and another variable name BLACK, other computers may recognize only one variable, BL, but the TI-99/4A knows you are using two variables. You also do *not* have to worry about embedded reserved words in variable names.

Documentation. Two excellent manuals are included with the computer. One teaches you programming in TI BASIC. The manual is very easy to understand, and a person with no previous computer experience can learn to program with this book. Also included is the *User's Reference Manual,* which may cost over $15 for other computers. The reference manual, which is in loose-leaf form, includes all the commands along with explanations and sample programs.

Plug-in modules. The easiest way to use the TI-99/4A is to insert a command module which contains a program. The modules actually add memory to the computer while they are being used. Unfortunately many of the very best modules are difficult to find or even completely unavailable.

Speech. Even though this feature is not built in, I'm going to include speech in this list of unique features of the TI-99/4A because it is very easy to use. The speech synthesizer is a small box that attaches to the side of your console.

16-bit microprocessor. The TI-99/4A uses a TMS9900, 16-bit microprocessor, which offers more computing power and greater expansion and configuration flexibility than an 8-bit microprocessor. You can get higher numeric precision and simplified memory addressing.

Programmer's aids. Programmers will enjoy the easy line editing features. Various function keys allow you to insert or delete characters or to erase or clear a line. There is also a TRACE command to help in debugging.

Another feature programmers like is the built-in automatic numbering. Just type in NUM, press ENTER, and you can start programming. The line numbers start with 100 and automatically increment by 10. Or you may specify any starting number and increment. NUM 5,2 will start with line 5 then increment by 2.

After you have programmed and added or deleted statements here and there, you'll enjoy the automatic resequencing command, RES. This command will automatically renumber your statements, including all statement numbers referenced by other statements.

Using the Cassette Recorder

Cassette. Probably one of the first items you'll need is a cassette cable to connect a cassette recorder to the computer. Nearly any cassette recorder is acceptable; however, the volume setting for the TI-99/4A is quite critical. In general, a battery-operated recorder does not work well enough for accurate data retrieval. Also, your recorder should have a tone control and a volume control. I have had the greatest success using the Panasonic RQ2309A cassette recorder.

Page I-9 in the *User's Reference Guide* tells how to connect the cassette cable, and the pages following describe how to save and load data from modules. Page II-42 shows an example of how to load a program that you have saved or purchased. Some other hints for using the cassette recorder are:

Turn the tone control to the highest setting.
Start with the volume about mid-range.
Follow the instructions after you type in OLD CS1.
If you get the message NO DATA FOUND, increase the volume.
If you get the message ERROR IN DATA, decrease the volume.

Sometimes a fraction of a change in volume can make all the difference in your success in reading a program. Once in a while, if I alternate between the two error messages at a volume setting near 2 or 3, I turn the volume to about 8 or 9 and the program will load.

The smallest jack of the cassette cable goes into the remote switch of the cassette recorder so the computer can turn the recorder on and off automatically. If the recorder does not turn on and off properly, simply remove the remote jack from the plug. You can operate the cassette recorder manually to save and load programs. For programs using the cassette recorder for data entry, you will need the remote capability. An adapter is available for the remote switch.

Disk drives. You can save and retrieve data or programs on a diskette much more quickly than by using a cassette system. The TI-99/4A uses 5¼-inch, single-sided, soft-sectored diskettes. To connect a disk drive, you also need a disk controller. One disk controller can handle up to three disk drives. Many business applications require two disk drives.

Memory expansion. The TI Memory Expansion is for 32K RAM, and you need a module that will access it. You cannot use it with console BASIC. Extended BASIC does not require the memory expansion but can use it. Pascal, TI Logo, and Editor/Assembler require the memory expansion.

Peripheral box. The "old" method had each peripheral in a separate "box" connected to the computer or the previous peripheral; each had its own power cord. The "new" system is the peripheral box, which has its own power supply and slots for cards for the RS-232 interface, memory expansion, disk controller, P-code, one disk drive, and possible future cards.

Monitor. Although most TI users connect their computers to a regular television set, it is possible to connect to a monitor. A monitor will give a very clear, sharp picture.

Making the Computer Speak

Speech. The TI Speech Synthesizer allows you to hear the computer speak to you. You will need a command module with built-in speech to hear the computer speak.

To program your own speech or to use any cassette or disk programs that use speech, you will need a module. Speech Editor and Extended BASIC have speech capabilities with a given list of words. Terminal Emulator II allows unlimited speech; the accompanying documentation gives you ideas for programming speech using this module. You may vary the pitch, slope, and inflections. You may use allophones to create words, or you may have the computer speak words which you spell phonetically.

Telecommunications and Languages

Terminal. The Terminal Emulator II command module (or Terminal Emulator I, which does not have speech) allows you to use your TI-99/4A to act as a terminal either to another computer or to a large telecommunications service. You will also need the TI RS-232 Interface and a telephone modem.

Printer. You may use a number of different brands of printers with your microcomputer. To connect your TI-99/4A to a printer, you'll need the TI RS-232 Interface and a cable to go from the interface to the printer (the cable is usually sold with the printer).

RS-232. The RS-232 Interface has two ports so you may be connected to a modem and a printer at the same time. An instruction book comes with the RS-232 so you'll know how to operate the computer under different conditions.

Extended BASIC. TI Extended BASIC (XBASIC) is a programming language contained on a module. A manual (over 200 pages) and a programmer's reference card come with the module. No other peripherals are necessary to use XBASIC. If a program has been written in XBASIC, the XBASIC module must be inserted for the program to run. Some of the advantages of XBASIC are multistatement lines, complex IF-THEN-ELSE logic, subroutine and MERGE capabilties, DISPLAY AT and PRINT USING, program security (SAVE protection), speech (with speech synthesizer), and moving sprites with greater graphics capabilities.

Editor/Assembler. For machine language programmers, it requires the memory expansion, disk controller, and one disk drive.

Software

I've mentioned software (programs) last, although it's probably the first extra purchase you will make for your computer. Software is what you need to use your computer. Software is available on command modules, cassettes, diskettes, and by typing in programs you find in books and magazines. This book is an example of a source of inexpensive software.

Write Your Own Games

C. Regena

Some tips on getting the most out of your TI when writing games.

You have probably discovered that one of the fun things to do with your TI-99/4A is to play games. In fact, many people who wanted one of the popular game machines have discovered that for about the same amount of money they could have a *computer* and still be able to play games. Many of the games written for the TI-99/4A are arcade quality—that is, they have good graphics and fast action.

To program your own games with fast, smoothly moving objects, you will want to use TI Extended BASIC. It allows you to use up to 28 *sprites*. You may define the shapes of the sprites and designate a certain magnification. You may also specify the sprites' speed. The row velocity and the column velocity may vary from −127 to +127, and by specifying numbers for both velocities you will get a diagonal movement. Sprites "wrap" at the edges of the screen, so you don't need to worry about "crashing" your program on edge conditions. With *one* CALL SPRITE statement you can define the sprite number, shape, color, position, and speed. (For more information about sprites see chapter 6.)

TI Console BASIC (the BASIC built in with no accessories or peripherals) is a language powerful enough that you can design a variety of fun games with it. If you have moving objects, however, they have to move a square at a time and thus will have jerky movement. Depending on the number of objects, BASIC games tend to be slow; however, I have seen several fast action games that really require nimble fingers.

Whether you are writing a game in TI BASIC or in TI Extended BASIC, I can offer a few programming tips. Keep in mind that the best way to learn is to actually start programming—and playing.

Randomness

Probably a central tool in computer games is the machine's ability to choose things randomly. Most computers have the command RND, but each computer has a slightly different syntax (way of writing the command). On the TI-99/4A, RND represents a random number between zero and one. Turn on your computer, press any key to begin, and press 1 for TI BASIC. Now type in PRINT RND and press ENTER. The computer will print a decimal fraction (to ten places). Usually in game situations you won't want a fraction, so multiply that fraction by a number. For example, multiply RND by 10 like this: PRINT 10*RND or PRINT RND*10. Now you will get ten times that decimal fraction.

You probably want just the whole number part of that mixed decimal number. Use the INTeger function to get the whole number. PRINT INT(10*RND). If you keep trying this command, you will get numbers from zero to nine. Remember, INT truncates the decimal portion; it does not round the number. Suppose you really wanted a random number from one through ten. The command would be: PRINT INT(10*RND)+1 or PRINT INT(10*RND+1).

One more step. Assume you want a number N to be a random number between 10 and 20, inclusive. 20−10=10. There are 10 numbers plus 1 ("inclusive"). The command could be N=INT(11*RND)+10. The portion INT(11*RND) will give you numbers from 0 to 10; then you add 10 to get numbers from 10 to 20.

Now try this short program:

```
100 FOR I=1 TO 10
110 PRINT INT(10*RND)+1
120 NEXT I
```

Run the program. Run it again. And again. The program is printing ten random numbers from 1 to 10. However, you'll notice that each time you run it, you get the same numbers in the same order. You need to add the line: 105 RANDOMIZE.

The RANDOMIZE command mixes up the numbers so that each time the program is run you will get different numbers—and that's what you want in a game. The *User's Reference Guide* indicates that the RANDOMIZE statement only needs to be somewhere in the program to generate different numbers; however, I have found that one RANDOMIZE statement at the beginning of a program does not always work. It's better to use the RANDOMIZE statement just before you use the statement containing RND. Note: If you are debugging a program, you may want to leave RANDOMIZE out so that you'll know exactly what numbers your program is choosing. Debug your program, then add the statement and test it.

Moving Objects

In general, the fewer moving objects you have in your game, the faster the action can be, and the logic will be a lot less complex. Also, each moving object should be specified by only one character number so you don't have to use up valuable time by building an object out of several characters. To move an object in TI BASIC you need to erase the object in the first position (replace it with a space) and draw it again in the second position—each move takes two statements.

Player Input

There are two main ways the computer can understand what you want: by using the joysticks or pressing keys on the keyboard. Your game may be designated for joysticks only, keyboard only, or both. Because of the logic involved, a game using both methods of input will be slightly slower in response; and depending on the branch sequence, one of the methods will be slower than the other.

Joysticks may be easier to use to learn a game, especially if the player is used to a videogame using joysticks. My own children, and many other players I know, prefer using the keyboard for *TI Invaders* and *Munchman* because the joystick response is considerably slower than the keyboard response.

The keyboard action is easy to learn because there are standard arrow keys for all games designed for the TI-99/4A. Programmers writing games for other computers often choose their own favorite keys to use, and the directions are different for each game. On the TI-99/4A, the arrow keys are E (up), X (down), S (left), and D (right), with the shooting key either the ENTER key or the period key. If there are two players, the standard arrow keys on the right half of the keyboard are I, J, K, and M.

The TI joysticks (wired remote controllers) come with a little instruction book with some sample programs. The main command is CALL JOYST(K,X,Y), which returns an X and Y value for the position of the joystick, where X and Y may be 4, −4, or 0.

To detect keys pressed on the keyboard, use the CALL KEY command. This command is like the GET command in other BASIC languages. The form is CALL KEY(0,KEY,STATUS) where 0 means to scan the whole keyboard. STATUS is a variable name (it could be ST or S, or whatever you wish) which will return whether a key has been pressed or not. KEY is a variable name (again, use whatever you wish) that will return the ASCII code of the key pressed, such as 13 for the ENTER key, 65 for the letter A, 69 for the letter E, etc.

By using IF statements, you can check which key was pressed and branch accordingly. You can also GOTO the CALL KEY statement for other keys to make the computer act as if it is ignoring all responses except the keys allowed. Here is a sample using arrow keys:

```
100 CALL KEY(0,K,S)
110 IF K=69 THEN 1000    (up arrow)
120 IF K=68 THEN 2000    (right arrow)
130 IF K=88 THEN 3000    (down arrow)
140 IF K=183 THEN 4000  (left arrow)
    ELSE 100             (any other key will
                          be ignored)
```

Remember, there are several ways to program the same procedure; this is just one way. You may prefer to use "not equal" signs or a split keyboard and an ON-GOTO statement.

A split keyboard approach scans half the keyboard using CALL KEY(1,K1,S1) or CALL KEY(2,K2,S2). The key codes returned for up, right, down, and left are 5, 3, 0, and 2. A sample program using the split keyboard is:

```
100 CALL KEY(1,K,S)
110 IF (K<0)+(K>5) THEN 100
120 ON K+1 GOTO 3000,100,4000,2000,100,1000
```

Line 110 makes sure the K value is in the right range; the key value must be from 0 to 5. All other keys are ignored. Line 120 branches according to which key was pressed. The keys corresponding to 1 and 4 were not acceptable, so they return to the CALL KEY statement. If you want to try out either of these programs, add the following lines, then run and try pressing various keys.

```
1000 PRINT "UP"
1010 GOTO 100
2000 PRINT "RIGHT"
2010 GOTO 100
3000 PRINT "DOWN"
3010 GOTO 100
4000 PRINT "LEFT"
4010 GOTO 100
```

There is a slight problem in testing for zero on the TI-99/4A console. Use logic such as IF K+1<>1 rather than IF K<>0. Also, some of the split keyboard codes are different for the TI-99/4A than for the TI-99/4. It's better not to use the comma, period, semicolon, slash, space bar, ENTER, SHIFT, B, and G so that programs may be used on either console.

Easy Editing

C. Regena

If you use these editing keys and built-in programmers' commands, you'll soon discover how fun and easy-to-use the TI-99/4A can be.

You are writing a program or keying one in from this book or *COMPUTE!* magazine when—oops!—you make an error. Hold it! Don't type the whole line over! Take advantage of the easy-to-use editing capabilities built into the TI-99/4A.

Take a look first at the arrow keys (found on letters E,S,D,X). You thought they were just for games? They will probably be the most frequently used editing keys once you get used to them. Suppose you have typed lines 100–150 and look up at the screen and notice you want to change the number in line 130:

```
130 CALL SCREEN(14)
```

Type in 130 then hold the function key (FCTN) down while you press the down arrow (↓). (It might be best to follow through this article as you sit at your TI-99/4A.) You'll notice line 130 comes up at the bottom of the screen with the cursor at the first position. Now press FCTN and the right arrow. The cursor will go toward the right. You may go one space at a time, or hold the key and it will repeat. Go over to the 4 in 14. Stop right over the 4 and type 6. Press ENTER, and the line will now be:

```
130 CALL SCREEN(16)
```

Any characters you don't want to change you can just pass over with the arrow key. Change the character you want, then press ENTER—you don't need to go to the end of the line either.

Now suppose you don't like color 16 (white) and decide you want color 6. Type 130 then FCTN ↓. Use FCTN→ to get over to the 1 in 16. Stop right on top of the 1. Now press FCTN and 1, which is DEL, for DELete. Now press ENTER and you should have:

```
130 CALL SCREEN(6)
```

Try another function key. Type 130 then FCTN↓. Use FCTN→ to go on top of the 6 and type 2. Just a second, though. You don't want screen 2; you want 12. Use FCTN← to back up one spot (cursor on 2). Press FCTN 2 for INSert. You won't notice anything right away, but now type 1—you have color 12. Press ENTER and your line has been changed.

Automatic Repeats

The left arrow, right arrow, and DELete keys repeat automatically when you hold the key down. The INSert key needs to be pressed just once and characters will keep being inserted as you type until you press ENTER, DELete, or one of the arrow keys. To delete or get rid of a whole line, type the line number and then press ENTER.

Two more handy editing keys are the up arrow and down arrow. Let's assume you have the following lines:

```
200 CALL HCHAR(3,5,42)
210 CALL HCHAR(3,8,42)
220 CALL HCHAR(3,20,33)
```

You run your program and discover the graphics need to be a line lower—the row value needs to be changed from 3 to 4.

Type 200, press FCTN↓, and use the right arrow to change the 3. Instead of pressing the ENTER key, press FCTN↓. After line 200 has been edited, the very next line, line 210 in this case, will appear for editing. Likewise, the up arrow will give you the line just before the one on which you were working.

Two other editing keys you should be aware of are ERASE (FCTN 3) and CLEAR (FCTN 4). You may already be familiar with CLEAR. If you are running a program and want to stop, FCTN 4 will interrupt the program. (QUIT, FCTN =, will stop the program, erase it from memory, and return to the TI title screen; CLEAR stops the program but retains it in memory and you may either CONtinue or RUN.)

CLEAR has another function while you are programming. If you start typing a line and decide you don't want that line after all, press CLEAR. The cursor will go to the next line and the line you were working on is ignored. ERASE will erase the line that you are working on.

The other function keys you see along the top row of your keyboard are used in some of the command modules and are described in the manuals accompanying the modules.

Some helpful commands for programmers are LIST, NUM, and RES. As you are writing a program, each command needs a line number. When the program is run, the computer executes each line in numerical order. The command LIST will list your complete program in order. As your program lists, the lines scroll off the top if the program is too long for one screen. If you want to stop the listing, press CLEAR. If you want to list only part of your program, just list the lines you wish:

Command	**Lists:**
LIST	Whole program
LIST–200	All lines up to and including line 200
LIST 100–300	Lines 100 to 300 inclusive
LIST 300–	Lines 300 to the end

When you're typing in a program, it will save time and reduce the chance for error if you let the computer type the line numbers. Type in the command NUM (for NUMBER). The computer will automatically start with line 100. Now type in CALL CLEAR and press ENTER. The computer enters line 100 and starts you on line 110. The NUM command automatically increments the line numbers by 10.

You may start anywhere—for example, type NUM 3220 and press ENTER. Your program starts with line 3220 and increments by 10.

Yes, you can change the increments also. Type NUM 200,5 and you'll start with line 200 and increment by 5 (line 200,205,210, etc.). The general form is: NUM initial line, increment.

If you want the program to start with line 100 but the increments to be 7 instead of 10, you may use NUM ,7.

To get out of the automatic numbering, just press ENTER after the line number or CLEAR. You'll also notice that if you have a program in the computer and type NUM the computer will show you what is on that line. If you want to keep the line as is, just press ENTER.

Complete Renumber

RES is a command that stands for RESEQUENCE. You've been programming and adding lines here and there and want it to look nice again, all numbered by tens. Type RES and press ENTER. As soon as the cursor reappears, your program is resequenced or renumbered, including all line numbers referenced in other lines. Try this sample:

```
12 CALL SCREEN(14)
20 FOR I=1 TO 8
30 CALL SOUND(500,-I,2)
35 NEXT I
```

Now type RES and press ENTER, then LIST. The lines are resequenced, starting with 100 and incrementing by 10. Like the NUM command, you may specify the starting line number and the increment: RES initial line, increment.

Try RES 10 then LIST.

Try RES 1,1 or RES ,5 and experiment with your own numbers.

Quite often I like to start writing programs with line numbers incrementing by 10. Type in NUM and start programming. If the program has several branches, I may start one branch at 1000 (NUM 1000), another at 2000, etc. Leaving gaps in the line numbers makes it easier to add lines later.

For example, if I have a line 200 and the next line is line 210, I may easily add lines in between by numbering them 202, 204, etc. But what if I had to add 15 lines between lines that are only ten apart? RES ,50 will spread the lines apart and allow more numbers in between. Of course, when I'm through with the program, I RES so the program starts at 100 and increments by 10, and you can't tell where I planned poorly and had to add lines.

All About the Character Set

Michael A. Covington

This brief outline of the TI character set explains how the computer recognizes each character. The author discusses some uses of the characters' numeric codes and indicates which characters' graphic representations can be assigned or changed.

Chances are you've never given your computer's character set much thought. You press keys on the keyboard and the characters appear on the screen; that's all there is to it, or so it seems. But there's a lot more going on than meets the eye.

Inside the computer, each character is represented by a *numeric code*—a number between 0 and 255 inclusive. For instance, the code for capital E is 69; the code for an exclamation mark is 33; the code for a blank (a blank is a character just like all the others) is 32. To associate these codes with the characters you see on the screen, the computer has to know two more things about each of them: a *graphic representation* that describes how the character is supposed to look on the screen, and a *key assignment* that indicates what key or combination of keys you can hit on the keyboard to type the character. For instance, the character string "HELLO THERE!" (not counting the quotation marks) is represented as shown in Table 1.

Table 1. Representation of the String "HELLO THERE!"

Graphic representation:	**H**	**E**	**L**	**L**	**O**	
Numeric code:	**72**	**69**	**76**	**76**	**79**	**32**
Key assignment:	**H key**	**E key**	**L key**	**L key**	**O key**	**space bar**

Graphic representation:	T	H	E	R	E	!
Numeric code:	84	72	69	82	69	33
Key assignment:	T key	H key	E key	R key	E key	shift & 1 keys

Statements Using Numeric Codes

Normally (when you type characters in response to a string INPUT statement or when you type them as part of a program) you enter characters by hitting the keys that correspond to them. That is, you access them by means of their key assignments, and within the program you treat them as character-string data. But there are ways of referring to characters by their numeric codes and treating them as numbers. For instance, the CALL HCHAR and CALL VCHAR statements, which you meet at an early stage as you work through the manuals that come with the computer, refer to characters by their numbers. The statement:

```
CALL HCHAR(3,3,69,20)
```

will place a row of 20 capital E's (character number 69) on the screen beginning at row 3, column 3.

Also, you can input characters as numeric codes. The CALL KEY statement senses whether a particular key on the keyboard is up or down; when a key is pressed, CALL KEY gives you the numeric code corresponding to it. For instance, here is a program which will tell you the numeric code of any key on the keyboard:

```
10 PRINT "PRESS ANY KEY..."
20 CALL KEY(5,CODE,STATUS)
30 IF STATUS <> 1 THEN 20
40 PRINT CODE
50 GO TO 10
```

The heart of the program is lines 20 and 30. Line 20 tells the CALL KEY subroutine to look at the keyboard and report what's going on. The variable STATUS will equal 1 only if the condition of the keyboard has changed since the last time the routine looked at it. If STATUS does not equal 1, we simply go back to line 20, since we don't want to do anything more if the user hasn't pressed a key or hasn't yet let go of the one already looked at. The variable CODE contains the numeric code associated with the key being pressed, if any. (The first parameter of CALL KEY, the number 5, simply indicates that we want the usual BASIC set of codes; specifying other numbers there instructs the computer to use other sets of key assignments for various special purposes.)

The ASC and CHR$ functions allow you to convert back and forth between numeric codes and character strings. If A$ is a character string, ASC(A$) is the numeric code of its first character; thus ASC("E") is 69. Conversely, if N is a number, CHR$(N) is a one-character string of which N is the numeric code; thus CHR$(69) is E. If we want the program above to print the characters themselves rather than their codes, we can convert the codes into characters by changing line 40 to:

```
40 PRINT CHR$(CODE)
```

The CALL CHAR subroutine allows you to alter graphic representations using a hexadecimal code that the manual describes in detail. For instance, if you want to change the dollar sign ($) into a British pound sign (£), just execute this statement:

```
CALL CHAR(36,"001C22207C20207E")
```

That will do it, at least as long as the program is running: The key assignment and numeric code will be the same, but the dollar sign will look like a pound sign. (It will revert to its original appearance when your program stops executing.)

What's Not in the Manual

Those are the preliminaries; now we get to the really interesting part (the part that isn't in the manual, at least not entirely). Internally, the computer can use any number from 0 to 255 as a character code; any such code can be an element in a character string and can be referred to by CALL VCHAR, CALL HCHAR, and CHR$. (In fact, CALL VCHAR, CALL HCHAR, and CHR$ will actually take numbers up to 32767; multiples of 256 are subtracted as necessary to get a number in the 0 to 255 range.) But not all the codes have key assignments or graphic representations. The breakdown (by numeric codes is as follows:

0—Undefined (no key assignment, no graphic representation).

1–15—Function keys (Table 2). Most of these characters can be input by means of the CALL KEY statement, but they cannot be typed in normal contexts (for example, in response to an INPUT) because there they are interpreted as requests to perform cursor movements or the like. They have no graphic representations (if you print them, you get blanks or garbled patches).

16–29—Undefined (like 0, these codes have no key assignments and no graphic representations, and there is no straightforward way of giving them either).

30—The graphic representation of this character is the black square that marks the cursor; thus, CHR$(30) is handy if you want a black square. No key is assigned to it.

31—This is the screen border character—a blank that is the color of the border rather than the typing area. No key is assigned to it.

32–126—Standard ASCII character (Table 3). These are the characters you use every day, including the alphabet, the numbers, and all the punctuation marks and mathematical symbols. Their graphic representations can be changed with CALL CHAR but will revert to their original form when the program ends.

127–159—User-defined characters (Table 4). These start out with no graphic representations, but you can define them with CALL CHAR, and, contrary to what the TI manual says, such definitions remain in effect after the program stops running (though most are disrupted when another program is loaded).

What most people don't realize is that these characters can be typed—they have key assignments and are acceptable in the same context as any other character (that is, in response to an INPUT or CALL KEY, or within quotes in a program). All but one of them require you to hold down the CTRL key (at the lower-left corner of the keyboard) when typing them; character number 127 uses the FCTN key instead.

160–175—Undefined

176–198—These characters have key assignments (Table 5), but no graphic representations and no direct way of giving them any. They can be used as special function keys of some sort (in response to either CALL KEY or INPUT), but not as displayable characters.

199–255—Undefined.

Even the undefined character codes (those that cannot be typed on the keyboard or displayed on the screen) are not completely useless. You can refer to them by means of CHR$ and ASC and use them as special markers of various kinds when manipulating character strings. They also may come into play when you are transmitting data to other devices (for example, printers or other computers) that have definitions for characters that are undefined on the TI-99.

Finally, consider this possibility. Each character in a character string has a code between 0 and 255 inclusive, accessible through CHR$ and ASC. Also, the SEG$ function allows you to address individual characters in a string, and the & (concatenation) operator allows you to construct strings out of individual characters. This means that a character string gives you a compact way of storing a set of integers between 0 and 255—each element occupies only one byte in memory, as compared to the eight bytes normally needed to store a number. So if you have a program that needs to keep track of thousands of small integers—more than will fit in available memory in numeric form—then character strings may be the answer.

Table 2. Function Key Codes

(None of these characters have graphic representations, nor can they be given them. They can be typed only through the CALL KEY statement, not in response to a string INPUT statement, or within a program.)

Code	Key
1	FCTN 7("AID")
2	None usable. The key definition associated with this code is FCTN4, but in BASIC, hitting that key interrupts the program.
3	FCTN 1("DELETE")
4	FCTN 2("INSERT")
5	None usable. The key definition associated with this code is FCTN =, but hitting that key forces a machine reset and the program in memory is lost.
6	FCTN 8("REDO")
7	FCTN 3("ERASE")
8	FCTN S(left arrow)
9	FCTN D(right arrow)

10	FCTN X(down arrow)
11	FCTN E(up arrow)
12	FCTN 6("PROC'D")
13	ENTER
14	FCTN 5("BEGIN")
15	FCTN 9("BACK")

Table 3. ASCII Graphic Characters on the TI-99/4A

(This table gives the numeric codes and graphic representations; the key assignments are marked on the keyboard. The graphic representations can be changed by the CALL CHAR statements but revert to their original form when the program stops running.)

Code	Graphic Representation	Code	Graphic Representation
32	(space)	53	5
33	!	54	6
34	"	55	7
35	#	56	8
36	$	57	9
37	%	58	:
38	&	59	;
39	'	60	<
40	(	61	=
41	)	62	>
42	*	63	?
43	+	64	@
44	,	65	A
45	−(minus)	66	B
46	.	67	C
47	/	68	D
48	0	69	E
49	1	70	F
50	2	71	G
51	3	72	H
52	4	73	I

74	J	97	a
75	K	98	b
76	L	99	c
77	M	100	d
78	N	101	e
79	O	102	f
80	P	103	g
81	Q	104	h
82	R	105	i
83	S	106	j
84	T	107	k
85	U	108	l
86	V	109	m
87	W	110	n
88	X	111	o
89	Y	112	p
90	Z	113	q
91	[	114	r
92	\ (back slash)	115	s
93	]	116	t
94	^	117	u
95	_ (underline)	118	v
96	'	119	w
120	x		
121	y		
122	z		
123	{		
124	¦		
125	}		
126	-		

Table 4. User-Definable Graphics Characters

These characters can be typed using the key combinations listed and are acceptable in any context (that is, they can be input using the CALL KEY or INPUT statements and can appear between quotes within a BASIC program).

Graphic representations can be given to these characters with the CALL CHAR statement. Contrary to TI documentation, such representations, once assigned, will persist after the program stops running.

Code	Key	Code	Key
127	FCTN V	144	CTRL P
128	CTRL ,(comma)	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 .(period)
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		

Table 5. Characters with Key Assignments But No Graphic Representations

These characters are not mentioned in TI documentation. They can be typed in any context (that is, in response to an INPUT or CALL KEY statement or between quotes in a program), but they have no graphic representations and cannot be given any.

Code	Key	Code	Key
176	CTRL 0	188	FCTN 0 (zero)
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 , (comma)	196	FCTN N
185	FCTN . (period)	197	FCTN Q
186	FCTN /	198	FCTN Y
187	CTRL /		

2
The Basics

2
TI BASIC One-Liners

Michael A. Covington

The BASIC DEF statement can become a powerful tool in your programmer's bag of tricks. Here's how to use it.

If you've been programming in BASIC for any time at all, you've surely come across, and used, some of the built-in functions that the language provides, such as INT, SIN, COS, TAN, ATN, and LOG. But did you know that you can use the DEF statement to create functions of your own? Defining your own functions lets you type a complicated formula only once, and it allows you to build complex functions out of simple ones in a most efficient way.

Suppose, for instance, that your LOG function gives you natural (base *e*) logarithms, and you want base 10 logarithms. (If you're not sure which you've got, type PRINT LOG(10)—if the answer is 1, you're in base 10, and if it's about 2.3026, you're in base *e*.) You can convert base *e* logarithms to base 10 by dividing them by 2.302585093, so one of the options open to you is obviously to write LOG(X)/2.302585093 (or whatever) every time you need a base 10 log. But there's an easier way.

Creating Functions

To create your own function—let's call it LOG10, though some computers may insist that you name it something like FNL—just include, early in your program, a statement like this:

```
10 DEF LOG10(X)=LOG(X)/2.302585093
```

From then on, you'll be able to use the new function LOG10 to get base 10 logarithms. Try it out with a program like this:

```
10 DEF LOG10(X)=LOG(X)/2.302585093
20 FOR I=1 TO 10 STEP 0.1
30 PRINT I,LOG10(I)
40 NEXT I
```

and compare the results against a table of logarithms.

The DEF statement is different from most BASIC statements in that it can't refer to variables. (The X in it—it could be any variable name—is used only as a placeholder for the number within the parentheses; it is completely separate from any variable named X that you may use elsewhere in the program.) You can refer only to numbers or other functions. Some computers require that the name of the function be three letters and that the first two be FN—FNA, FNB, FNL, and so forth—although the TI-99, and many other microcomputers, allow you to name functions with the same type of names you use for variables.

Sample One-Liners

So that's how it's done. Now let's look at some practical examples.

1. *Base 10 logarithms.* That's what we've just discussed. For reference, here is the statement:

```
DEF LOG10(X)=LOG(X)/2.302585093
```

(assuming your machine's LOG function gives you base *e* logs).

2. *Base 2 logarithms.* For a machine on which the LOG function gives base *e* logarithms, you can get base 2 logarithms by using:

```
DEF LOG2(X)=LOG(X)/0.6931471806
```

If your machine's LOG function gives base 10 logarithms, you'll need to use DEF LOG2(X)=LOG(X)/0.3010299957 instead.

3. *Degrees to radians.* If X is the measure of an angle in degrees, then RAD(X) will be the same angle measured in radians, if you define the following function:

```
DEF RAD(X)=X/57.29577951
```

4. *Radians to degrees.* The opposite function, converting X in radians to DEG(X) in degrees, is:

```
DEF DEG(X)=X*57.29577951
```

5. *Arcsine* (in radians). The following definition will give you the arcsine function (which is not usually provided in implementations of BASIC, although the arctangent is).

```
DEF ASN(X)=2*ATN(X/(1+SQR(1-X^2)))
```

If you look through a table of trigonometric identities, you may find an apparently equivalent, but simpler, formula that would lead to the statement DEF ASN(X)=ATN(X/SQR (1−X^2)). But note that this version won't do ASN(1) correctly (it will try to divide by zero). Hence the first version is preferable.

6. *Arccosine* (in radians). If you have the arcsine function, you can get the arccosine, as follows:

```
DEF ACS(X)=1.570796327-ASN(X)
```

Remember that the DEF statement for ASN must precede the DEF statement for ACS (you can't refer to a function until you've defined it).

7. *Rounding to a particular number of decimal places.* Where n stands for the number of decimal places you want, use the definition:

```
DEF ROU(X)=INT(((10^N)*X)+0.5)/(10^N)
```

Note that you *must* substitute a number for n; in most implementations, n cannot be a variable. Hence, if you want to round to three decimal places, your statement will read DEF ROU(X)=INT(((10^3)*X)+0.5)/(10^3). The number of decimal places can be negative, of course; if you want to round to the nearest 20, ask for −1 decimal place, and if you want to round to the nearest 1000, ask for −3 decimal places.

8. *Rounding to a particular number of significant digits.* Often, you'll find that the most convenient type of rounding involves coming up with a particular number of significant digits rather than a particular number of decimal places. You can accomplish this with the definition:

```
DEF RSF1(X)=(N-1)-INT(LOG10(X))
DEF RSF(X)=INT(((10^RSF1(X))*X)+0.5)/(10^RSF
1(X))
```

Here the definition is so complex that it is best done in two stages: first we define RSF1, which is a function used internally in RSF, and then we define RSF, which is the function we actually use. The character n stands for the number of significant digits you want; as before, you must substitute a number for it when typing the definition into the computer.

A word of warning: RSF (with its subsidiary calls to RSF1, which in turn calls LOG10) can take quite a bit of time to execute (about half a second of realtime on the TI-99).

9. *Sexagesimal output: minutes.* Our practice of expressing time in hours, minutes, and seconds, and angles in degrees, minutes, and seconds, is a remnant of an ancient Babylonian base-60 (sexagesimal) number system. Often, in a computer program dealing with time or with angles, it's necessary to express the output in terms of units, minutes, and seconds. The units are derived by taking INT(X); thus the units part of 2.5 hours = INT(2.5) = 2 hours. Here is a function that gives the minutes part:

```
DEF MNT(X)=INT(60*(X-INT(X)))
```

the INT of that.

10. *Sexagesimal output: seconds.* The seconds part of the value, in turn, is given by:

```
DEF SCD(X)=60*(60*(X-INT(X))-MNT(X))
```

That is, we subtract the integer part *and* the minutes; what's left gets multiplied by 60 twice.

The sexagesimal output functions can be tested by means of a program such as the following:

```
10 DEF MNT(X)=INT(60*(X-INT(X)))
20 DEF SCD(X)=60*(60*(X-INT(X))-MNT(X))
30 FOR H=0 TO 2 STEP 0.01
40 PRINT
50 PRINT H,"HOURS"
60 PRINT INT(H),MNT(H),SCD(H)
70 NEXT H
```

From this we learn, for example, that 0.01 of an hour is 36 seconds, and that 0.5 of an hour is 30 minutes. (If your computer uses binary, rather than BCD or Radix-100, internal representations of numbers, you may get odd errors due to rounding or lack of it. The solution would be to round the number of hours to some reasonably small number of decimal places before invoking the conversions, and perhaps to insert some rounding in the definitions of MNT and SCD themselves.)

Incidentally, for sexagesimal *input*, you don't need any special functions, only a bit of multiplication. For instance, the statements:

```
10 PRINT "TYPE HOURS, MINUTES, SECONDS"
20 INPUT H,M,S
30 H=H+M/60+S/3600
```

will give you (as H) the number of hours expressed as a decimal.

11. *Modulo 12 arithmetic.* In dealing with hours, you'll often want to reduce numbers to modulo 12. For instance, if it's 11 a.m., then you can calculate the time four hours later by adding 11 + 4 (which gives you 15) and then taking the resulting modulo 12. The function definition is:

```
DEF MOD12(X)=12*(X/12-INT(X/12))
```

(unless, of course, your computer has a built-in MOD function, which is even simpler to use). This particular function is likely to be bothered by rounding and truncation errors. On the TI-99, I get accurate results for numbers under 1000 or so, but larger numbers give slightly erroneous answers.

12. *Modulo 60 arithmetic.* The same function, giving modulo 60 answers (for dealing with minutes and seconds), is:

```
DEF MOD60(X)=60*(X/60-INT(X/60))
```

(as if you couldn't have guessed). The following program starts with a time expressed as H hours M minutes, and adds M1 minutes:

```
10 DEF MOD12(X)=12*(X/12-INT(X/12))
20 DEF MOD60(X)=60*(X/60-INT(X/60))
30 INPUT H,M
40 INPUT M1
50 M=MOD60(M+M1)
60 H=H+INT(M1/60)
70 PRINT H,M
```

Line 50 adds the right number to the minutes part, and line 60 adds to the hours part if necessary.

CALL KEY Hints

Roger Lathrop

CALL KEY is often used in programs, but there are a number of ways to use CALL KEY which are rarely seen and easy to use.

If you use a TI-99/4A and do your own programming you already know how to use the CALL KEY routine. In fact you probably use it in just about all your programs. However there is something they don't tell you in the user's manual that you may find very useful. First let's look at a typical use of CALL KEY, then let's see how it can be improved:

```
10 PRINT " KEY R TO REPEAT KEY E TO END"
20 CALL KEY(0,A,B)
30 IF A=69 THEN 50
40 IF A=82 THEN 10 ELSE 20
50 END
```

This kind of program is often used to get information from the user: "Do you want to play another game Y or N?" It works fine as long as uppercase letters are entered. *You* know why it won't work with lowercase letters so you can quickly correct your mistake and go on. It's just a minor nuisance. Now change line 20 to read:

```
CALL KEY(3,A,B)
```

With this simple change you have eliminated the problem altogether. Using a three as the first argument returns uppercase characters only, so there is no chance of error. You may remember reading this in the users manual, maybe you even use it sometimes. But now let's go a step further, into something they don't tell you in the manual. Try this simple program:

```
10 INPUT A$
20 PRINT A$
30 CALL KEY(3,A,B)
40 INPUT B$
50 PRINT B$
60 GOTO 10
```

Run this program using lowercase letters (it doesn't matter what you enter). You will see that your first input will be lowercase, just as you typed it, but all the following entries will be returned as uppercase characters. Line 30 puts the computer in key unit three, and it will stay in that mode even when it performs an INPUT statement. Now add this line, and run it again:

```
55 CALL KEY(5,A,B)
```

You are now switching back and forth between key units three and five. Key unit five is the mode the computer is in normally. You can use this to ask a question, such as YES or NO, and have it come back as uppercase to simplify verification, no matter how it is entered. You may then switch back to key unit five to enter information such as names, where both upper- and lowercase letters may be desired. Note that control keys are inactive in key unit three, and that numeric and punctuation keys work normally with the SHIFT key.

If you need to switch back and forth often in a program you may wish to make the CALL KEY statements a separate subroutine. You can use dummy CALL KEYs, as we did in lines 30 and 55, or you can use an active CALL KEY using the key unit you wish. Any following INPUT statements will react accordingly. Once you have the keyboard mapped the way you wish, any following CALL KEYs may use a key unit of zero. Key unit zero will not change the keyboard mapping.

Take the time to learn this simple programming trick. It's easy to learn, will help make your programs easier to run, and in many cases can make them simpler to write and debug.

All Sorts of BASIC Sorts

C. Regena

One of the functions of a computer is to organize data. You may want to alphabetize lists, arrange events by date, or list a class in order by test scores. There are a variety of sort routines or algorithms to arrange data.

Computer programmers and analysts often enjoy looking at sort routines and comparing speed and efficiency. Usually the amount of time it takes a computer to sort depends on how many items are in the list and how out-of-order the items are. Different computers vary in speed also. (Although the TI-99/4A computer is slower than other microcomputers in PRINTing or LISTing, it's just as fast or faster in calculations and comparisons.)

Here are four different sort routines written in BASIC for you to try, and to implement in your own programs. They will work on a TI with regular or Extended BASIC.

In the listings, line 100 tells the type of sort being used. Lines 110–170 randomly choose 50 integers from 1 to 100. Ordinarily, you would INPUT, READ, or calculate the numbers used. The actual sorting starts at line 200. Lines 500 to the end print the final sorted list of numbers in the example.

Bubble Sort

The Bubble Sort (or simple interchange sort) is probably the most common and easy to understand sort. It's fine for small numbers of items or for a list of items that is not much out of order. The program compares each number to the next number and exchanges numbers where necessary.

If one switch has been made during a pass through all the numbers, the loop of comparisons starts over. In this example, if the 50 numbers happened to be in exact opposite order, the maximum number of passes would be necessary, and the process would take longer than if only a few numbers were out of place. For larger numbers of items, this sort can seem to take forever.

Shell Sort

The Shell Sort is considerably faster than the Bubble Sort. In general, for a random order of 50 numbers, the shell sort is about two or three times as fast as the Bubble Sort. The Shell Sort speeds up execution because the number of comparisons that need to be made is reduced.

In an array of N numbers, it first determines B so that $2^B<N<2^{B+1}$ and then the variable B is initialized to 2^{B+1}. The loop varies the counter I from 1 to N − B. First, it checks if A(I)<A(I+B). If so, it increments I and continues with the comparisons. If not, it exchanges A(I) and A(I+B) and changes the subscript.

When I reaches the value of N, it reduces B by a factor of two and starts the loop again. When B = 0 the sort is complete. I've used a couple of extra variables in the example for clarity.

Sort C

The third kind of sort routine offered here is also faster than the Bubble Sort if the numbers are quite mixed up. The program goes through all the numbers and places the minimum value in the first spot of the array. The loop keeps finding the minimum of the numbers remaining and replaces it in order.

Sort D

This sort is similar to the previous one, except that with each pass through the numbers, both the minimum and the maximum numbers are found and placed at the appropriate end spots.

The way these sorts are listed, the given numbers will be arranged in ascending order. To change to descending order, simply exchange the less than or greater than signs in the sort comparisons.

If you are alphabetizing, the variable terms will be string variables, such as A$(I).

You may have several items which need to be associated as they are sorted. For example, suppose you have names and scores to be arranged by score. The names and scores are first arranged as N$(1), S(1); N$(2), S(2); etc. In the interchange you would need to sort the S values, and then switch both terms, like this:

```
SS=S(I)
NN$=N$(I)
S(I)=S(I+1)
N$(I)=N$(I+1)
S(I+1)=SS
N$(I+1)=NN$
```

Program 1. BASIC Bubble Sort

```
100 REM TI BASIC BUBBLE SORT
110 DIM A(50)
120 FOR I=1 TO 50
130 RANDOMIZE
140 A(I)=INT(RND*100+1)
150 PRINT A(I);
160 NEXT I
170 PRINT : :
200 LIM=49
210 SW=0
220 FOR I=1 TO LIM
230 IF A(I)<=A(I+1)THEN 290
240 AA=A(I)
250 A(I)=A(I+1)
260 A(I+1)=AA
270 SW=1
280 LIM=I
290 NEXT I
300 IF SW=1 THEN 210
500 FOR I=1 TO 50
510 PRINT A(I);
520 NEXT I
530 END
```

Program 2. BASIC Shell Sort

```
100 REM  TI BASIC SHELL SORT
110 DIM A(50)
120 FOR I=1 TO 50
130 RANDOMIZE
140 A(I)=INT(RND*100+1)
150 PRINT A(I);
160 NEXT I
170 PRINT : :
200 B=1
210 B=2*B
220 IF B<=50 THEN 210
230 B=INT(B/2)
240 IF B=0 THEN 500
250 FOR I=1 TO 50-B
```

```
260 C=I
270 D=C+B
280 IF A(C)<=A(D)THEN 340
290 AA=A(C)
300 A(C)=A(D)
310 A(D)=AA
320 C=C-B
330 IF C>0 THEN 270
340 NEXT I
350 GOTO 230
500 FOR I=1 TO 50
510 PRINT A(I);
520 NEXT I
530 END
```

Program 3. BASIC Sort C

```
100 REM{3 SPACES}TI BASIC SORT C
110 DIM A(50)
120 N=50
130 FOR I=1 TO N
135 RANDOMIZE
140 A(I)=INT(RND*100+1)
150 PRINT A(I);
160 NEXT I
170 PRINT : :
200 M=A(1)
210 IM=1
220 FOR I=2 TO N
230 IF A(I)<M THEN 260
240 M=A(I)
250 IM=I
260 NEXT I
270 AA=A(N)
280 A(N)=A(IM)
290 A(IM)=AA
300 N=N-1
310 IF N>1 THEN 200
500 FOR I=1 TO 50
510 PRINT A(I);
520 NEXT I
530 END
```

Program 4. BASIC Sort D

```
100 REM{4 SPACES}TI BASIC SORT D
110 DIM A(50)
120 N=50
130 FOR I=1 TO N
```

```
135 RANDOMIZE
140 A(I)=INT(RND*100+1)
150 PRINT A(I);
160 NEXT I
170 PRINT : :
200 S=1
210 MN=A(S)
220 IMIN=S
230 MX=MN
240 IMAX=S
250 FOR I=S TO N
260 IF A(I)<=MX THEN 290
270 MX=A(I)
280 IMAX=I
290 IF A(I)>MN THEN 320
300 MN=A(I)
310 IMIN=I
320 NEXT I
330 IF IMIN<>N THEN 350
340 IMIN=IMAX
350 AA=A(N)
360 A(N)=A(IMAX)
370 A(IMAX)=AA
380 N=N-1
390 AA=A(S)
400 A(S)=A(IMIN)
410 A(IMIN)=AA
420 S=S+1
430 IF N>S THEN 210
500 FOR I=1 TO 50
510 PRINT A(I);
520 NEXT I
530 END
```

Searching Algorithms

Doug Hapeman

Searching through data using BASIC can be very slow. Some searching algorithms can be much faster than others.

The word *algorithm* is derived from Al Khuwarizmi, a ninth-century Arabic mathematician. He was interested in solving certain problems in arithmetic, and devised a number of methods for doing so. These methods were presented as a list of specified instructions, and eventually his name became attached to such methods.

An algorithm is simply a formula to use for getting done what you want to accomplish. It's *a sequence of operations that, when applied to given information, will produce a desired result.* Algorithms are used unknowingly everyday. For instance, the instruction sheet for assembling your child's new bicycle, directions for opening a combination lock, kitchen recipes for cooking, the rules for playing a game, and road maps are all examples of algorithms. An algorithm, then, is a precisely described set of directions to follow in order to accomplish a stated task. The algorithms we have in mind are the set of procedures that can be used in searching through data lists.

In many program applications you will be storing a wide variety of information, from inventory management, membership and address files, genealogical records, meteorological data—the list is endless! Most lists are stored in a data structure called a one-dimensional array, or subscripted variable.

Storing Information

An array is a block of storage locations in computer memory which is reserved for a collection of variables. Each variable in the list is called an element of the array. TI BASIC permits you to use one-, two-, or three-dimensional arrays, in addition to simple variables.

If you assign a numeric value or string expression to a simple variable, then a specific memory location with an address that is unique is set aside. For example, LET A=12; the value of 12 is placed in a memory location and its address is the variable A. If you ask the computer to PRINT A, it will print 12, the value assigned to it. If you assign a second value to A, LET A = 100, the first value is then forgotten. A simple variable can hold only one value or expression at a time.

An array brings new dimensions to the variable. In an array the variable is subscripted, A(I)=12, and you may assign many values or expressions to it. TI BASIC permits 11 elements without any special dimensioning. If the number of elements exceeds 11, then extra room must be allocated with the use of the DIMension statement. The array then sets aside a big enough block of memory locations for the number of elements you set in the DIM statement.

What is in the space set aside for these elements? Try these two short programs:

```
100 FOR I=0 TO 10
110 PRINT "A(";I;")=";A(I)
120 NEXT I

100 FOR I=0 TO 10
110 PRINT "A$(";I;")=";A$(I)
120 NEXT I
```

Notice that each element in the string array is a null string and each element in a numeric array is a zero until you replace them with values during the program. When the array is accessed, each element within the block must be given an address that is unique. For example, A(1)=12; A(2)=100. The A is the name of the array, and the specific address is the subscripted number given to the array A. As an illustration of a one-dimensional string array, let's set up an array called NAME$ that will hold the names of people. To INPUT a number of names and fill in the array, you can key in the following code:

```
100 CALL CLEAR
110 I=0
120 INPUT "ENTER THE NAMES:":NAME$(I)
130 I=I+1
140 GOTO 120
```

This program will fill array NAME$ until 11 names are entered and then end with an ERROR MESSAGE**BAD SUBSCRIPT, because we did not DIMension a larger array.

The one-dimensional array is often called a *list,* and has only one integer value following its name A(6). The two-dimensional array is referred to as a *table,* or *matrix,* because it can represent any two-dimensional condition, such as charts, graphs, or any tabular display that uses rows and columns. It is described with two integer values which define the number of rows and columns A(12,3). The three-dimensional array has three integer values defining its characteristics A(5,2,11).

Comparing Using ASCII Codes

A very common problem in working with lists stored in one-dimensional arrays is the need to search the array to access a particular item or to determine whether it is in the array. Some of the slowest procedures in BASIC (and other computer languages) are searching and sorting, because the process involves time-consuming comparisons, whether string or numeric.

In order to understand how strings are processed, some background about ASCII character codes is necessary. ASCII stands for American Standard Code for Information Interchange, and it is an established standard for computers. There are 128 different codes defined in the ASCII standard to represent alphabetic, numeric, special characters, and control codes (see "All About the Character Set" elsewhere in this book).

The way the ASCII codes are ordered—the space (32), punctuation, numbers and other special characters (33–64), uppercase alphabet (65–90), more special characters (91–96), and then the lowercase alphabet (97–122)—makes it possible to compare strings by using the same relational operators that are used to compare numbers. The computer compares two strings by comparing one character at a time, moving in a left-to-right direction until a difference is found. Here are some examples:

Is JORDEN greater than JORDAN?
J O R D E N > J O R D A N ?
74 79 82 68 **69** 78 74 79 82 68 **65** 78
69 is greater than 65, therefore JORDEN is greater than JORDAN.

Is GREENE equal to GREEN?
G R E E N E = G R E E N ?
71 82 69 69 78 **69** 71 82 69 69 78 **32**
69 is greater than 32, so GREENE is not equal to GREEN.
(Note that 32 is ASCII for a space.

In data processing most of the time you will be working with alphabetically ordered lists of information. There are times though when you will have to work with unordered lists. The program listing at the end of the article will demonstrate how much faster a list can be searched when the information is ordered (alphabetized).

The Linear Search

When processing unordered data the most common algorithm is the linear search. The linear search takes the item you are searching for and compares it with each succeeding item in the list until it finds a match (this process can be very time consuming). If the item is not in the list, the search cannot detect it without passing through the entire array. Only then can it verify that the item is not present. Line 340, IF C$=B$(I) THEN 660, where C$ is the item you are searching, is compared to each element in the array until it finds the item being searched.

The time required to search unordered data varies depending on the length of the list and where exactly in the list the item being searched for is located.

If you want to reduce the searching time, the first step would be to order the list. How do you get an ordered list? You could INPUT all the information alphabetically when using the given application program, but that would not be feasible or practical. Much easier would be to include a sorting routine in the program. A sorting routine will alphabetize or arrange numerics in ascending or descending order (see "All Sorts of BASIC Sorts" elsewhere in this book).

The Alphabetical/Linear Search

How can an ordered list be searched efficiently? Once the list is alphabetized it immediately becomes easier to process, particularly when searching for items that are not in the list. In the case of the unordered list, the entire list had to be searched to determine that an item was not there. For a list of one hundred items that meant one hundred comparisons. But when the list is ordered, the search only needs to move forward until an item is found whose value is greater than the item being searched. This is done in line 400, IF C$=B$(I) THEN 660, and line 410, IF C$<B$(I) THEN 640. These two lines make comparisons with each item in the array until the item is located or an item of greater value is detected.

The Binary Search

The third search routine in the program listing is called a binary search and is a very efficient search for long lists of ordered data. It is called binary not because it uses machine code, but because the maximum number of comparisons that it needs to make is represented by the power of 2 that results in a number greater than the number of items in the list. For example, in the program listing, there are 100 names taken from the phone book. 2↑7 is the next power of 2 larger than 100; therefore, the binary search will take a maximum of seven comparisons to locate the item in the list.

The first comparison is made with the middle item in the list. If the item being searched is greater than that item, then the upper half of the list becomes the new list. The second comparison is then made with the middle item of the upper section. This procedure of dividing the list in half is repeated until the item is located.

Here is an example of how it works. Suppose you want to locate the name "Usher" from the data in the program listing. There are 100 items in the list.

1. Comparison at item 50.
   Usher>Jones, so the new range is 50 to 100.
2. Comparison at item INT((100−50)/2+.5)+50 = 75.
   Usher>Peverill, so the new range is 75 to 100.
3. Comparison at item INT((100−75)/2+.5)+75 = 88.
   Usher>Stewart, so the new range is 88 to 100.
4. Comparison at item INT((100−88)/2+.5)+88 = 94.
   Usher<Ward, so the new range is 88 to 94.
5. Comparison at item INT((94−88)/2+.5)+88 = 91.
   Usher>Thomas, so the new range is 91 to 94.
6. Comparison at item INT((94−91)/2+.5+91 = 93.
   Usher<Vickruck, so the new range is 91 to 93.
7. Comparison at item INT((93−91)/2+.5)+91 = 92.
   Usher=Usher, so GOTO 660.

Either of the linear searches would have required 92 comparisons to locate Usher. You can see, therefore, that the binary search is quite powerful. You will discover that as lists become longer and longer, the binary search algorithm becomes much more powerful and efficient than the linear methods.

Explanation of the Program

100–220 Read and Display Data
230–300 Print Main Menu
310–360 Linear Search Routine
370–430 Alphabetical/Linear Search Routine
440–620 Binary Search Routine
630–760 Common Print Routines
770–840 Data Statements

Searching Algorithms

```
100 REM **SEARCHING ALGORITHMS**
120 DIM B$(100)
130 N=100
140 REM **READ AND DISPLAY DATA**
150 CALL CLEAR
160 FOR I=1 TO N
170 READ A$
180 B$(I)=A$
190 PRINT B$(I),
200 NEXT I
210 FOR T=1 TO 400
220 NEXT T
230 REM **PRINT MAIN MENU**
240 CALL CLEAR
250 PRINT "  **SEARCHING ALGORITHMS**": : :
    : :"PRESS{3 SPACES}FOR": : :"  1 = LIN
    EAR SEARCH": :
260 PRINT "  2 = ALPHA/LINEAR SEARCH": :"
     3 = BINARY SEARCH": :"  4 = FINISH
    SESSION": : : : : :
270 CALL KEY(0,KEY,S)
280 IF KEY<49 THEN 270
290 IF KEY>52 THEN 270
300 IF KEY=52 THEN 750 ELSE 700
310 REM **LINEAR SEARCH**
320 FOR I=1 TO N
330 PRINT I;
340 IF C$=B$(I)THEN 660
350 NEXT I
360 GOTO 640
370 REM **ALPHABETICAL LINEAR SEARCH**
380 FOR I=1 TO N
390 PRINT I;
400 IF C$=B$(I)THEN 660
410 IF C$<B$(I)THEN 640
420 NEXT I
430 GOTO 640
```

```
440 REM **BINARY SEARCH**
450 LOW=0
460 HIGH=N
470 K=1
480 X=INT(N/2+.5)
490 X=INT(X/2+.5)
500 K=K+1
510 IF X>1 THEN 490
520 I=0
530 FOR J=1 TO K
540 I=I+1
550 X=INT((HIGH-LOW)/2+.5)+LOW
560 PRINT X;
570 IF C$=B$(X)THEN 660
580 IF C$<B$(X)THEN 610
590 LOW=X
600 GOTO 620
610 HIGH=X
620 NEXT J
630 REM **GENERAL PRINT ROUTINES**
640 PRINT : : : :"SORRY,";C$:"IS NOT IN THE
    LIST."
650 GOTO 670
660 PRINT : : : :C$;",":"FOUND IN";I;"COMPAR
    ISONS."
670 PRINT : :"*PRESS ANY KEY TO CONTINUE*"
680 CALL KEY(0,KEY,S)
690 IF S=0 THEN 680 ELSE 240
700 CALL CLEAR
710 PRINT "THE NAME YOU ARE SEARCHING:": :
720 INPUT C$
730 PRINT : :"COMPARING WITH NAME #";
740 ON KEY-48 GOTO 320,380,450,750
750 CALL CLEAR
760 PRINT "{6 SPACES}HAVE A NICE DAY!": : :
    : : : : : : :
765 STOP
770 DATA ACKER,AINSLIE,ALLEN,ANDERSON,ARMSTR
    ONG,BANCROFT,BAULD,BEATON,BEATTIE,BLACK,
    BOWER,BROOKS,BROWN
780 DATA BURKE,CHANG,CHRISTIAN,CHU,COCHRANE,
    CODNER,COLLINS,COMEAU,COOK,COOPER,COX,DA
    RROW,DAVIS,DAY
790 DATA DELONG,DICKIE,DOGGETT,DOUGLAS,EBBET
    T,ELLIS,EMBREE,EULOTH,FIELD,FIFIELD,FOY,
    GAMMON,GREENE,HAPEMAN
800 DATA HARPELL,HARTLIN,HILL,HUBLEY,HUSKINS
    ,JAMES,JAMIESON,JOHNSON,JONES,KENDALL,KE
    TCHAM,KILLAWEE,KILLORAN
```

```
810 DATA LAMFORTH,LANGILLE,LERUE,LLOY,LYSEN,
    MACDONALD,MACFADYEN,MACFAWN,MACLACHLAN,M
    AILLET,MARSHALL,MASKELL
820 DATA MATTHEWS,MCCONNELL,MCDOWELL,MERCER,
    MOULTON,NAGLE,NAPER,NICKERSON,PEVERILL,P
    RESTON,PRICE,PROCTOR,RODDAM
830 DATA RONALDS,RUSSELL,SCHOEMAKER,SCHOFIEL
    D,SHERIDAN,SMITH,STARRATT,STEVENS,STEWAR
    T,SYKES,TAYLOR,THOMAS
840 DATA USHER,VICKRUCK,WARD,WEBB,WHITE,WHIT
    ING,WILBUR,WINTER,ZACHARY
```

Transferring Variables in TI Extended BASIC

Patrick Parrish

Variables can be passed from one program to another in most microcomputers by POKEing them into memory. But on the TI-99/4A, standard PEEKs and POKEs can't be used. Here's a way to transfer variables in TI Extended BASIC that uses redefined characters.

The TI-99/4A has outstanding graphic capabilities. With its subprogram CHAR, you can readily redefine characters within the standard ASCII character set (character codes 32–126). Or, you can create additional characters using codes 127–159 (codes 127–143 in Extended BASIC).

But there's a potentially more powerful application for the CHAR subprogram. Variable data can be passed from one program to another using CHAR and CHARPAT, an Extended BASIC subprogram. So, if you use up the TI's memory, it's now possible to write a program in two parts and send variables to a second program. Even the user's name could be among the variables transferred. But first let's take a brief look at the traditional use of the CHAR subprogram.

Defining Characters

On the TI-99/4A, characters are defined by a 16-character hexadecimal string expression known as a *pattern-identifier.* Pattern-identifiers are *dot* codes for depicting each character in an eight by eight grid (see the TI-99/4A *User's Reference Guide* pp. II-76 through II-79 for more).

Changing the pattern-identifier in memory for a character enables you to define that character to suit yourself. For example, suppose you wanted to represent the ASCII character 65 (normally, an A) as a box in a program. You could do this with the CHAR subprogram as:

```
CALL CHAR(65,"FFFFC3C3C3C3FFFF")
```

Within the parentheses following CALL CHAR is the ASCII character number (65) and the new pattern-identifier for the character ("FFFFC3C3C3C3FFFF"). By redefining characters in this manner, you can produce figures which greatly enhance and enliven screen displays in your programs.

Protected Memory

CHAR, within a program, can also be used to store variable data in the form of a pattern-identifier. Once stored, a second program can fetch this variable data with the CHARPAT subprogram.

CHARPAT is the converse of CHAR. Rather than specifying the pattern-identifier for an ASCII character, it returns from memory the pattern-identifier assigned to a particular character. For instance, CALL CHARPAT (65,A$) returns the pattern-identifier for ASCII character 65 as A$.

When you interrupt a TI program using redefined characters, certain character codes retain their redefined configuration while others return to their standard definitions. If you haven't seen this before, enter and run the following program:

```
100 CALL CLEAR
110 CALL CHAR(126,"FFFFFFFFFFFFFFFF")
120 CALL CHAR(127,"FFFFC3C3C3C3FFFF")
130 PRINT CHR$(126),CHR$(127)
140 FOR I=1 TO 1000
150 NEXT I
```

Here, we redefined character 126 (it's normally a tilde) in line 110 as a solid block and defined character 127 in line 120 as a hollow box. Next, we PRINTed both characters in line 130.

When you run this program, the two characters we've defined will appear on the screen momentarily. Once the program ends, the block character will change to a tilde while the box character remains.

Why does this happen? When a program is interrupted, only the standard character set on the TI (ASCII characters 32–126) is restored. Pattern-identifier data stored in ROM for characters 32–126 is copied to RAM (this process also occurs when the TI is first powered up or reset). As a result, ASCII character 126, seen as a block during program execution, becomes a tilde when our program ends. But character 127 (the box) retains its redefined shape.

Indeed, all characters above 127 will keep their defined form even if another program is run (provided these characters are not defined differently by this subsequent program).

Normally, the RUN command clears all variables in memory—both numeric and string. That is, all numeric variables become zero while string variables are set to null. So, if you chain to another program with RUN "device.program-name", the variables will be cleared. The fact that certain character codes remain intact even after a RUN will enable us to pass variables between programs by storing them as pattern-identifiers.

Storing Variable Data

The variable data that we wish to pass must be in hexadecimal form so that it can be stored as a pattern-identifier. Once it has been converted to hexadecimal form, it can be placed in the character codes beginning at 127 for retrieval by a second program.

Program 1 is a sample program which demonstrates the necessary routines for storing variable data in character codes. In this program, variables to be passed are generated in the main portion of the program (lines 100–798). In this case, we've simply assigned values to the three variables we want to transfer (line 500).

Two of these variables are numeric (X and Y) while the third is a string variable (NAME$). In line 800, the numeric variables are converted to string variables, and then all three variables are stored in the array D$. (Note in line 100 that we've DIMensioned D$ for the number of variables we intend to pass.) In line 900, we concatenate all values of D$() and store them in E$.

Seventeen character codes (codes 127–143) are available for variable storage. Each pattern-identifier is 16 hexadecimal characters in length, so we have room to store 272 (17 × 16) hexadecimal characters. Since 2 hexadecimal digits will be required to encode each character of E$, the length of E$ is limited to 136 characters (actually, 135 characters because the end of E$ is marked with an additional CHR$(255) in line 930).

After each D$() is concatenated to E$ and CHR$(255) is added as a separator between variables, a check of E$'s length is made in line 910. If the last variable added to E$ causes it to exceed 135 characters in length, the program will terminate, and the computer will display the number of variables you are allowed to transfer.

As mentioned, pattern-identifiers must be stored as hexadecimal code. Our best approach here is to represent each character of E$ by its ASCII value before converting it to hexadecimal.

Lines 1000 to 1020 contain routines for doing this. In line 1010, each character of E$ is converted to its ASCII equivalent. These ASCII values are, in turn, converted to a hexadecimal string expression, M1$, in line 1020.

Once M1$ reaches a length of 16 characters (or the end of E$ is reached), it is assigned as a pattern-identifier (line 1025). At this point, if M1$ is less than 16 characters long, TI Extended BASIC automatically fills the remaining characters in the pattern-identifier with zeros.

Recovering Variable Data

Variable data stored with Program 1 can be recovered with Program 2. Both programs serve as examples.

Again, you would place the main portion of your program in lines 100–798. Be sure to DIMension D$() and D() in line 100 for the number of variables you stored with Program 1.

Lines 800 to 980 contain routines for recalling each variable. In line 800, each pattern-identifier used to store data is assigned as A$ using CHARPAT. In line 910, if the end of variable data is detected as signified by "FFFF" (sequential CHR$(255)'s), a flag variable FL is set to 1.

Line 920 looks for the delimiter "FF" (CHR$(255)) following each variable D$. If a delimiter is seen, the length of the prior D$() is calculated as D().

Two characters of A$ are set equal to M$ in line 930. The two-digit hexadecimal number contained in M$ is subsequently converted to a decimal value in line 940. These decimal values are then converted to CHR$s in line 960 and stored as F$.

In line 1000, F$ is divided into D$()'s using lengths D(). As before, D$() represents the string form of each variable. Finally, as a demonstration, our original variables are PRINTed in line 1030. Of course, this may not be necessary in your program.

Program 1. Passing Variables

```
10 REM PROGRAM 1 (VARIABLE ORIGIN PROGRAM)
99 REM IE., LINES 100-798 = MAIN PORTION OF
   YOUR PROGRAM
100 OPTION BASE 1 :: DIM D$(3):: REM DIMENSI
    ON D$ FOR NUMBER OF VARIABLES TO TRANSFE
    R
499 REM VARIABLES IN LINE 500 ARE ASSIGNED W
    ITHIN THE MAIN PROGRAM
500 X=100 :: Y=-5.05 :: NAME$="JEFF TUDOR"
799 REM DEFINE STRING AND NUMERIC VARIABLES
    AS D$()
800 D$(1)=STR$(X):: D$(2)=STR$(Y):: D$(3)=NA
    ME$
810 HEX$="0123456789ABCDEF"
899 REM CONCATENATE D$()'S TO E$ AND DELIMIT
     WITH CHR$(255)
900 E$="" :: FOR I=1 TO 3 :: E$=E$&D$(I)&CHR
    $(255)
909 REM CHECK TO MAKE SURE LENGTH OF E$ DOES
     NOT EXCEED 136(272/2)
910 IF LEN(E$)>135 THEN E$=SEG$(E$,1,LEN(E$)
    -LEN(D$(I))-1):: PRINT "ONLY ";I-1 :: PR
    INT "VARIABLES CAN BE TRANSFERRED." :: S
    TOP
920 NEXT I
929 REM PLACE ADDITIONAL CHR$(255) AT END OF
     E$
930 E$=E$&CHR$(255)
999 REM CONVERT E$ TO ASC'S AND THEN TO HEX
    - CONCATENATE EVERY 16 AS M1$ OR END OF
    E$
1000 J=127 :: M1$="" :: FOR K=1 TO LEN(E$)
1010 D=ASC(SEG$(E$,K,1))
1020 MH=INT(D/16):: ML=D-MH*16 :: M$=SEG$(HE
     X$,MH+1,1)&SEG$(HEX$,ML+1,1):: M1$=M
1$&M$
1024 REM STORE HEX STRING M1$ AS CHAR PATTER
     N-IDENTIFIER
1025 IF (LEN(M1$)=16)+((LEN(E$)*2)=(J-127)*1
     6+LEN(M1$))THEN CALL CHAR(J,M1$):: J=J+
     1 :: M1$=""
1030 NEXT K
1040 CALL CLEAR :: PRINT "NOW RUN PROGRAM 2.
     "
```

Program 2. Receiving Variables

```
10 REM PROGRAM 2 (VARIABLE RECEPTOR PROGRAM)
99 REM IE., LINES 100-798 = MAIN PORTION OF
   THE PROGRAM
100 OPTION BASE 1 :: DIM D(3),D$(3):: REM DI
    MENSION D AND D$ FOR NUMBER OF VARIABLES
     TO RECEIVE
110 GOSUB 800 :: STOP
799 REM RECALL PATTERN IDENTIFIERS USED FOR
    VARIABLE STORAGE
800 K=1 :: P=1 :: FOR L=127 TO 143 :: CALL C
    HARPAT(L,A$)
899 REM SEPARATE A$'S INTO M$'S AND CONVERT
    BACK TO D$()'S
900 FOR I=1 TO LEN(A$)STEP 2
909 REM CHECK FOR END OF STRING AND SET FLAG
910 IF SEG$(A$,I,4)="FFFF" THEN FL=1
919 REM CHECK FOR DELIMITER . FROM THIS, DET
    ERMINE LENGTH OF EACH D$() - STORE AS D(
    )
920 IF SEG$(A$,I,2)="FF" THEN D(K)=(L-127)*1
    6+I-P :: K=K+1 :: P=(L-127)*16+I+2 :: IF
     FL=1 THEN I=LEN(A$):: GOTO 970
929 REM TAKE TWO CHARACTERS OF A$ AND CALL T
    HEM M$
930 M$=SEG$(A$,I,2)
939 REM CONVERT HEX STRING TO DECIMAL, THEN
    TO CHR$'S
940 M=0 :: FOR J=1 TO 2 :: M1=ASC(M$):: M1=M
    1-48+(M1>64)*7 :: M$=SEG$(M$,2,1):: M=16
    *M+M1
950 NEXT J
959 REM CONCATENATE ALL CHR$'S TO F$
960 F$=F$&CHR$(M)
970 NEXT I
975 IF FL=1 THEN L=143
980 NEXT L
999 REM DEFINE D$()'S USING D()'S AND F$, TH
    EN PRINT EACH D$().
1000 P=0 :: FOR I=1 TO 3 :: D$(I)=SEG$(F$,P+
     1,D(I)/2):: P=P+D(I)/2+1
1010 NEXT I
1020 X=VAL(D$(1)):: Y=VAL(D$(2)):: NAME$=D$(
     3)
1030 PRINT X,Y,NAME$
1040 RETURN
```

Computer Visuals

Richard D. Jones and Howard Alvir

Use your microcomputer to create effective visuals.

We give many presentations to groups of all sizes and have found that good graphics increase understanding of theoretical concepts and capture the interest of the audience. Recently the TI-99/4A joined the overhead projector and flipchart in our arsenal of visual aids.

Because of its small size, the TI-99/4A is very portable. A briefcase holds the computer, power cord, RF modulator, portable cassette recorder and cable. Rigging a snap connection for the RF cable and an extension cord will make setup a little easier. The television monitor is not as easily transported, so it's best to arrange to have it at the meeting site before you arrive.

Usually it takes 5–10 minutes prior to the meeting to set up (it usually takes that long to set up an easel tripod!). Make a few connections, load your program and begin. Since a 25-inch monitor can be easily seen from 30 feet, we have used the microcomputer with audiences of up to 75 people. Multiple monitors work well for larger audiences.

There are several advantages to using the computer visuals. First, visuals can be changed frequently and easily. (We are always changing presentations). Second, information is presented one point at a time. Third, since our presentations usually focus on technology, we practice what we preach. Fourth, the system is inexpensive and of high quality.

The following is a simple program illustration for presenting visuals. The routine organizes ten screens of information with words stored in DATA statements. Each screen can be called up a line at a time or the entire screen at once. The title screen is displayed initially and is set for full screen display. Each of the following screens is displayed in this sequence, unless called by the "F" key.

During display there are several function keys. These are as follows:

C—clear screen
F—display entire screen

space—scroll up
T—input additional words during display
1–9—calls appropriate screen
0—calls title screen

All other keys advance the screen a line at a time.

Color is added by the CALL COLOR command. As a result any character in character set 2 is displayed as color. Thirty-two characters will display a colored line.

Experiment with numerous screen variations (e.g., color combinations, larger letters, and speech to introduce major points). We have even experimented with using the speech synthesizer to open the presentation. Adding commands in Extended BASIC can improve graphics but it adds complexity to the equipment.

Generating visuals by computer opens exciting possibilities for the future. Certainly improvements in video display and microcomputers will expand the application of computer visuals. In the meantime, you can enter a new arena of professional computer use and discard your image as a hobbyist.

Computer Visuals

```
100 REM ********************
110 REM * COMPUTER VISUALS *
120 REM ********************
130 REM
140 REM
150 REM
160 REM
170 CALL CLEAR
180 PRINT TAB(4);"PRESS ANY KEY TO BEGIN"
190 CALL KEY(0,K,S)
200 IF S=0 THEN 190
210 CALL CLEAR
220 CALL COLOR(2,5,5)
230 CALL SCREEN(12)
240 RESTORE 1050
250 REM FULL SCREEN ROUTINE
260 CALL CLEAR
270 READ LINE$
280 IF LINE$="END" THEN 500
290 IF LINE$="#" THEN 970
300 IF LINE$="@" THEN 510
310 L=LEN(LINE$)
320 M=L/2
330 I=15-M
```

```
340 PRINT TAB(I);LINE$
350 PRINT
360 GOTO 270
370 REM
380 REM LINE ROUTINE
390 CALL CLEAR
400 READ LINE$
410 IF LINE$="END" THEN 500
420 IF LINE$="#" THEN 970
430 IF LINE$="@" THEN 370
440 L=LEN(LINE$)
450 M=L/2
460 I=15-M
470 PRINT TAB(I);LINE$
480 PRINT
490 GOTO 510
500 END
510 CALL KEY(0,K,S)
520 IF K=32 THEN 530 ELSE 550
530 PRINT
540 GOTO 510
550 IF K=67 THEN 560 ELSE 580
560 CALL CLEAR
570 GOTO 510
580 IF K=84 THEN 590 ELSE 640
590 PRINT
600 INPUT LINE$
610 GOTO 410
620 PRINT
630 GOTO 510
640 IF K=70 THEN 270
650 IF K=48 THEN 240
660 IF K=49 THEN 920
670 IF K=50 THEN 900
680 IF K=51 THEN 880
690 IF K=52 THEN 860
700 IF K=53 THEN 840
710 IF K=54 THEN 820
720 IF K=55 THEN 800
730 IF K=56 THEN 780
740 IF K=57 THEN 760
750 IF S=0 THEN 510 ELSE 950
760 RESTORE 1590
770 GOTO 400
780 RESTORE 1530
790 GOTO 400
800 RESTORE 1470
810 GOTO 400
820 RESTORE 1410
```

```
830 GOTO 400
840 RESTORE 1350
850 GOTO 400
860 RESTORE 1290
870 GOTO 400
880 RESTORE 1230
890 GOTO 400
900 RESTORE 1170
910 GOTO 400
920 RESTORE 1110
930 GOTO 400
940 GOTO 950
950 IF S=0 THEN 510
960 GOTO 400
970 CALL SOUND(100,294,5)
980 GOTO 510
990 REM DATA FOR SCREENS
1000 REM
1010 REM
1020 REM
1030 REM
1040 REM TITLE SCREEN
1050 DATA ----------------------------------
     ----------------------.,COMPUTER VISUAL
     S,USING THE
1060 DATA TI 99 4A  COMPUTER,
1070 DATA ----------------------------------
     ----------------------
1080 DATA
1090 DATA #,@
1100 REM SCREEN 1
1110 DATA ADVANTAGES
1120 DATA ----------------------------
1130 DATA EASY TO EDIT,INEXPENSIVE,ORGANIZED
     ,EFFICIENT,COLORFUL,PORTABLE,ATTENTION
     GATHERING
1140 DATA
1150 DATA #,@
1160 REM SCREEN 2
1170 DATA EQUIPMENT NEEDED,-----------------
     -----------,TI 99 4A,CASSETTE RECORDER,
     TELEVISION OR MONITOR.-----------------
     -----------
1180 DATA REALLY THAT IS ALL !!
1190 DATA
1200 DATA
1210 DATA #,@
1220 REM SCREEN 3
1230 DATA HOW TO USE
1240 DATA ----------------------------
```

```
1250 DATA OUTLINE PRESENTATION,ENTER KEY POI
     NTS IN DATA.SAVE THE PROGRAM,SET UP YOU
     R COMPUTER AND TV
1260 DATA LOAD AND RUN PROGRAM,IMPRESS YOUR
     AUDIENCE.------------------------------
1270 DATA #,@
1280 REM SCREEN 4
1290 DATA SCREEN 4
1300 DATA
1310 DATA
1320 DATA
1330 DATA #,@
1340 REM SCREEN 5
1350 DATA SCREEN 5
1360 DATA
1370 DATA
1380 DATA
1390 DATA #.@
1400 REM SCREEN 6
1410 DATA SCREEN 6
1420 DATA
1430 DATA
1440 DATA
1450 DATA #,@
1460 REM SCREEN 7
1470 DATA SCREEN 7
1480 DATA
1490 DATA
1500 DATA
1510 DATA #,@
1520 REM SCREEN 8
1530 DATA SCREEN 8
1540 DATA
1550 DATA
1560 DATA
1570 DATA #,@
1580 REM SCREEN 9
1590 DATA SCREEN 9
1600 DATA
1610 DATA
1620 DATA
1630 DATA #,@
1640 DATA END
```

Using a Printer

C. Regena

These tips will give you a good start on adding a printer to the TI-99/4A. Here are the fundamentals, from the RS-232 Interface to PRINT # statements.

Texas Instruments has a thermal printer which attaches to the side of the TI. It's a small unit which uses a special thermal printer paper and can print a 30-column line. A number of other printers may also be used with your TI. The price depends on whether the printing is dot-matrix or letter quality, on various options available, and on how the printer is built.

To connect your printer to your TI-99/4A, you will need the RS-232 Interface. You may use either the "old-style" individual RS-232 Interface peripheral or the RS-232 Interface Card which fits in the TI Peripheral Expansion Box. You will also need a cable to go from the interface to the printer, and the cable should be sold with the printer. If you want to wire your own cable, the plug is a standard DB-25, and the pin connections are given in the manual that comes with the RS-232 Interface.

Configurations

Manuals are important. The manual that comes with the RS-232 Interface describes how you list parameters for your "printer configuration" so you can give instructions to your computer to access the printer through the RS-232. The manual that comes with the printer should describe how to achieve various type styles (fonts) and how to set margins, line lengths, and the top of the form. Be prepared to spend some time experimenting with the different switches and features of your printer.

When you use the printer configuration in a command, it is set off in quotes. Parameters may be chosen for baud rate, stop bits, and number of nulls. Some examples are:

"RS232.TW.BA=110" (teletype)
"RS232.BA=600" (TI 825 or TI 840 printer)
"RS232.BA=9600.DA=8" (Epson MX 80)

One of the primary uses of a printer is to obtain a hardcopy listing of a program. Using your own printer configuration in the quotes, the following commands may be used:

LIST "RS232.BA=600"
Lists whole program
LIST "RS232.BA=600":-250
Lists program lines up to line 250
LIST "RS232.BA=600":300-330
Lists program lines 300 to 330
LIST "RS232.BA=600":700-
Lists program from line 700 to end

Another valuable use for a printer is to print a report from your program. Before you print, an OPEN statement is necessary. The OPEN statement designates a device number and your printer configuration. You may have several devices, and you may number your devices in any order. An example statement is:

120 OPEN #1:"RS232.BA=600"

After the OPEN statement, you may print to the printer by a statement such as:

130 PRINT #1:"MY NAME IS REGENA."

When you've finished printing or you're at the end of the program, you should close all devices. This can be done with the following statement:

550 CLOSE #1

Here is a short sample program that illustrates printing to a printer:

100 OPEN #1:"RS232.BA=600"
Opens device 1 for printer.
110 OPEN #2:"SPEECH",OUTPUT
Opens device 2 for speech (Terminal Emulator II required).
120 PRINT "HERE IS A SAMPLE."
Prints message on screen.
130 PRINT #1:"TEST REPORT
Prints on printer.
140 PRINT #2:"HELLO"
Speaks the word using synthesizer.
150 CLOSE #1
Closes device 1.

160 CLOSE #2
Closes device 2.
170 END

The print list following the colon in a PRINT # statement follows the same rules as regular printing to the screen. Since the length of lines may be longer on the printer (the screen has 28 columns in a print line), you may use the TAB function to arrange your printing:

```
100 OPEN #1:"RS232.BA=600"
110 PRINT #1:TAB(25):"MONTHLY PAYMENTS"
```

You may use a variable in the TAB function:

```
200 PRINT #1:TAB(T+A);MONTH$;X
```

You may also use colons to print blank lines:

```
220 PRINT #1:::
```

If you have adjusted your printer properly for vertical tabs, you may go to the top of the next page by using:

```
300 PRINT #1:CHR$(12)
```

3
Applications

3
Mailing List

Doug Hapeman

This program can be used for developing small mailing lists, for families or for organizations. There are ten options, including printing a single label or an entire alphabetized mailing list.

Have you ever kept a file of addresses on index cards, hoping to organize them someday in an orderly fashion? It sounds simple, but in practice you know how difficult it is to organize and update a paper-based filing system. "Mailing List" offers you an easy method of creating, maintaining, and utilizing a mailing list file.

Without any programming experience you can keep an up-to-date, well-organized file. The program will prompt you step-by-step through the entry of names, addresses, and phone numbers. Then, with a few simple keystrokes, you can update your file, print lists in two different modes, or save your file on a storage device. It's that easy.

Mailing List is designed specifically as a family mailing list, but is flexible enough to accommodate a number of applications. The program will store last names, first names, children's names, addresses, and phone numbers.

The program is written in a Canadian format—that is, province and postal code. However, the format can be easily adjusted to the American system as you type in the program.

Program Environment

The program is set up for 45 entries. After 45 entries you will be given the message *DATA FILE IS FULL*. This feature will prevent your program from crashing with a MEMORY FULL error message. If you have more than 45 addresses to enter, you may easily divide your list into two or more files—for example: (A–L) and (M–Z).

When you run the program, the initial title screen appears. The next display permits you to initialize the printer. Be sure to enter the proper name and spelling of the device you're using, because an improper name will cause the program to break when you attempt to address the device later in the program.

Ten Options

Once the computer "environment" is established, you are taken to the Main Index. Here you will discover ten options:

 1 View Names List
 2 Search for a Name
 3 Add Names
 4 Change Names
 5 Delete Names
 6 Alphabetize List
 7 Save Data File
 8 Load Data File
 9 Print Labels/List
10 Finish Session

Of course, to create a mailing list you would first choose option 3 (Add Names). The other options will enable you to update, maintain, and utilize an existing file. The program will guide you step-by-step through the procedure for each option. There are many helpful features, such as the Search, Change, and Delete. You can also enter names and addresses in any order, and then, by choosing the Alphabetize option, have the computer sort them for you.

The Data File

The program is written to both save and load data files for either cassette or disk storage. When you choose either the Save or Load option, you will be given any further step-by-step instructions.

Print Options

The program offers you two print options—one for mailing labels, and the other for the mailing list.

The Print Labels option will print the first name, followed by the last name, and then the address on lines two and three. For example:

John Doe
1234 Street Address
City Province Postal Code

The Print Mailing List option will print the last name first, followed by the first name and children's names, with the address on line two, and the phone number on line three. For example:

Doe, John Mary Joe Sally
1234 Address City Province Postal Code
(p)-444/4456

Line spacing between addresses is flexible via a minor program change. If you wish to alter the line spacing, program lines 497 (labels) and 517 (list) may be adjusted by either increasing or decreasing the number of colons (:) at the end of each line. Each colon represents one line space. For example:

```
497 PRINT #2:TAB(5);NA$(I);" ";LN$(I):TAB(5);AD$(I):TAB(5);
    CP$(I);" ";PC$(I):::: (Add or delete colons here.)
```

In the Print Labels option, you may wish to print two labels per line instead of one. If so, you should adjust the line listing as follows:

Change line 487 to:

```
87 FOR I=1 to N STEP 2
```

Change line 497 to:

```
497 PRINT #2:TAB(5);NA$(I);" ";LN$(I);TAB(45);NA$(I+1);
    " ";LN$(I+1):TAB(5);AD$(I);TAB(45);AD$(I+1)
```

Add line 498:

```
498 PRINT #2:TAB(5);CP$(I);" ";PC$(I);TAB(45);CP$(I+1);
    " ";PC$(I+1)::::
```

The Search option permits the printing of a single mailing label. After finding the name you are seeking, the display asks if you would like a mailing label printed. If yes, the program branches to the print routine and then returns to the search option.

Mailing List Program Structure

1–21 REMs and computer environment.
23–47 Main loop, main index.
49–73 Subroutine to view names.
75–109 Subroutine to search for a name.
111–181 Subroutine to add names.
183–285 Subroutine to change data.
287–331 Subroutine to delete names.
333–423 Subroutine to alphabetize list.
425–441 Subroutine to save data.
443–471 Subroutine to load data.
473–521 Subroutine to print.
523–533 Subroutine to finish session.

Mailing List

```
5 REM  **COMPUTER ENVIRONMENT**
7 DIM LN$(45),NA$(45),CH$(45),AD$(45),CP$(45
  ),PC$(45),TP$(45)
9 CALL CLEAR
11 PRINT " *{3 SPACES}99/4A MAILING LIST
   {3 SPACES}*": : : : : : : : : : : :
13 INPUT "{4 SPACES}PRESS ENTER TO BEGIN":X$
15 CALL CLEAR
17 PRINT "{5 SPACES}WHAT IS THE NAME OF":"
   {4 SPACES}YOUR PRINTING DEVICE?": :"  (EX
   AMPLE: RS232.BA=4800)": : : : : : : : : :
    : :
19 INPUT P$
21 G$="{7 SPACES}PLEASE WAIT...{7 SPACES}WHI
   LE THE PRINTER IS WORKING"
23 REM  **MAIL LIST MENU**
25 CALL CLEAR
27 PRINT "{8 SPACES}MAIN INDEX": : : :
29 PRINT "PRESS{3 SPACES}TO": : :
31 PRINT "  1  =  VIEW NAMES LIST":"  2  =
   SEARCH FOR A NAME":"  3  =  ADD NAMES":"
    4  =  CHANGE NAMES"
33 PRINT "  5  =  DELETE NAMES":"  6  =  ALP
   HABETIZE LIST":"  7  =  SAVE DATA FILE":"
     8  =  LOAD DATA FILE"
35 PRINT "  9  =  PRINT LABELS/LIST":" 10  =
     FINISH SESSION": : : :
37 INPUT P
39 IF P>10 THEN 37
41 IF P<1 THEN 37
43 CALL CLEAR
45 ON P GOSUB 51,77,113,185,289,335,427,445,
   475,525
47 GOTO 25
49 REM{3 SPACES}**VIEW NAMES LIST**
51 T=0
53 FOR I=1 TO N
55 T=T+1
57 PRINT NA$(I),LN$(I):CH$(I):AD$(I):CP$(I):
   PC$(I):"(P)-";TP$(I): : :
59 IF T<2 THEN 69
61 PRINT " *PRESS ENTER TO CONTINUE*":" *""R
   "",ENTER FOR MAIN INDEX*"
63 INPUT X$
65 IF X$="R" THEN 73
67 T=0
```

```
69 NEXT I
71 INPUT "{7 SPACES}*END OF FILE*{9 SPACES}*
   PRESS ENTER TO CONTINUE*":X$
73 RETURN
75 REM{3 SPACES}**SEARCH NAMES**
77 INPUT "LAST NAME? ":Y$
79 FOR I=1 TO N
81 IF LN$(I)<>Y$ THEN 1Ø3
83 PRINT : : :"  IS THE PERSON:": :"   ";NA$(
   I):"  ";LN$(I): :
85 INPUT "  (Y/N)?":X$
87 IF X$="N" THEN 1Ø3
89 PRINT : : :NA$(I),LN$(I):CH$(I):AD$(I):CP
   $(I):PC$(I):"(P)-";TP$(I): : :
91 INPUT "{3 SPACES}DO YOU WISH TO PRINT
   {6 SPACES}A MAILING LABEL?  (Y/N)":Z$
93 IF Z$<>"Y" THEN 97
95 GOSUB 495
97 INPUT "SEARCH MORE NAMES?  (Y/N)":X$
99 IF X$="Y" THEN 77
1Ø1 GOTO 1Ø9
1Ø3 NEXT I
1Ø5 PRINT : : :"  THE ";Y$:"  YOU ARE SEARCH
    ING FOR":"  IS NOT IN THIS FILE.": : :
1Ø7 GOTO 97
1Ø9 RETURN
111 REM{3 SPACES}**ADD NAMES**{5 SPACES}
113 A=N+1
115 FOR I=A TO 45
117 CALL CLEAR
119 PRINT : : : :"ENTER DATA: ";"#";I;"  (MA
    X:45)": : :
121 PRINT "  *LAST NAME:"
123 INPUT LN$(I)
125 PRINT :"  *FIRST NAME(S):"
127 INPUT NA$(I)
129 PRINT :"  *CHILDREN:":"{3 SPACES}NOTE--D
    O NOT USE COMMAS!"
131 INPUT CH$(I)
133 PRINT :"  *STREET ADDRESS:"
135 INPUT AD$(I)
137 PRINT :"  *CITY/PROVINCE:":"{3 SPACES}NO
    TE--DO NOT USE COMMAS!"
139 INPUT CP$(I)
141 PRINT :"  *POSTAL CODE:"
143 INPUT PC$(I)
145 PRINT :"  *PHONE:"
147 INPUT TP$(I)
149 V=I
```

```
151 REM  **VERIFY ENTRIES**
153 CALL CLEAR
155 PRINT "ENTRY";"#";V: : :
157 PRINT "YOU ENTERED:": :"  ";LN$(V);", ";
    NA$(V):"  ";CH$(V):"  ";AD$(V):"  ";CP$(
    V)
159 PRINT "  ";PC$(V):"  PHONE: ";TP$(V): :
    : : : : :
161 INPUT "CHANGE ANYTHING?  (Y/N)":X$
163 IF X$<>"Y" THEN 171
165 C=N+1
167 CALL CLEAR
169 GOSUB 201
171 INPUT "ADD MORE NAMES?  (Y/N)":X$
173 N=N+1
175 IF X$="N" THEN 181
177 NEXT I
179 INPUT "{4 SPACES}*DATA FILE IS FULL*
    {6 SPACES}*PRESS ENTER TO CONTINUE*":X$
181 RETURN
183 REM{3 SPACES}**CHANGE DATA**
185 PRINT "  LAST NAME OF THE PERSON
    {3 SPACES}WHOSE DATA IS TO BE CHANGED:":
     : : :
187 INPUT C$
189 CALL CLEAR
191 FOR C=1 TO N+1
193 IF LN$(C)=C$ THEN 195 ELSE 239
195 PRINT "IS THE PERSON:":"  ";NA$(C):"  ";
    LN$(C): :
197 INPUT "  (Y/N)?":X$
199 IF X$="Y" THEN 201 ELSE 239
201 PRINT : : : : : : :"PRESS{3 SPACES}TO CH
    ANGE": :
203 PRINT "  1  =  LAST NAME":"  2  =  FIRST
     NAME(S)":"  3  =  CHILDREN":"  4  =  ST
    REET ADDRESS"
205 R=C
207 R$="  *ENTER THE NEW DATA:"
209 PRINT "  5  =  CITY/PROVINCE":"  6  =  P
    OSTAL CODE":"  7  =  PHONE":"  8  =  NO
    CHANGE": : : : : : :
211 INPUT P
213 CALL CLEAR
215 IF P<1 THEN 211
217 IF P>8 THEN 211
219 IF P=8 THEN 229
221 ON P GOSUB 245,251,257,263,269,275,281
223 PRINT : :"MORE CHANGES FOR:":"  ";NA$(R)
    :"  ";LN$(R): :
```

```
225 INPUT "   (Y/N)?":Y$
227 IF Y$<>"N" THEN 201
229 PRINT : : :"CHANGE DATA FOR OTHER NAMES?
    ": : :
231 INPUT "   (Y/N)":Z$
233 CALL CLEAR
235 IF Z$<>"N" THEN 185
237 RETURN
239 NEXT C
241 RETURN
243 REM  **CHANGE LOOPS**
245 PRINT "LAST NAME WAS:": :LN$(R): : :R$
247 INPUT LN$(R)
249 RETURN
251 PRINT "FIRST NAME(S) WERE:": :NA$(R): :
    :R$
253 INPUT NA$(R)
255 RETURN
257 PRINT "CHILDREN WERE:": :CH$(R): : :R$
259 INPUT CH$(R)
261 RETURN
263 PRINT "ADDRESS WAS:": :AD$(R): : :R$
265 INPUT AD$(R)
267 RETURN
269 PRINT "CITY/PROVINCE WAS:": :CP$(R): : :
    R$
271 INPUT CP$(R)
273 RETURN
275 PRINT "POSTAL CODE WAS:": :PC$(R): : :R$
277 INPUT PC$(R)
279 RETURN
281 PRINT "PHONE NUMBER WAS:": :TP$(R): : :R
    $
283 INPUT TP$(R)
285 RETURN
287 REM{3 SPACES}**DELETE NAMES**
289 INPUT "LAST NAME? ":X$
291 FOR I=1 TO N
293 IF LN$(I)<>X$ THEN 325
295 PRINT : : :"IS THE PERSON:":"   ";NA$(I):
    "   ";LN$(I): :
297 INPUT "   (Y/N)?":Y$
299 IF Y$<>"Y" THEN 325
301 A=I
303 FOR D=A TO N-1
305 LN$(D)=LN$(D+1)
307 NA$(D)=NA$(D+1)
309 CH$(D)=CH$(D+1)
311 AD$(D)=AD$(D+1)
313 CP$(D)=CP$(D+1)
```

```
315 PC$(D)=PC$(D+1)
317 TP$(D)=TP$(D+1)
319 NEXT D
321 N=N-1
323 GOTO 327
325 NEXT I
327 INPUT "MORE DELETIONS? (Y/N)":X$
329 IF X$="Y" THEN 289
331 RETURN
333 REM  **ALPHABETIZE LIST**{3 SPACES}
335 PRINT "{7 SPACES}PLEASE WAIT...": : :" T
    HE LIST IS BEING ARRANGED": : : : : :
    : : :
337 B=1
339 B=2*B
341 IF B<=N THEN 339
343 B=INT(B/2)
345 IF B=Ø THEN 369
347 FOR Y=1 TO N-B
348 X=Y
349 I=X+B
351 IF LN$(X)=LN$(I)THEN 363
353 IF LN$(X)<LN$(I)THEN 365
355 GOSUB 381
357 X=X-B
359 IF X>Ø THEN 349
361 GOTO 365
363 GOSUB 373
365 NEXT Y
367 GOTO 343
369 RETURN
371 REM  **ORDER FIRST NAMES**{3 SPACES}
373 IF NA$(X)<NA$(I)THEN 377
375 GOSUB 381
377 RETURN
379 REM  **CHANGE ORDER**
381 N$=LN$(X)
383 LN$(X)=LN$(I)
385 LN$(I)=N$
387 N$=NA$(X)
389 NA$(X)=NA$(I)
391 NA$(I)=N$
393 N$=CH$(X)
395 CH$(X)=CH$(I)
397 CH$(I)=N$
399 N$=AD$(X)
401 AD$(X)=AD$(I)
403 AD$(I)=N$
405 N$=CP$(X)
407 CP$(X)=CP$(I)
```

```
409 CP$(I)=N$
411 N$=PC$(X)
413 PC$(X)=PC$(I)
415 PC$(I)=N$
417 N$=TP$(X)
419 TP$(X)=TP$(I)
421 TP$(I)=N$
423 RETURN
425 REM  **SAVE DATA FILE**{5 SPACES}
427 GOSUB 467
429 OPEN #1:L$,INTERNAL,OUTPUT,FIXED 150
431 PRINT #1:N
433 FOR I=1 TO N
435 PRINT #1:LN$(I),NA$(I),CH$(I),AD$(I),CP$
    (I),PC$(I),TP$(I)
437 NEXT I
439 CLOSE #1
441 RETURN
443 REM{3 SPACES}**LOAD DATA FILE**
    {6 SPACES}
445 GOSUB 467
447 OPEN #1:L$,INTERNAL,INPUT ,FIXED 150
449 INPUT #1:N
451 FOR I=1 TO N
453 INPUT #1:LN$(I),NA$(I),CH$(I),AD$(I),CP$
    (I),PC$(I),TP$(I)
455 NEXT I
457 CLOSE #1
459 CALL CLEAR
461 PRINT "  ";L$: :"  THIS FILE HAS";N;"ENT
    RIES.": :"  *45 ENTRIES IS MAXIMUM*": :
    : : : : : : : : :
463 INPUT " *PRESS ENTER TO CONTINUE*":X$
465 RETURN
467 PRINT "{5 SPACES}WHAT IS THE NAME OF":"
    {4 SPACES}YOUR STORAGE DEVICE?": :"(EXAM
    PLE: CS1 OR DSK1.FILE)": : : : : : : : :
     :
469 INPUT L$
471 RETURN
473 REM **SUB TO PRINT LABELS/LIST**
475 PRINT "PRESS{3 SPACES}TO PRINT": : :"  1
    {5 SPACES}MAILING LABELS": :"  2
    {5 SPACES}MAILING LIST": : : : : : : :
477 INPUT P
479 IF P<1 THEN 477
481 IF P>2 THEN 477
483 PRINT : : : : : : : : : : : :G$: : : : :
     : : : : :
```

```
485 IF P<>1 THEN 5Ø5
487 FOR I=1 TO N
489 GOSUB 495
491 NEXT I
493 RETURN
495 OPEN #2:P$
497 PRINT #2:TAB(5);NA$(I);" ";LN$(I):TAB(5)
    ;AD$(I):TAB(5):CP$(I);"  ";PC$(I): : : :
499 CLOSE #2
5Ø1 RETURN
5Ø3 REM{3 SPACES}**PRINT MAIL LIST**
5Ø5 FOR I=1 TO N
5Ø7 GOSUB 513
5Ø9 NEXT I
511 RETURN
513 OPEN #2:P$
515 PRINT #2:TAB(5);LN$(I);", ";NA$(I);"
    {6 SPACES}":CH$(I):TAB(5);AD$(I);"
    {3 SPACES}";CP$(I);"  ";PC$(I)
517 PRINT #2:TAB(6Ø);"(P)-";TP$(I): :
519 CLOSE #2
521 RETURN
523 REM  **FINISH SESSION**{5 SPACES}
525 INPUT "{7 SPACES}DO YOU WISH TO
    {1Ø SPACES}TERMINATE THIS SESSION?
    {5 SPACES}(Y/N)":X$
527 CALL CLEAR
529 IF X$<>"Y" THEN 25
531 PRINT "{6 SPACES}HAVE A NICE DAY!": : :
    : : : : : : :
533 STOP
```

Statistics For Nonstatisticians

A. Burke Luitich
TI Translation by Patrick Parrish

Basic statistical methods can help you make logical decisions in everyday situations.

For the most part, elementary statistical methods measure a group of similar things to see how these measurements vary when compared to some standard. Another use for statistics is to see how creating a group of objects can cause variations in these objects.

This program, "Statistics," takes your raw data and returns figures which you can use to make everyday decisions, for example, about the best way to build a wall or how much cash you'll need when you go shopping.

As a first example, let's look at two ways to cut a two-by-four by using a power table saw and a handsaw. We set the table saw guide to one foot and cut five pieces. We cut five more pieces using a handsaw, then measure the actual lengths of all ten pieces to see how accurately we made the cuts.

If nothing unusual is allowed to affect the cutting, we can expect the length of the pieces to vary depending on the process used. Statisticians call this an *unbiased random sample.*

Assume the measurements are as follows:

Table saw lengths (feet)	**Handsaw lengths** (feet)
1.05	1.22
0.98	0.91
1.03	0.80
1.07	1.28
0.96	0.88

The Same Mean

A look at the values alone suggests that cutting with the handsaw is a far less consistent method than using the table saw. However, if you add up the lengths for each method and divide by 5 (the total cuts for each) you will find that both methods give the same *mean* (average) length of 1.018 feet.

Just finding an average length doesn't tell us much. What we need to know is how widespread the values are likely to be, and which method gave us the most lengths that were nearer our standard of one foot. In statistical terms, we need to calculate the *range* and the *standard deviation.*

We find the range by subtracting the shortest length from the longest, for each cutting method. For the handsaw the range is .48 feet (1.28−0.80), and for the table saw the range is .11 feet (1.07−0.96). Immediately, we can see that the table saw cut more consistently, because the range, or variation, is smaller.

We can use the standard deviation and the mean length to predict how often a given length is likely to occur. You don't have to worry about how to calculate a standard deviation: The program does this for you. If you type in the above lengths for the handsaw, the program will return a standard deviation of 0.217 feet. The standard deviation for the table saw is 0.047 feet.

Degree of Accuracy

If we made a large number of cuts, then measured and graphed the lengths, the graph would form a bell curve, or normal distribution. By combining the standard deviation and the mean length, we get a range of lengths that includes 68.3 percent of all lengths (again, you don't have to know the theory; just use the number). To illustrate, first take the mean length, 1.018 feet, and subtract from it the standard deviation for the handsaw, 0.217 feet, to get 0.801 feet. Then add the standard deviation to the mean length to get 1.235 feet. This means that 68.3 percent of our lengths fall in the range between 0.801 and 1.235 feet.

By adding and subtracting the standard deviation (0.047 feet) with the mean length of the table saw cuts (1.018 feet), we find that 68.3 percent (roughly two-thirds) of these lengths fall in the range from 0.971 to 1.065 feet.

If you want a wider sample, you must increase the number of standard deviations. To include 95.4 percent of all lengths, use two standard deviations. For the handsaw, we now have 0.434 feet, two standard deviations. Combining it with the mean length, we get a range of 0.584 to 1.452 feet.

Our table saw range becomes 0.924 to 1.102 feet (1.018 ± 0.094).

Food For Thought

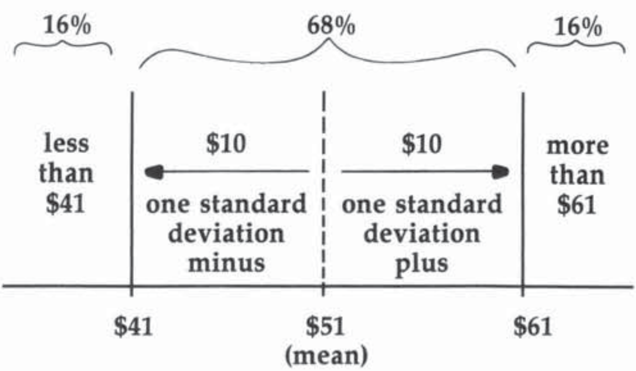
You can use the same methods to calculate a food budget. In this case, your data consists of the amounts you spent on groceries over a 13-week period (one-fourth of a year):

Week	Amount	Week	Amount
1	$42	8	47
2	50	9	65
3	75	10	49
4	37	11	43
5	51	12	52
6	45	13	54
7	56		

If you type this data into the Statistics program, you will find that your mean amount spent was about $51; that your spending varied from $37 to $75, for a range of $38; that you spent more than $50 (your medium amount) as often as you spent less than that; and your standard deviation is about $10.

Applying the Statistics

Combining one standard deviation and the mean (or average) amount spent, we find that two-thirds of the weeks you spend between $41 and $61 at the grocery store. One-sixth of the time you spend less than $41; one-sixth of your bills are more than $61. So, if you budget $61 for groceries, you'll have enough 84 percent of the time.



If you want to be sure you'll have enough in case prices rise, you might want to use two standard deviations. By adding two standard deviations ($20) to the mean amount ($51), you will find that, to be about 98 percent sure, you should budget $71 each week.

There are other factors to be considered, of course, such as vacations, birthday parties, or visiting relatives, that can affect your food budget. The Statistics program does not take these kinds of things into account. But it does give you a tool which takes some of the guesswork out of everyday decision making.

Statistics

```
100 DIM SA(300)
110 CALL CLEAR
120 PRINT TAB(10):"STATISTICS"
130 PRINT : : :
140 PRINT TAB(13):"FOR"
150 PRINT : : :
160 PRINT TAB(7):"NON-STATISTICIANS"
170 PRINT : : : : : :
180 FOR K=1 TO 400
190 NEXT K
200 CALL CLEAR
210 PRINT "THIS PROGRAM CALCULATES THE": :
220 PRINT "FOLLOWING VALUES FROM DATA": :
230 PRINT "YOU INPUT:"
240 PRINT : :
250 PRINT TAB(4):"1. MEAN"
260 PRINT : :
270 PRINT TAB(4):"2. STANDARD DEVIATION"
280 PRINT : :
290 PRINT TAB(4):"3. MEDIAN"
300 PRINT : :
310 PRINT TAB(4);"4. RANGE"
320 PRINT : : :
330 PRINT TAB(2);"PRESS ANY KEY TO CONTINUE"
340 PRINT :
350 GOSUB 2170
360 SUM=0
370 MEAN=0
380 DFF=0
390 SDDEV=0
400 RG=0
410 REM INSTRUCTIONS REQUEST
420 PRINT TAB(6):"INSTRUCTIONS (Y/N)?"
430 PRINT : : : : : : : : : :
```

```
440 GOSUB 2170
450 IF (K<>89)*(K<>78)THEN 440
460 IF K=78 THEN 490
470 GOSUB 1330
480 REM DATA ENTRY
490 CALL CLEAR
500 PRINT TAB(3);"ENTER SAMPLE SIZE ":
510 INPUT N
520 IF (N>300)+(N<=1)THEN 490
530 CALL CLEAR
540 PRINT TAB(3);"ENTER YOUR DATA ONE VALUE"
    : :
550 PRINT "AT A TIME, THEN PRESS": :
560 PRINT "RETURN.": : : :
570 PRINT TAB(3);"IF YOU MAKE AN ERROR,": :
580 PRINT "CONTINUE WITH DATA ENTRY.": :
590 PRINT "YOU WILL BE ABLE TO MAKE": :
600 PRINT "CORRECTIONS LATER.": : : :
610 PRINT TAB(2);"PRESS ANY KEY TO CONTINUE"
    : :
620 GOSUB 2170
630 FOR I=1 TO N
640 CALL CLEAR
650 PRINT "DATA ENTRY #";I;
660 INPUT R$
670 SA(I)=VAL(R$)
680 NEXT I
690 REM ERROR CORRECTION REQUEST
700 CALL CLEAR
710 PRINT TAB(3);"ANY CORRECTIONS (Y/N) ?"
720 PRINT : : : : : : : : : :
730 GOSUB 2170
740 IF K<>89 THEN 770
750 GOSUB 1800
760 REM CALCULATION OF MEAN AND STD. DEVIATI
    ON
770 PRINT TAB(9);"PLEASE WAIT": :
780 PRINT "STATISTICS BEING CALCULATED"
790 PRINT : : : : : : : : :
800 FOR I=1 TO N
810 SUM=SUM+SA(I)
820 NEXT I
830 MEAN=SUM/N
840 FOR I=1 TO N
850 DFF=DFF+(SA(I)-MEAN)^2
860 NEXT I
870 SDDEV=SQR(DFF/(N-1))
880 REM SORT OF DATA INTO NUMERIC ORDER
890 FL=0
900 FOR I=1 TO N-1
```

```
91Ø IF SA(I)<=SA(I+1)THEN 96Ø
92Ø Q=SA(I)
93Ø SA(I)=SA(I+1)
94Ø SA(I+1)=Q
95Ø FL=1
96Ø NEXT I
97Ø IF FL=1 THEN 89Ø
98Ø REM CALCULATION OF RANGE
99Ø RG=SA(N)-SA(1)
1ØØØ LR=SA(1)
1Ø1Ø HR=SA(N)
1Ø2Ø REM CALCULATION OF MEDIAN
1Ø3Ø IF N/2<>INT(N/2)THEN 1Ø9Ø
1Ø4Ø IF SA(N/2)<>SA(N/2+1)THEN 1Ø6Ø
1Ø5Ø MDD=SA(N/2)
1Ø6Ø IF SA(N/2)=SA(N/2+1)THEN 1Ø8Ø
1Ø7Ø MDD=(SA(N/2)+SA(N/2+1))/2
1Ø8Ø GOTO 111Ø
1Ø9Ø MDD=SA(INT(N/2+1))
11ØØ REM PRINT RESULTS TO SCREEN
111Ø CALL CLEAR
112Ø PRINT TAB(5);"CALCULATION RESULTS": :
113Ø PRINT "****************************": :
     :
114Ø PRINT "SAMPLE SIZE";TAB(19);N: :
115Ø PRINT "MEAN (X BAR)";TAB(19);INT(MEAN*1
     ØØØØ+.5)/1ØØØØ: :
116Ø PRINT "STD. DEVIATION";TAB(19);INT(SDDE
     V*1ØØØØ+.5)/1ØØØØ: :
117Ø PRINT "MEDIAN";TAB(19);INT(MDD*1ØØØØ+.5
     )/1ØØØØ: :
118Ø PRINT "RANGE";TAB(19);INT(RG*1ØØØØ+.5)/
     1ØØØØ: :
119Ø PRINT "LOWEST VALUE";TAB(19);LR: :
12ØØ PRINT "HIGHEST VALUE";TAB(19);HR: : : :
121Ø PRINT TAB(8);"PRESS ANY KEY"
122Ø GOSUB 217Ø
123Ø REM REQUEST TO CONTINUE OR END
124Ø PRINT " WISH TO PROCESS MORE DATA": :
125Ø PRINT TAB(12);"(Y/N)?": : : : : : : : :
     :
126Ø GOSUB 217Ø
127Ø IF K=78 THEN 132Ø
128Ø FOR I=1 TO N
129Ø SA(I)=Ø
13ØØ NEXT I
131Ø GOTO 36Ø
132Ø END
133Ø PRINT TAB(3);"THE MAXIMUM NUMBER OF EN-
     ": :
```

```
1340 PRINT "TRIES YOU CAN MAKE IS 300.": :
1350 PRINT "THE MINIMUM NUMBER IS 2.": : :
1360 PRINT TAB(3);"THE MEAN IS THE ARITH-":
     :
1370 PRINT "METIC AVERAGE OF THE NUMBERS": :
1380 PRINT "YOU ENTER.": : :
1390 PRINT TAB(3);"STANDARD DEVIATION IS A":
      :
1400 PRINT "MEASURE OF HOW WIDELY YOUR": :
1410 PRINT "NUMBERS SPREAD FROM THE": :
1420 PRINT "AVERAGE.": : :
1430 GOSUB 2160
1440 CALL CLEAR
1450 PRINT TAB(3);"SINCE THE VALUES YOU ENTE
     R": :
1460 PRINT "TEND TO FORM A BELL CURVE": :
1470 PRINT "(NORMAL DISTRIBUTION), THE": :
1480 PRINT "STD. DEVIATION IS A MEASURE": :
1490 PRINT "OF THE AREA UNDER THE BELL": :
1500 PRINT "CURVE.": : :
1510 PRINT TAB(4);"NO. OF STD.{4 SPACES}% AR
     EA"
1520 PRINT TAB(5);"DEV.(+/-)"
1530 PRINT TAB(4);"-----------{4 SPACES}----
     --": :
1540 PRINT TAB(8);"1{11 SPACES}68.3"
1550 PRINT TAB(8);"2{11 SPACES}95.5"
1560 PRINT TAB(8);"3{11 SPACES}99.7"
1570 PRINT TAB(8);"4{11 SPACES}99.9": : :
1580 GOSUB 2160
1590 PRINT TAB(3);"THE MEDIAN IS THE VALUE A
     T": :
1600 PRINT "THE MID-POINT OF YOUR DATA.": :
     :
1610 PRINT TAB(3);"THE RANGE IS THE DIF-": :
1620 PRINT "FERENCE BETWEEN YOUR LOWEST": :
1630 PRINT "DATA VALUE AND THE HIGHEST.": :
1640 PRINT "IT IS A QUICK-AND-DIRTY": :
1650 PRINT "ESTIMATE OF THE SPREAD.": :
1660 PRINT "STANDARD DEVIATION IS MORE": :
1670 PRINT "RELIABLE, HOWEVER.": : : :
1680 PRINT TAB(3);"PRESS ANY KEY TO START"
1690 GOSUB 2170
1700 RETURN
1710 REM DISPLAY CORRECTION OPTION
1720 GOSUB 2170
1730 IF (K<>67)*(K<>78)*(K<>81)THEN 1720
```

```
1740 FL=0
1750 IF K<>78 THEN 1780
1760 FL=1
1770 GOTO 1980
1780 IF K=81 THEN 770
1790 REM ERROR CORRECTION SUBR
1800 PRINT "REMEMBER INCORRECT SAMPLE #": :
1810 PRINT TAB(11);"(Y/N) ?": : : : : : : :
     : :
1820 GOSUB 2170
1830 IF K=78 THEN 1980
1840 INPUT "WHAT IS THE SAMPLE # ? ":EN$
1850 EN=VAL(EN$)
1860 IF (EN>N)+(EN<1)+(EN<>INT(EN))THEN 1840
1870 PRINT : :
1880 PRINT "SAMPLE";EN;"{3 SPACES}";"VALUE="
     ;SA(EN)
1890 PRINT : :
1900 PRINT "ENTER YOUR NEW VALUE : "
1910 INPUT SA(EN)
1920 PRINT : : : : : :
1930 PRINT TAB(3);"ANY MORE CHANGES (Y/N)?":
      : : : :
1940 GOSUB 2170
1950 CALL CLEAR
1960 IF K=78 THEN 770
1970 GOTO 1800
1980 IF FL=1 THEN 2020
1990 PRINT "THESE ARE THE FIRST TEN": :
2000 L=1
2010 GOTO 2040
2020 CALL CLEAR
2030 PRINT "THESE ARE THE NEXT TEN": :
2040 PRINT "VALUES.": : :
2050 PRINT TAB(5);"ENTRY";TAB(15);"VALUE": :
2060 FF=0
2070 FOR L=L TO L+9
2080 FF=FF+1
2090 IF L>300 THEN 770
2100 PRINT TAB(5);L;TAB(15);SA(L)
2110 NEXT L
2120 PRINT : :
2130 PRINT "C=CHANGE DATA{3 SPACES}N=NEXT TA
     BLE": :
```

```
2140 PRINT TAB(12);"Q=QUIT"
2150 GOTO 1720
2160 PRINT TAB(3);"PRESS ANY KEY FOR MORE";
2170 CALL KEY(0,K,S)
2180 IF S=0 THEN 2170
2190 CALL CLEAR
2200 RETURN
```

TIcalc

Raymond J. Herold

Spreadsheets are exceptionally useful tools: for calculating, modeling, or predicting. This program creates a spreadsheet of ample size (26 rows by 14 columns). For the TI-99/4A with Extended BASIC.

"TIcalc" is an electronic spreadsheet program for the TI-99/4A computer with Extended BASIC. Electronic spreadsheets, useful and popular programs, allow the user to answer a multitude of "what if" questions in areas such as budgeting, sales projections, cost estimating, scheduling, and more.

Spreadsheets allow you to enter a set of values and calculation rules for a given application, such as budgeting. The program will then calculate the projections, estimates, totals, or whatever, based on the calculation rules. Changing one or more of the original values results in a complete recalculation of the figures. The special utility of spreadsheet programs lies in their ability to do, in a few seconds, what a human—with pencil, paper, and calculator—would need hours, or even days, to do.

Program Requirements

Before explaining how to use TIcalc, let's establish the ground rules for the program. First, it requires at least a 16K TI-99/4A with Extended BASIC. Although the TIcalc spreadsheet is 26 rows by 14 columns, with 16K of memory built into the TI console, you are limited to roughly 150 "slots." For example, you could have a spreadsheet that is 12 × 12, 15 × 10, 20 × 7, or 10 × 14. You will find this adequate for almost all applications. Those of you who have the 32K memory expansion can use the complete 26 × 14 spreadsheet. When using the program, you should leave the ALPHA LOCK key depressed.

Spreadsheets can be saved and loaded from tape. If you have a disk drive, you can change the OPEN statements in lines 1950 and 2000 accordingly. The use of a printer is optional, but the program does provide the option of making a printout of your results.

The TIcalc spreadsheet is 26 rows by 14 columns (see Figure 1). The rows of the spreadsheet are defined by the letters A–Z. The columns are defined by A–N. Note that any slot in the spreadsheet is referred to by row and column. For example, slot CD would be the entry at row 3, column 4; AF would be row 1, column 6. It's important that you keep this sequence in mind.

The TI-99/4A is not capable of displaying the entire 26 × 14 array. What will appear on your screen is a 10 × 3 "window" on the spreadsheet. Just as looking into different windows of a house shows different things, the computer's window shows different "views" of the spreadsheet, depending on where the window is positioned. A window's position is defined by its top-left slot. Looking again at Figure 1, notice that the shaded area marked A is the 10 by 3 spreadsheet window at AA (remember, row and column). The shaded area marked B is the window at IH. By moving the window, the entire 364-slot spreadsheet is accessible 30 slots (a window) at a time.

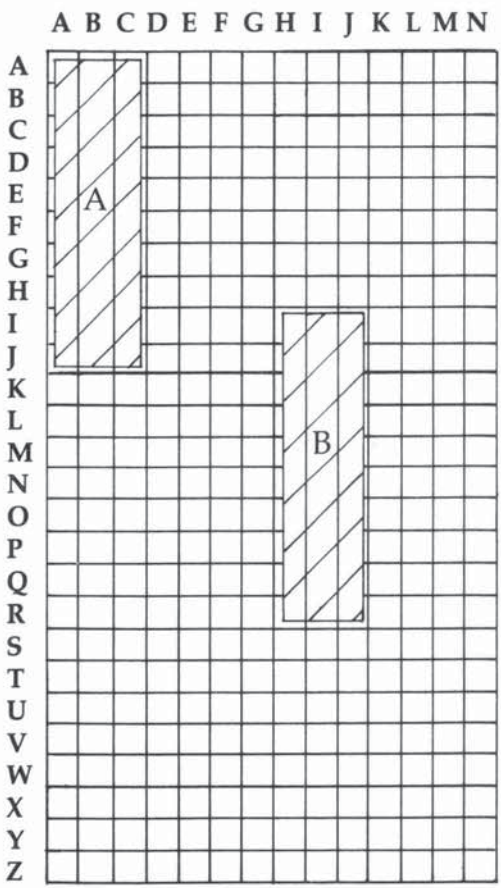
The best way to demonstrate TIcalc is by example. You should spend a few minutes getting acquainted with the command summary shown in Table 1. Also, you might want to examine the list of major program variables shown in Table 2. The following paragraph will detail a somewhat simplistic scenario for our demonstration.

Starting a Business

We are starting a small manufacturing business and want to estimate our net profit or loss for the first four months. We are anticipating sales of $2,700 the first month and a 10 percent growth rate for each succeeding month. Space is being leased for $800 a month, and there are two employees making a total of $1,200 a month. Cost for materials is based on sales and is expected to be 30 percent, while utilities are expected to run at roughly 5 percent of sales.

When the program begins, it displays the window with a HOME position of AA. That is, it is displaying rows A through J and columns A, B, and C. The COMMAND —> prompt is displayed, and the program is awaiting your reply. Since the first thing we want to do is enter spreadsheet data, reply INSERT. This places the cursor (actually two sprites at line 860) at the top-left slot in the window, in this case AA. The prompt

Figure 1: Windows on the Spreadsheet



asks for an INSERT COMMAND?. Figure 2 shows the data we plan to enter (refer to it as we go along). As you can see, there isn't any data for AA, so we press X(↓) to move the cursor down to CA.

At this point we want to place the label SALES in the CA slot, so we press L. The prompt then asks us what the label is and we type SALES. When we press ENTER, the label is placed in CA. We then press X(↓) again to get to DA and enter the label RENT. Continue this for all the labels in column A. Then use the arrow keys (really E, S, D, and X) to place the cursor at AB, where you enter the label JANUARY. Then move the cursor down to CB.

This slot is to be the amount of our first month's sales, so press N for numeric value. The prompt asks for the number; respond 2700 and press ENTER. Do the same for RENT at DB and SALARY at EB. At FB we come to the first calculation, so press C. Remember that material costs are expected to be 30 percent of monthly sales. Therefore, we need to multiply SALES by .30. The .30 will have to be stored as a value in a "workfield" outside the main body of the spreadsheet. We will arbitrarily make this BJ and make a note to ourselves to add the value after finishing the main portion of the spreadsheet. So, the calculation becomes JANUARY SALES (CB)*.30(BJ) or CB*BJ. Refer to Table 3 for examples of valid calculations. An error detection routine enforces valid syntax.

We then position the cursor at GB, which is January utility costs. This is similar to material costs, and we make a note to store the 5 percent figure at CJ. Press C and then enter CB*CJ. The cursor is then positioned at IB, which is the slot for total January expenses. This is again a calculation, so press C. Enter the calculation command SUMCOLDG, which means sum this column starting at row D (RENT) and ending with row G (UTILITY) and place the result in this slot. The cursor is then placed at JB, which is the NET PROFIT/LOSS for January. This is simply SALES (CB) minus TOTAL EXPENSES(IB) or CB−IB.

Next, position the cursor at AC and enter the February label. When you position the cursor at February SALES, you'll see that you no longer have a number, but rather a calculation. Sales are assumed to be 10 percent greater than each previous month, so make a note to store 1.10 at AJ and enter the calculation CB*AJ, which is January SALES*1.10. The remainder of the column is entered in a manner similar to the entries for January, adjusting for the proper row/column designators.

At this point, all the slots for the window being displayed have been entered, so you'll need to move the window. First press Q to exit from INSERT mode. When the command prompt is displayed, enter HOME and press ENTER. When asked for row and column, enter AD. The window will be moved to view rows A through J, columns D, E, and F. Type INSERT and press ENTER to get back into INSERT mode. The columns for March and April can now be entered as were the columns for January and February. Column F, the total columns of the calculation, is a little different. The SUMROWBE

Figure 2. Example Spreadsheet

	A	B	C	D	E	F	G	H	I	J
A		JANUARY	FEBRUARY	MARCH	APRIL	-TOTAL-	% SALES			1.10
B										.30
C	SALES	2700	CB*AJ	CC*AJ	CD*AJ	SUMROWBE				.05
D	RENT	800	800	800	800	SUMROWBE	CF%DF			
E	SALARY	1200	1200	1200	1200	SUMROWBE	CF%EF			
F	MATERIAL	CB*BJ	CC*BJ	CD*BJ	CE*BJ	SUMROWBE	CF%FF			
G	UTILITY	CB*CJ	CC*CJ	CD*CJ	CE*CJ	SUMROWBE	CF%GF			
H										
I	TOT EXP	SUMCOLDG	SUMCOLDG	SUMCOLDG	SUMCOLDG	SUMCOLDG	CF%IF			
J	NET +/−	CB-IB	CC-IC	CD-ID	CE-IE	CF-IF	CF%JF			

command tells TIcalc to total the row starting at column B (January) and ending at column E (April), and place the result in the current slot.

We have again filled the window being displayed, so press Q to exit INSERT mode. Typing the HOME command and then AG gives us slot AG in the top left of the screen. Type INSERT again and enter the calculation rules to give each expense, the total expense, and net as a percent of sales. Finally, exit (Q), HOME on AJ, INSERT, and enter the workfield values for AJ, BJ, and CJ. Type Q to get back to command mode. At this point, you've completed your working copy (MODE1) of the spreadsheet.

Procedures

Now you can use the CALC command to calculate the result of the working copy. The calculation will take anywhere from a few seconds to a few minutes, depending on the size of the working copy and the number of calculations. When the calculation is complete, the program will automatically go into MODE2 and set the HOME row and column to AA. You can then view the results by moving the window, using the HOME command. Figure 3 shows the results from the sample. If you want to see the calculation that gave a particular result, you can type MODE1 to see the original working copy as shown in Figure 2. Typing MODE2 will return you to the "result copy." This is particularly useful in finding errors.

Figure 3. Printout of Example Worksheet Results

	JANUARY	FEBRUARY	MARCH	APRIL	-TOTAL-	% SALES
SALES	2700	2970	3267	3593.7	12530.7	
RENT	800	800	800	800	3200	25.53
SALARY	1200	1200	1200	1200	4800	38.3
MATERIAL	810	891	980.1	1078.11	3759.21	30
UTILITY	135	148.5	163.35	179.68	626.53	4.99
TOT EXP	2945	3039.5	3143.45	3257.79	12385.74	98.84
NET +/−	−245	−69.5	123.55	335.91	144.96	1.15

The Daisychain Effect

Anytime TIcalc encounters a calculation it cannot complete when in its calculation mode, it will fill the current slot with all *. This kind of error is usually caused by one of two conditions. The first is when a calculation refers to a slot which is not defined as a number or calculation. For example, if our sample had a calculation CB*AH, the result would be an error because slot AH has no value. If a slot contained a label, the same error would occur. The second type of error occurs when a current calculation points to a slot that contains a calculation which previously contained an error. In this case, the current calculation is correct, but the calculation it refers to must be corrected. This type of error tends to have a daisychain effect.

All calculations are taken to a maximum of two decimal places. There is no provision for rounding. Also, all calculations are carried out in row/column sequence. That is, AA is processed first, then AB, AC, AD, then BA, BB, BC, and BD. This is very important to understand since errors will be generated if you reference a slot which has not yet been processed. For example, if slot AC contains the calculation AB*BC, an error will occur since BC has not yet been processed. Thus, the selection of AJ, BJ, and CJ for workfields is not as arbitrary as it first appears.

Printing and Saving

You can print the result of the calculation by using the PRINT command. It will print all rows for the beginning and ending columns you specify. Figure 3 was produced by PRINTing for columns A through G. You may have to adjust the OPEN command at line 2070 for your particular printer.

You may save a spreadsheet or load one from tape. Note that if you load a spreadsheet from tape, only the working copy is loaded. You will have to issue the CALC command to compute a result copy.

The usefulness of TIcalc may be demonstrated by using our sample. If, after the first month, there were any deviations from the assumptions made at the outset, or if you wanted to see what a higher or lower sales figure would do, you would merely need to change the desired variable(s) and recalculate.

Table 1. TIcalc Command Summary

Command	Action
HOME	Aligns the TIcalc window to the desired row/column.
INSERT	Places TIcalc in INSERT mode; defaults to MODE1 (see subcommands below).
MODE1	Displays the working copy; automatic for INSERT.
MODE2	Displays the result copy; automatic after CALC command.

CALC	Calculates the results for the values and calculations in the working copy; invokes MODE2 at completion.
LOAD	Load a spreadsheet from tape.
SAVE	Save a spreadsheet to tape.
PRINT	Print spreadsheet.

INSERT Subcommands

Subcommands	Action
←(S)	Move cursor left.
→(D)	Move cursor right.
↑(E)	Move cursor up.
↓(X)	Move cursor down.
L	Indicates a label is to be placed in the current cursor position.
N	Indicates a numeric value is to be placed in the current cursor position.
C	Indicates a calculation is to be placed in the current cursor position.
Q	Quit; return to command mode.

Table 2. Major Program Variables

Variable	Use
A$(r,c)	Working copy array
B$(r,c)	Result copy array
COMM$	Command entered
ROW	Row shown at top left of window
COL	Column shown at top left of window
RC$	A through Z values
MODE	MODE1 or MODE2 indicator
LOC$	Row/column desired by HOME command
R	Loop control—row
C	Loop control—column
X	Row DISPLAY AT position
Y	Column DISPLAY AT position
SR	Cursor row position
SC	Cursor column position
L$	Label entered
N$	Number entered
C$	Calculation entered
RM	Highest row number used
CM	Highest column number used
RLIM	Row limit for display window
CLIM	Column limit for display window

Table 3. Valid TIcalc Calculations

OPERATORS	+,−,*,/,%
SUMROWXY	Where X is the beginning column and Y is the ending column
SUMCOLXY	Where X is the beginning row and Y is the ending row

Examples

AB*CG
AL−AI
EF+AH
BC/CA
AB+CB*BC
AB+CB+CA Processed left to right
CB/AB−CH
SUMROWCF
SUMCOLAH

TIcalc

```
100 DIM A$(26,14),B$(26,14)
110 CALL CHAR(96,"FFFFFFFFFFFFFFFF"):: CALL
    COLOR(9,13,1)
120 ROW=1 :: COL=1 :: RLIM=10 :: CLIM=3
130 RC$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
140 CALL CHAR(104,"FFFFE0E0E0E0FFFF"):: CALL
     CHAR(105,"FFFF07070707FFFF"):: CALL COL
    OR(10,7,1)
150 CALL CLEAR :: CALL SCREEN(9)
160 DISPLAY AT(5,7):"E A S Y C A L C" :: DIS
    PLAY AT(9,9):"ELECTRONIC" :: DISPLAY AT(
    11,9):"SPREADSHEET"
170 FOR DELAY=1 TO 2000 :: NEXT DELAY
180 CALL CLEAR :: CALL SCREEN(8)
190 CALL HCHAR(4,3,96,29)
200 CALL VCHAR(5,4,96,19)
210 CALL VCHAR(5,13,96,19)
220 CALL VCHAR(5,22,96,19)
230 GOSUB 340 :: MODE=1
240 DISPLAY AT(1,1):"COMMAND: --->" :: ACCEP
    T AT(1,15)SIZE(6)BEEP:COMM$
250 IF SEG$(COMM$,1,4)="HOME" THEN 410
260 IF SEG$(COMM$,1,6)="INSERT" THEN 720
270 IF SEG$(COMM$,1,5)="MODE1" THEN GOSUB 52
    0 :: GOTO 240
280 IF SEG$(COMM$,1,5)="MODE2" THEN GOSUB 62
    0 :: GOTO 240
```

```
290 IF SEG$(COMM$,1,4)="CALC" THEN 1370
300 IF SEG$(COMM$,1,4)="SAVE" THEN 1950
310 IF SEG$(COMM$,1,4)="LOAD" THEN 2000
320 IF SEG$(COMM$,1,5)="PRINT" THEN 2050
330 GOTO 240
340 FOR LOOP=2 TO 20 STEP 2
350 DISPLAY AT(3+LOOP,1):SEG$(RC$,ROW+(LOOP/
    2)-1,1);
360 NEXT LOOP
370 FOR LOOP=6 TO 26 STEP 9
380 DISPLAY AT(3,LOOP):SEG$(RC$,COL-1+(LOOP/
    8),1)
390 NEXT LOOP
400 RETURN
410 DISPLAY AT(1,1):"ROW/COL ---> .." :: ACC
    EPT AT(1,14)VALIDATE(RC$)SIZE(-2)BEEP:LO
    C$
420 IF SEG$(LOC$,2,1)="." THEN 410
430 IF SEG$(LOC$,2,1)>"N" THEN 410
440 ROW=(ASC(SEG$(LOC$,1,1))-64):: IF ROW>17
     THEN ROW=17
450 RLIM=ROW+9
460 COL=(ASC(SEG$(LOC$,2,1))-64):: IF COL>12
     THEN COL=12
470 CLIM=COL+2
480 GOSUB 340
490 IF MODE=1 THEN GOSUB 520
500 IF MODE=2 THEN GOSUB 620
510 GOTO 240
520 X=5 :: FOR R=ROW TO RLIM
530 Y=3
540 FOR C=COL TO CLIM
550 DISPLAY AT(X,Y):"{8 SPACES}";
560 DISPLAY AT(X,Y):SEG$(A$(R,C),3,8);
570 Y=Y+9
580 NEXT C
590 X=X+2
600 NEXT R
610 MODE=1 :: RETURN
620 X=5 :: FOR R=ROW TO RLIM
630 Y=3
640 FOR C=COL TO CLIM
650 DISPLAY AT(X,Y):"{8 SPACES}";
660 DISPLAY AT(X,Y):B$(R,C);
670 Y=Y+9
680 NEXT C
690 X=X+2
700 NEXT R
710 MODE=2 :: RETURN
```

```
720 IF MODE=2 THEN GOSUB 520
730 SR=32 :: SC=32 :: R=ROW :: C=COL :: X=5
    :: Y=3
740 GOSUB 860
750 DISPLAY AT(1,1):"INSERT COMMAND?" :: CAL
    L SOUND(200,1100,4)
760 CALL KEY(3,KEY,STATUS):: IF STATUS=0 THE
    N 760
770 IF KEY=76 THEN 880
780 IF KEY=78 THEN 930
790 IF KEY=67 THEN 1030
800 IF KEY=81 THEN CALL DELSPRITE(ALL):: GOT
    O 240
810 IF KEY=83 THEN 1250
820 IF KEY=68 THEN 1280
830 IF KEY=69 THEN 1310
840 IF KEY=88 THEN 1340
850 GOTO 750
860 CALL SPRITE(#1,104,7,SR,SC,0,0,#2,105,7,
    SR,SC+56,0,0)
870 RETURN
880 DISPLAY AT(1,1):"LABEL: --->" :: ACCEPT
    AT(1,13)SIZE(8)BEEP:L$
890 DISPLAY AT(X,Y):L$;
900 A$(R,C)="L:"&L$
910 RM=MAX(RM,R):: CM=MAX(CM,C)
920 GOTO 740
930 DISPLAY AT(1,1):"NUMBER: --->" :: ACCEPT
     AT(1,14)SIZE(8)VALIDATE("0123456789.-+"
    )BEEP:N$
940 W$="" :: W=0
950 FOR Z=8 TO 1 STEP -1
960 IF SEG$(N$,Z,1)<>"" THEN W$=SEG$(N$,Z,1)
    &W$ ELSE W=W+1
970 NEXT Z
980 W$=RPT$(" ",W)&W$
990 DISPLAY AT(X,Y):W$;
1000 A$(R,C)="N:"&W$
1010 RM=MAX(RM,R):: CM=MAX(CM,C)
1020 GOTO 740
1030 DISPLAY AT(1,1):"CALCULATION: --->" ::
     ACCEPT AT(1,19)SIZE(8)BEEP:C$
1040 IF SEG$(C$,1,6)="SUMROW" THEN 1190
1050 IF SEG$(C$,1,6)="SUMCOL" THEN 1190
1060 AA$=SEG$(C$,3,1)
1070 IF AA$="+" OR AA$="-" OR AA$="*" OR AA$
     ="/" OR AA$="%" THEN 1090
```

```
1080 DISPLAY AT(1,1):"*** ERROR ***" :: FOR
     DELAY=1 TO 1200 :: NEXT DELAY :: GOTO 1
     030
1090 AA$=SEG$(C$,1,1):: IF AA$<"A" OR AA$>"Z
     " THEN 1080
1100 AA$=SEG$(C$,2,1):: IF AA$<"A" OR AA$>"N
     " THEN 1080
1110 AA$=SEG$(C$,4,1):: IF AA$<"A" OR AA$>"Z
     " THEN 1080
1120 AA$=SEG$(C$,5,1):: IF AA$<"A" OR AA$>"N
     " THEN 1080
1130 AA$=SEG$(C$,6,1)
1140 IF (AA$="" OR AA$=" ")AND(SEG$(C$,7,2)<
     >"" AND SEG$(C$,7,2)<>"  ")THEN 1080
1150 IF AA$="" OR AA$=" " THEN 1210
1160 IF AA$="+" OR AA$="-" OR AA$="*" OR AA$
     ="/" OR AA$="%" THEN 1170 ELSE 1080
1170 AA$=SEG$(C$,7,1):: IF AA$<"A" OR AA$>"Z
     " THEN 1080
1180 AA$=SEG$(C$,8,1):: IF AA$<"A" OR AA$>"N
     " THEN 1080
1190 IF SEG$(C$,4,3)="ROW" THEN IF SEG$(C$,7
     ,1)<"A" OR SEG$(C$,7,1)>"N" OR SEG$(C$,
     8,1)<"A" OR SEG$(C$,8,1)>"N" THEN 1080
1200 IF SEG$(C$,4,3)="COL" THEN IF SEG$(C$,7
     ,1)<"A" OR SEG$(C$,7,1)>"Z" OR SEG$(C$,
     8,1)<"A" OR SEG$(C$,8,1)>"Z" THEN 1080
1210 DISPLAY AT(X,Y):C$;
1220 A$(R,C)="C:"&C$
1230 RM=MAX(RM,R):: CM=MAX(CM,C)
1240 GOTO 740
1250 IF SC-72<32 OR C-1<1 THEN 750
1260 SC=SC-72 :: C=C-1 :: Y=Y-9
1270 GOTO 740
1280 IF SC+72>176 OR C+1>26 THEN 750
1290 SC=SC+72 :: C=C+1 :: Y=Y+9
1300 GOTO 740
1310 IF SR-16<32 OR R-1<1 THEN 750
1320 SR=SR-16 :: R=R-1 :: X=X-2
1330 GOTO 740
1340 IF SR+16>176 OR R+1>26 THEN 750
1350 SR=SR+16 :: R=R+1 :: X=X+2
1360 GOTO 740
1370 DISPLAY AT(1,1):"CALCULATION IN PROGRES
     S"
1380 FOR R=1 TO RM
1390 FOR C=1 TO CM
```

```
1400 IF SEG$(A$(R,C),1,2)="L:" OR SEG$(A$(R,
     C),1,2)="N:" THEN B$(R,C)=SEG$(A$(R,C),
     3,8)
1410 IF SEG$(A$(R,C),1,2)="C:" THEN GOSUB 14
     70
1420 NEXT C
1430 DISPLAY AT(1,25):R
1440 NEXT R
1450 MODE=2 :: LOC$="AA"
1460 GOTO 440
1470 IF SEG$(A$(R,C),3,3)="SUM" THEN 1770
1480 R1=ASC(SEG$(A$(R,C),3,1))-64 :: C1=ASC(
     SEG$(A$(R,C),4,1))-64
1490 R2=ASC(SEG$(A$(R,C),6,1))-64 :: C2=ASC(
     SEG$(A$(R,C),7,1))-64
1500 IF SEG$(A$(R,C),9,1)>="A" THEN R3=ASC(S
     EG$(A$(R,C),9,1))-64 :: C3=ASC(SEG$(A$(
     R,C),10,1))-64
1510 IF SEG$(A$(R1,C1),1,2)<>"N:" AND SEG$(A
     $(R1,C1),1,2)<>"C:" THEN B$(R,C)="*****
     ***" :: RETURN
1520 IF SEG$(A$(R2,C2),1,2)<>"N:" AND SEG$(A
     $(R2,C2),1,2)<>"C:" THEN B$(R,C)="*****
     ***" :: RETURN
1530 IF SEG$(A$(R,C),9,1)<"A" THEN 1550
1540 IF SEG$(A$(R3,C3),1,2)<>"N:" AND SEG$(A
     $(R3,C3),1,2)<>"C:" THEN B$(R,C)="*****
     ***" :: RETURN
1550 ON ERROR 1920
1560 W1=VAL(B$(R1,C1)):: W2=VAL(B$(R2,C2))::
      IF SEG$(A$(R,C),9,1)>"A" THEN W3=VAL(B
     $(R3,C3))
1570 W4=0 :: F$=(A$(R,C))
1580 IF SEG$(F$,5,1)="+" THEN W4=W1+W2
1590 IF SEG$(F$,5,1)="-" THEN W4=W1-W2
1600 IF SEG$(F$,5,1)="*" THEN W4=W1*W2
1610 IF SEG$(F$,5,1)="/" THEN W4=W1/W2
1620 IF SEG$(F$,5,1)="%" THEN W4=W2/W1*100
1630 IF SEG$(F$,8,1)="+" THEN W4=W4+W3
1640 IF SEG$(F$,8,1)="-" THEN W4=W4-W3
1650 IF SEG$(F$,8,1)="*" THEN W4=W4*W3
1660 IF SEG$(F$,8,1)="/" THEN W4=W4/W3
1670 IF SEG$(F$,8,1)="%" THEN W4=W3/W4*100
1680 IF INT(W4)<>W4 THEN W4=INT(W4*100)/100
1690 R$=STR$(W4):: W$="" :: W=0
1700 FOR Z=8 TO 1 STEP -1
1710 IF SEG$(R$,Z,1)<>"" THEN W$=SEG$(R$,Z,1
     )&W$ ELSE W=W+1
1720 NEXT Z
```

```
1730 W$=RPT$(" ",W)&W$
1740 B$(R,C)=W$
1750 ON ERROR STOP
1760 RETURN
1770 IF SEG$(A$(R,C),6,3)="ROW" THEN 1800
1780 IF SEG$(A$(R,C),6,3)="COL" THEN 1860
1790 RETURN
1800 W4=0 :: ON ERROR 1920
1810 V=ASC(SEG$(A$(R,C),9,1))-64 :: W=ASC(SE
     G$(A$(R,C),10,1))-64
1820 FOR Z=V TO W
1830 IF SEG$(A$(R,Z),1,2)="N:" OR SEG$(A$(R,
     Z),1,2)="C:" THEN W4=W4+VAL(B$(R,Z))
1840 NEXT Z
1850 GOTO 1680
1860 W4=0 :: ON ERROR 1920
1870 V=ASC(SEG$(A$(R,C),9,1))-64 :: W=ASC(SE
     G$(A$(R,C),10,1))-64
1880 FOR Z=V TO W
1890 IF SEG$(A$(Z,C),1,2)="N:" OR SEG$(A$(Z,
     C),1,2)="C:" THEN W4=W4+VAL(B$(Z,C))
1900 NEXT Z
1910 GOTO 1680
1920 B$(R,C)="********"
1930 RETURN 1940
1940 RETURN
1950 CALL CLEAR :: OPEN #1:"CS1",OUTPUT,INTE
     RNAL,FIXED 192
1960 PRINT #1:CM;RM
1970 FOR Z=1 TO RM
1980 PRINT #1:A$(Z,1);A$(Z,2);A$(Z,3);A$(Z,4
     );A$(Z,5);A$(Z,6);A$(Z,7);A$(Z,8);A$(Z,
     9);A$(Z,10);A$(Z,11);A$(Z,12);A$(Z,13);
     A$(Z,14)
1990 NEXT Z :: CLOSE #1 :: GOTO 180
2000 CALL CLEAR :: OPEN #1:"CS1",INPUT ,INTE
     RNAL,FIXED 192
2010 INPUT #1:CM,RM
2020 FOR Z=1 TO RM
2030 INPUT #1:A$(Z,1),A$(Z,2),A$(Z,3),A$(Z,4
     ),A$(Z,5),A$(Z,6),A$(Z,7),A$(Z,8),A$(Z,
     9),A$(Z,10),A$(Z,11),A$(Z,12),A$(Z,13),
     A$(Z,14)
2040 NEXT Z :: CLOSE #1 :: GOTO 180
2050 DISPLAY AT(1,1):"BEGIN/END COLUMN .." :
     : ACCEPT AT(1,18)SIZE(-2)BEEP:C$
2060 IF SEG$(C$,1,1)<"A" OR SEG$(C$,1,1)>"Z"
      OR SEG$(C$,2,1)<"A" OR SEG$(C$,2,1)>"N
     " THEN 2050
```

```
2070 OPEN #2:"RS232",OUTPUT,DISPLAY
2080 I=ASC(SEG$(C$,1,1))-64 :: J=ASC(SEG$(C$
     ,2,1))-64
2090 FOR L=1 TO RM
2100 FOR M=I TO J
2110 P$=RPT$(" ",10-LEN(B$(L,M)))&B$(L,M)
2120 PRINT #2:P$;
2130 NEXT M
2140 PRINT #2:" " :: PRINT #2:" "
2150 NEXT L
2160 CLOSE #2
2170 GOTO 240
```

Financial Interests

Doug Hapeman

Interest rates can be a disappointment or a pleasant surprise if you are paying interest on a loan or earning interest on your savings. "Financial Interests" can help you make sense of such mysteries as amortization and compound interest before you sign on that bottom line. You'll also learn a few things about finance in general. For the TI-99/4A, with Extended BASIC, and 16K memory.

"Financial Interests" allows you to calculate both the value of investments and the cost of borrowing.

You may be considering a savings investment fund. This program helps you examine savings and annuities with various compound periods and rates, letting you see the future value of your money. Or, if you're considering a loan, you can weigh the options of various amounts, rates, and amortization periods, and then choose the best alternative.

Simple and Compound Interest

To understand finance, you must grasp the idea of *interest.* There are two types: simple (or fixed) interest and compound interest.

For instance, if you borrowed $1000 and agreed to repay it with 12 percent interest, you would repay the principal amount ($1000) plus the 12 percent ($120)—regardless of the length of the repayment period. However, if you agreed to repay the loan at 12 percent *per annum,* the loan has *compound interest.* Now, 12 percent interest will be added onto the outstanding debt each year during the repayment period.

The more frequent the compounding, the more costly to the borrower. Today, most banks compound the interest monthly on personal loans for cars, household items, vacations, etc.

For once, wouldn't it be nice to sit down with the loan officer in the bank and know what your options are before you sign on the dotted line? One of the frustrating things about negotiating a loan is having to make a decision when you don't fully understand all the options.

For instance, when you're buying a new car, contrasting the differences between a 36-month and a 48-month repayment period can be helpful. How will the different periods affect the size of the monthly payment? How much more interest is paid in a 48-month amortization than in a 36-month period? What portion of each monthly payment is for interest? For principal?

This program, Financial Interests, will help you examine all those options and will even print them out on paper for you. The calculations used in the program are based on the assumption that the interest is compounded monthly and that payments will be made monthly.

The Dead Pledge

The word *mortgage* comes from two French words, *mort* (dead) and *gage* (pledge). The pledge becomes dead when the loan is paid off. To *amortize* means to deaden. To amortize a mortgage or a loan is to extinguish it by means of a "sinking fund"—a series of payments over a period of time which will reduce the debt to zero.

By the way, a mortgage deed is sometimes called an *indenture*. The word simply means an agreement between two or more parties, but its etymology is pretty interesting.

Many years ago, (before carbon paper and photocopiers) such an agreement would be penned in two original copies. The copies would be placed evenly, one on top of the other. A wavy line, or *indentation*, would literally be cut along one side of the copies. Each party would then receive one of the papers. When the two were later placed together, the wavy cutting would match. Thus, authenticity was established. The indentation matched.

Loans Vs. Mortgages

Everyone knows the difference between a loan and a mortgage, right? They're the same thing except you amortize a mortgage over a longer period, such as a 20- or 30-year period? No. Most personal loans compound the interest monthly, but the Federal Interest Act (in Canada) requires that, for a mortgage, interest can only be "calculated half-yearly, or yearly, not in advance." Therefore, the primary difference between a mortgage and a loan is that mortgage interest cannot be compounded as frequently, which means lower payments. Of course, there are other differences: Mortgages usually offer much lower interest rates, they have stiff penalties for paying against the principal in advance, and they require the involvement and expense of a lawyer.

Financial Interests calculates mortgage payments on the assumption that the interest is compounded semiannually, not in advance. If you compare the figures from Financial Interests with the figures from a mortgage interest guidebook, you may find the figures vary slightly. This is because the 13-digit accuracy of the 99/4A gives a more exact calculation than most guidebooks.

No More Than a Million

When either the Loans analysis or the Mortgage analysis is chosen, the program first asks the size of the loan you are considering. The program will accept amounts up to, but not including, one million dollars. If you are considering more than that, adjust the program to accept larger amounts by changing the SIZE variable of the ACCEPT statement in line 420. Second, you are asked the annual interest rate, and third, the length of the loan in months. The information is then calculated and the screen displays the monthly payment needed to pay off the principal during the life of the loan.

At this point, you are given two options: a month-by-month analysis of the loan, or return to the main index. When you choose the analysis, you are asked whether you would like the amortization schedule printed. If yes, the printer configuration is requested. The printout shows the current state of the loan after each payment. The information includes the month number, the monthly payment, the monthly interest and principal, the remaining balance, total interest to date, and total payments to date.

When the printer is bypassed, the monitor screen displays one month at a time, and you can proceed month by month by pressing any key other than M or T. Pressing M permits you to jump ahead to any month you select, and pressing T jumps to the final breakdown totals following the last payment.

Savings Analysis

The Savings analysis lets you examine a combination of two investment procedures: investments (the future value of a one-time deposit) and annuities (the future value of regular deposits).

The Savings option first asks for the present amount in your savings account, then the rate of interest and the number of compound periods per year. Following this, you are asked whether you wish to make regular deposits, and if so, how often and how much. From there the calculations are performed and displayed, showing the beginning principal, the total deposits, the accumulated balance, and the total interest. Analysis is displayed on a yearly basis with the option of returning to the main menu at any time.

The two procedures, annuities and investments, can be analyzed in conjunction with each other, or individually. If you wish to examine just the future growth of a one-time investment, press N (No) in response to the question "Make regular deposits?" Calculations will then be made based solely on the future growth of a single deposit over a designated period of time. The growth of this fund depends upon the interest paid. The interest is compounded each period. This is interest earned on interest.

If you wish to analyze only an annuity, enter 0 in response to "Present amount in savings:", and then continue with the remaining information. This will give you calculations for the future growth of a regular contribution to an annuity fund, that is, the regular periodic investment, plus interest earned on the interest and on the continuing investment.

These investment factors are all based on the assumption that no funds will be withdrawn throughout the investment period.

For Formula Buffs

In case you want to know how it is done or would like to work it out the hard way, here are the formulae:

Compound Savings (Investment)

S = Amt*(1+I)^N

Amt = Amount deposited
S = The future value of amount deposited
I = Interest rate per period
N = Number of compounding periods

Annuities

S = $\frac{\text{Amt*(1+I)\^N)}-1}{\text{I}}$

Amt = Amount deposited per period
S = The future value of amount deposited per period
I = Interest rate per period
N = Number of compounding periods

Loan Payments

FR = (1+R/1200)−1

S = $\frac{\text{Amt*FR}}{1-(1/1+\text{FR})\hat{}\text{N})}$

FR = Loan amortization factor
R = Annual interest rate
S = The monthly payment
Amt = Amount to be borrowed
N = Length of loans in months

Mortgage Payments

FR = ((1+R/200) (1/6))−1

S = $\frac{\text{Amt*FR}}{1-(1/((1+\text{FR})\hat{}\text{N}))}$

FR = Mortgage amortization factor
R = Annual interest rate
S = The monthly payment
Amt = Amount to be borrowed
N = Length of mortgage in months

Program Outline

100–300 Initialization and title screen
310–340 Main menu
350–360 Finish session
370–430 Get loan and mortgage information
440–510 Calculate and display monthly payment
520–560 Month-by-month analysis
570–630 Analysis calculations
640–680 Print amortization schedule
690–790 Display calculations
800–900 Get savings information
910–940 Savings analysis
950–970 Analysis calculations
980–1020 Display calculations

Main Program Variables

Title Screen Variables

V = Vertical sprite motion
H = Horizontal sprite motion
R = Dot-row sprite location
C = Dot-column sprite location
RR = Row-character position
CC = Column-character position
J = Flag

Loan and Mortgage Calculation Variables

AMT = Beginning principal
R = Annual interest rate
M = Months in length of loan
FR = Working factor for mortgage and loan amortization
PA = Monthly payment
TP = Total payments
IN = Interest
TI = Total interest
BA = Remaining balance

Savings Calculation Variables

AMT = Amount in savings
R = Annual interest rate
C = Number of compound periods
D = Amount of deposits
ND = Number of deposits
Y = Number of years in analysis
CP = Interest rate per compound period
B = Future value of amount in savings
MA = Working variable for annuity
DE = Working variable for annuity
BP = Future value of annuity
TD = Total amount of deposits
BA = Accumulated balance
TI = Total interest

Financial Interest

```
100 REM **FINANCIAL INTERESTS**
110 REM EXTENDED BASIC REQUIRED
120 DIM A(5)
130 CALL CLEAR
140 REM *******INITIALIZATION & TITLE SCREEN
    *******
```

```
150 FOR I=Ø TO 1Ø :: READ C$ :: B$(I)=C$ ::
    CALL COLOR(I,2,8):: NEXT I
160 FOR I=9 TO 14 :: CALL COLOR(I,I,I):: NEX
    T I :: CALL VCHAR(1,31,12Ø,96):: CALL SC
    REEN(12)
170 C=96 :: X=8 :: Y=1Ø :: GOSUB 18Ø :: C=12
    Ø :: X=12 :: Y=14 :: GOSUB 18Ø :: GOTO 1
    9Ø
180 FOR I=X TO Y :: DISPLAY AT(I,1):RPT$(CHR
    $(C),1Ø);TAB(18);RPT$(CHR$(C),11):: C=C+
    8 :: NEXT I :: RETURN
190 FOR I=6 TO 16 :: DISPLAY AT(I,12)SIZE(-5
    ):B$(I-6):: NEXT I
200 DATA "  $  "," $$$ ","$ $ $","$ $ $"," $
    $  ","  $  ","  $$ "
210 DATA "$ $ $","$ $ $"," $$$ ","  $  "
220 CALL SPRITE(#1,36,2,188,12Ø):: CALL MAGN
    IFY(2):: V=-14 :: H=-13 :: R=76 :: C=16
    :: J=Ø :: GOSUB 28Ø
230 V=Ø :: H=27 :: R=76 :: C=24Ø :: RR=11 ::
     CC=3 :: J=1 :: C$="FINANCIAL{3 SPACES}$
    {3 SPACES}INTERESTS" :: GOSUB 28Ø
240 J=Ø :: R=172 :: C=24 :: V=12 :: H=-27 ::
     GOSUB 28Ø
250 C=256 :: V=Ø :: H=27 :: RR=23 :: CC=4 ::
     J=1 :: C$="*PRESS ANY KEY TO BEGIN*" ::
     GOSUB 28Ø
260 CALL DELSPRITE(#1)
270 CALL KEY(Ø,K,S):: IF S=Ø THEN 27Ø ELSE 3
    2Ø
280 CALL MOTION(#1,V,H):: IF J=Ø THEN 3ØØ
290 FOR I=1 TO LEN(C$):: X=ASC(SEG$(C$,I,1))
    :: CALL HCHAR(RR,CC+I,X):: NEXT I
300 CALL COINC(#1,R,C,12,Z):: IF Z=Ø THEN 3Ø
    Ø :: CALL MOTION(#1,Ø,Ø):: CALL LOCATE(#
    1,R,C):: RETURN
310 REM ******MAIN MENU*******
315 CALL VCHAR(1,3,32,672):: RETURN
320 GOSUB 315 :: DISPLAY AT(5,5)BEEP:"FINANC
    IAL INTERESTS": : : :"PRESS{3 SPACES}FOR
    ": : :"  1  =  LOAN ANALYSIS": :"  2  =
     MORTGAGE ANALYSIS"
330 DISPLAY AT(16,3):"3  =  SAVINGS ANALYSIS
    ": :"  4  =  FINISH SESSION" :: CALL KEY
    (Ø,K,S):: IF K<49 OR K>52 THEN 33Ø
340 ON K-48 GOTO 38Ø,4ØØ,81Ø,36Ø
350 REM ******FINISH SESSION*******
360 DISPLAY AT(14,7)ERASE ALL:"HAVE A NICE D
    AY!" :: STOP
370 REM ******GET LOAN INFORMATION*******
```

```
380 B$(Ø)="THE AMOUNT OF LOAN:" :: B$(1)="TH
    E RATE OF INTEREST:" :: B$(2)="LENGTH OF
     LOAN IN MONTHS:" :: GOTO 41Ø
390 REM *******GET MORTGAGE INFORMATION*****
    **
400 B$(Ø)="THE AMOUNT TO BE MORTGAGED:" :: B
    $(1)="THE RATE OF INTEREST:" :: B$(2)="M
    ORTGAGE LENGTH IN MONTHS:"
410 GOSUB 315 :: J=Ø :: FOR I=5 TO 13 STEP 4
     :: DISPLAY AT(I,1):B$(J):: J=J+1 :: NEX
    T I
420 J=Ø :: FOR I=7 TO 15 STEP 4 :: ACCEPT AT
    (I,3)SIZE(6)VALIDATE(NUMERIC)BEEP:A(J)::
     J=J+1 :: NEXT I :: AMT=A(Ø):: R=A(1)::
    M=A(2)
430 IF K=49 THEN 47Ø
440 REM *******CALCULATE MORTGAGE PAYMENT***
    ****
450 FR=(1+R/2ØØ)^(1/6)-1 :: PA=INT(AMT*FR/(1
    -1/((1+FR)^M))*1ØØ+.5)/1ØØ :: GOTO 49Ø
460 REM *******CALCULATE LOAN PAYMENT*******
470 FR=(1+R/12ØØ)-1 :: PA=INT((AMT*FR)/(1-(1
    /(1+FR)^M))*1ØØ+.5)/1ØØ
480 REM *******DISPLAY LOAN AND MORTGAGE PAY
    MENT*******
490 GOSUB 315 :: DISPLAY AT(5,3)BEEP:"TO BOR
    ROW $";AMT: :"  FOR";M;"MONTHS AT";R;"%"
    : : :"MONTHLY PAYMENT WILL BE:"
500 DISPLAY AT(12,2):USING "######.##":PA ::
     DISPLAY AT(22,5):"*PRESS I FOR INDEX*":
    "*ANY OTHER KEY FOR ANALYSIS*"
510 CALL KEY(Ø,KEY,S):: IF S=Ø THEN 51Ø :: I
    F KEY=73 THEN 32Ø
520 REM *******MONTH BY MONTH ANALYSIS******
    *
530 GOSUB 315 :: DISPLAY AT(1,7):"MONTHLY AN
    ALYSIS": :"{7 SPACES}DO YOU WISH TO":"
    PRINT THE AMORTIZATION? Y/N": :"PRINCIPA
    L":"REMAINING ="
540 DISPLAY AT(1Ø,3):"MONTHLY":"  PAYMENT ="
    : :" PAYMENTS":" TO  DATE =": :" INTERES
    T":"THIS MNTH =": :" INTEREST":" TO  DAT
    E ="
550 ACCEPT AT(5,28)SIZE(-1)VALIDATE("YN")BEE
    P:C$ :: CALL HCHAR(4,3,32,28):: CALL HCH
    AR(5,3,32,28):: IF C$="N" THEN 58Ø
560 DISPLAY AT(4,1):"ENTER PRINTER DEVICE NA
    ME:" :: ACCEPT AT(5,3)BEEP:P$ :: CALL HC
    HAR(4,1,32,64):: OPEN #1:P$
570 REM *******ANALYSIS CALCULATIONS*******
```

```
580 F,TI,TP,MON=Ø :: PA=PA*1ØØ :: BA=AMT*1ØØ
590 FOR Z=1 TO M
600 IN=INT(BA*FR+.5):: IF Z=M THEN PA=BA+IN
610 TP=TP+PA :: BA=BA-PA+IN :: TI=TI+IN
620 IF BA>Ø THEN 63Ø :: PA=PA+BA :: TP=TP+BA
     :: BA=Ø
630 DISPLAY AT(4,1):"{4 SPACES}MONTH =
    {7 SPACES}";Z :: IF C$="N" THEN 7ØØ :: I
    F F=1 THEN 67Ø
640 REM ******PRINT AMORTIZATION SCHEDULE**
    *****
650 PRINT #1:TAB(27);"AMORTIZATION SCHEDULE"
    : : :TAB(1Ø);"PRINCIPAL:";AMT;TAB(35);"R
    ATE:";R;TAB(55);"MONTHS:";M: :
660 PRINT #1:" MONTH{4 SPACES}PAYMENT
    {4 SPACES}INTEREST{3 SPACES}PRINCIPAL
    {5 SPACES}BALANCE{5 SPACES}TOT/INT
    {3 SPACES}TOT/PAYMT": : :: F=1
670 A(Ø)=PA/1ØØ :: A(1)=IN/1ØØ :: A(2)=PA/1Ø
    Ø-IN/1ØØ :: A(3)=BA/1ØØ :: A(4)=TI/1ØØ :
    : A(5)=TP/1ØØ
680 PRINT #1,USING "#####":Z;:: FOR I=Ø TO 5
     :: PRINT #1,USING "#########.##":A(I);:
    : NEXT I :: PRINT #1:"" :: GOTO 71Ø
690 REM ******DISPLAY CALCULATIONS*******
700 IF Z=MON OR Z=M THEN 71Ø :: IF K=84 OR K
    =77 THEN 76Ø
710 A(Ø)=BA/1ØØ :: A(1)=PA/1ØØ :: A(2)=TP/1Ø
    Ø :: A(3)=IN/1ØØ :: A(4)=TI/1ØØ
720 J=Ø :: FOR I=8 TO 2Ø STEP 3 :: DISPLAY A
    T(I,14):USING "#######.##":A(J):: J=J+1
    :: NEXT I
730 IF Z=M THEN 77Ø :: IF C$="Y" THEN 76Ø ::
     DISPLAY AT(23,1)BEEP:"T=FOR TOTALS  M=S
    ELECT MONTH":"*ANY OTHER KEY  TO CONTINU
    E*"
740 CALL KEY(Ø,K,S):: IF S=Ø THEN 74Ø :: IF
    K<>77 THEN 76Ø
750 DISPLAY AT(4,1):"SELECT WHICH MONTH:" ::
     ACCEPT AT(4,21)VALIDATE(DIGIT)SIZE(3)BE
    EP:MON :: IF MON<=Z THEN 75Ø
760 NEXT Z
770 IF C$="N" THEN 78Ø :: CLOSE #1
780 DISPLAY AT(23,1)BEEP:"PRESS ANY KEY FOR
    MAIN INDEX":RPT$(" ",28)
790 CALL KEY(Ø,K,S):: IF S=Ø THEN 79Ø ELSE 3
    2Ø
800 REM ******GET SAVINGS INFORMATION******
    *
```

```
810 B$(Ø)="PRESENT AMOUNT IN SAVINGS:" :: B$
    (1)="RATE OF INTEREST:" :: B$(2)="TIMES
    COMPOUNDED PER YEAR:"
820 B$(3)="MAKE REGULAR DEPOSITS? (Y/N)" ::
    B$(4)="HOW MANY DEPOSITS PER YEAR:" :: B
    $(5)="HOW MUCH PER DEPOSIT:" :: GOSUB 31
    5
830 J=Ø :: FOR I=3 TO 21 STEP 4
840 DISPLAY AT(I,1):B$(J):: J=J+1 :: IF I<>1
    5 THEN 85Ø :: I=17 :: GOTO 84Ø
850 NEXT I
860 J=Ø :: FOR I=5 TO 23 STEP 4 :: IF I<>17
    THEN 89Ø
870 ACCEPT AT(15,23)VALIDATE("YN")SIZE(-1)BE
    EP:C$ :: IF C$="N" THEN 88Ø :: I=19 :: G
    OTO 89Ø
880 A(3),A(4)=Ø :: GOTO 9ØØ
890 ACCEPT AT(I,3)SIZE(6)VALIDATE(NUMERIC)BE
    EP:A(J):: J=J+1 :: NEXT I
900 AMT=A(Ø):: R=A(1):: C=A(2):: ND=A(3):: D
    =A(4)
910 REM ******SAVINGS ANALYSIS******
920 GOSUB 315 :: DISPLAY AT(3,7):"SAVINGS AN
    ALYSIS": :"YEARS IN THIS ANALYSIS?": :
     :"BEGINNING":"PRINCIPAL =": :"
    {4 SPACES}TOTAL":" DEPOSITS ="
930 DISPLAY AT(15,3):"ACCRUED":"  BALANCE ="
    : :"{4 SPACES}TOTAL":" INTEREST ="
940 ACCEPT AT(6,25)VALIDATE(DIGIT)SIZE(4)BEE
    P:Y :: IF Y=Ø THEN 94Ø :: DISPLAY AT(23,
    2):"{3 SPACES}ONE MOMENT PLEASE...":RPT$
    (" ",28)
950 REM ******ANALYSIS CALCULATIONS******
960 CP=(1+R/(1ØØ*C))^(Y*C):: B=INT(AMT*CP*1Ø
    Ø+.5)/1ØØ :: MA=(CP-1)/(R/(1ØØ*C)):: DE=
    D*ND/C :: BP=INT(DE*MA*1ØØ+.5)/1ØØ
970 TD=D*ND*Y :: BA=B+BP :: TI=BA-AMT-TD
980 REM ******DISPLAY CALCULATIONS******
990 A(Ø)=AMT :: A(1)=TD :: A(2)=BA :: A(3)=T
    I
1000 J=Ø :: FOR I=1Ø TO 19 STEP 3 :: DISPLAY
      AT(I,14):USING "#########.##":A(J)::
     J=J+1 :: NEXT I
1010 DISPLAY AT(23,2)BEEP:"*M=MORE SAVINGS A
     NALYSIS*":"ANY OTHER KEY FOR MAIN INDEX
     "
1020 CALL KEY(Ø,KEY,S):: IF S=Ø THEN 1Ø2Ø ::
      IF KEY=77 THEN 94Ø ELSE 32Ø :: STOP
```

A Mini Data Base Management System

Raymond J. Herold

A Data Base Management System (DBMS) is, in its simplest form, a system for managing large amounts of diversified data. These two programs will allow you to store, update or delete records, sort data, save files to tape, and print reports. Requires Extended BASIC.

This Mini Data Base Management System (DBMS), which actually consists of two programs, was written for the TI-99/4A in Extended TI BASIC. Most of the people who purchase a TI computer are first-time computer owners. In addition, most TI-99/4A owners do not have disk drives and memory expansion for their systems. My purpose in writing "MINI-DBMS" was to provide a useful software tool that was relatively powerful, easy to use, and would run on a minimum TI-99/4A configuration. This minimum configuration consists of the basic 16K TI-99/4A, monitor or TV, cassette player and Extended BASIC (which I consider essential).

Roadblocks

The first obstacle to writing a program such as this was the 16K memory limitation. How do you include all the features the program should have to make it useful, yet still leave enough memory for the data? The first trade-off required splitting MINI-DBMS into two programs. The first, MINI-DBMS, would be responsible for defining new files, adding and updating records and sorting the file. The second, "MINI-REPT," would handle the summarization and reporting requirements.

Then came the question of the records themselves. Trade-off number two: there would be a maximum of eight data fields per record. This should be enough for most home applications. In considering the data fields, a maximum of 20 characters per field seemed reasonable. The above two trade-offs then determined the third: a maximum of 80 records per file, depending on the record size. Again, this seemed reasonable for the typical home application.

Consequently, the MINI-DBMS parameters break down like this: two programs with the features deemed essential; up to 80 records per file; 1 to 8 fields in each record; and 1 to 20 characters for each field. Not too bad for a 16K machine!

The programs are written so that they can easily be merged if you have more memory and a disk drive. These two items will allow you to expand the basic parameters of MINI-DBMS. The major program subdivisions are outlined below in Table 1. Should you decide to make modifications to the program, Table 2 lists the variable names and their use.

MINI-DBMS

Program 1 is MINI-DBMS. This program allows you to define new DBMS files, add records to a file, display, update or delete records, sort a file, and save a file to tape. When you first type in RUN the program displays the introduction banner then displays the main menu:

1—DEFINE NEW DBMS RECORDS
2—LOAD RECORDS FROM TAPE
3—ENTER NEW RECORDS
4—DISPLAY/UPDATE RECORDS
5—SORT BY SPECIFIED FIELD
6—SAVE DATA ON TAPE

Define new DBMS records. This is where you define what a particular file will look like. The information you must supply includes: filename (up to eight characters); numbers of data fields in each record (maximum allowed is eight); and define each field.

Field definition involves a number of steps. To start, give each field a 1-to-6-character field name. This name (including the periods if you leave them in) will be used to identify the field when requesting functions such as search, sort, or summarize. You must then define the field length (maximum length is 20 characters). Finally, you will tell the program whether the field is alpha or numeric format. Alpha fields permit any character to be entered; numeric fields will only allow 0–9, comma, and period. In addition, only a numeric field can be summarized. Figure 1 is an example of field definition.

Although the new file has now been defined this step is not quite complete. The program will allow you to set an initial value, or mask, for each field. These masks allow you to format fields for data input. They will override the default values which are period for alpha fields and zeros for numeric ones. You can see in Figure 2 that the DATE field was given a mask of 00/00/00 rather than periods, and the AMOUNT field was given a decimal point. The remaining fields use the default value. You can override the periods with a mask of blanks if you so desire, but the periods are useful in showing whoever is entering data how many characters they have to work with.

Figure 1. Create New DBMS Files

FIELD NAME	LENGTH (1-20)	TYPE (N/A)
NAME..	20	A
ADDR..	20	A
CITYST	20	A
ZIP...	05	N
DATE..	08	A
AMOUNT	08	N
FOR...	20	A

Figure 2. Set Initial Values

SET INITIAL VALUES

NAME..	....
ADDR..	....
CITYST	....
Zip...	00000
DATE..	00/00/00
AMOUNT	00000.00
FOR...	....

Load records from tape. This option will allow you to load an existing file on tape into the MINI-DBMS program. The program will first read the filename on the tape and ask you if it is the one you wanted.

Enter new records. Here is where you begin with a newly defined file, or add to an existing file loaded from tape. The program will display a screen with the name of each field in the record and its associated mask. You simply enter the data you want for each new record. After the record is added a display will show how many records are currently in the file and the maximum number allowed for that file. At this point you can add another record or return to the main program menu. Figure 3 shows a record that has just been added.

Figure 3. Add New Record

	ADD NEW RECORD
NAME..	COMPUTE! ....
ADDR..	P.O. BOX 5406 ....
CITYST	GREENSBORO NC ....
ZIP..	27403
DATE..	10/04/84
AMOUNT	00024.00
FOR...	SUBSCRIPTION ....

Display/update records. There are two methods available for displaying records. The first displays each record starting at the beginning of the file. Pressing the ENTER key displays the next record. Pressing M will return you to the program menu from anywhere in the file. Pressing U will put the program in update mode for the record being displayed. The cursor will appear in the leftmost position of the first data field. You can change the data in the field or press ENTER to put the cursor in the next field. This process continues until all fields have been updated or bypassed.

If you want to completely delete the record from the file enter $DEL into the first four positions of the first data field. This assumes that the first field is alpha format and at least four characters long. If you want to use a different control code or field you can change the IF statement in line 4146.

Method two displays and updates the records in the same manner as method one. The difference lies in which records are displayed. This second method allows you to search the file for a desired value in a particular field. Only records meeting the search criteria are displayed, thus eliminating the need to scroll through unwanted records. The search argument may be a generic value. That is, the argument "SMI" would display records for SMITH, SMITHERS, SMILEY, etc.

Sort by specified field. You can sort the file into ascending sequence on any field. Just provide the name of the field you want sorted. BASIC is a slow language for routines such as sorts, but the exchange sort which starts at line 5000 will sort most files in less than five minutes. The program will continually display the number of sort passes left. You can change the sort to descending sequence by changing the less than sign in line 5065 to a greater than sign, and by changing the A$ assignment in line 5050 to A$(0,Z)=" ".

Save data on tape. Depending on the size of the file, saving data to tape may be even slower than the sort. But then, no one purchases a home computer for its tape I/O speed.

MINI-REPT

MINI-REPT handles the summarization and reporting responsibilities of the MINI-DBMS system. The program menu provides the following options:

1—LOAD RECORDS FROM TAPE
2—DISPLAY RECORDS
3—SUMMARIZE BY FIELD(S)
4—PRODUCE PRINTED REPORT

The first two options function the same way as in Program 1, except that there is no update capability for DISPLAY RECORDS.

Summarize by field(s). You can summarize (total) a field based on the value of one or two search fields. To summarize using one search field you provide the name of the field to be searched, the search argument (which may be generic), and the name of the field to be summarized. The value of the search field and summary field for all records meeting the search criteria will be displayed. Once the entire file has been searched, the program will display the number of records meeting the search criteria and the total for the summary field.

It is possible to search on two fields. By providing the name of the two fields and their respective search arguments, you can have the program summarize only those records meeting the search criteria for both fields.

If you specify the second search field argument as $ALL, the program will qualify all records meeting the first search field criteria only. This allows you to display the second search field value as an identifier.

Produce printed report. This option is for those of you with printers. It allows you to produce a report of the data in the field. You first provide the program with the number of fields you want printed, and the name of the fields. You may summarize a field if desired, and you may selectively print based on the value of a search argument. The report to be printed may be given a title. The program checks for a maximum 80 columns of print data, but allows you to print more if desired.

If you request this option but don't have a printer you will get a syntax error. Also, you may have to adjust the OPEN statement in line 8006 to accommodate your particular printer.

Table 1. Program Subdivisions

Line Number	
10–30	Introduction banner
100–190	Menu display
1000–1400	Define new DBMS file
2000–2060	Read data from tape
3000–3200	Add records to file
4000–4440	Display/update records
5000–5220	Sort routine
6000–6050	Write data to tape
7000–7450	Search, summarize and display
8000–8200	Produce printed report

Table 2. Program Variables

A$(s,s)	File data array (record, field)
ARG$	First search argument
ARG2$	Second search argument
FIELDS	Number of fields in record
FLD	Field number of user-entered field name
FLD$	User-entered field name
FNM$(s)	Field name array
KEY	Value of key pressed
LN(s)	Field length array
MASK$(s)	Initial field value array
NAME$	Filename
NF	Number of fields to be printed
NUMREC	Maximum number of records in file
OPT$	User-entered option (Y or N)
P(s)	Field to be printed
P$(s)	Name of field to be printed

RECS	Number of search fields
SF	Number of search fields
STATUS	CALL KEY status
SUM	Number of field to be summarized
SUM$	Name of field to be summarized
TC	Total characters per line in print function
TI	Items selected in summary function
TOT	Accumulator for summary field
TOTCHR	Total characters per record
TYP$(s)	Field type array
X,L,Z,Q	Loop control

Program 1. MINI-DBMS

```
1 REM TI MINI-DBMS
2 REM
10 CALL CLEAR :: CALL SCREEN(9)
20 DISPLAY AT(3,1):RPT$("*",28):: DISPLAY AT
   (4,1):"*" :: DISPLAY AT(4,28):"*"
22 DISPLAY AT(5,1):"*{4 SPACES}M I N I - D B
    M S{5 SPACES}*"
24 DISPLAY AT(6,1):"*" :: DISPLAY AT(6,28):"
   *" :: DISPLAY AT(7,1):RPT$("*",28)
30 FOR X=1 TO 2000 :: NEXT X
40 DIM A$(81,8)
100 CALL CLEAR :: CALL SCREEN(8):: DISPLAY A
    T(2,10):"** MENU **"
110 DISPLAY AT(6,1):"1 - DEFINE NEW DBMS REC
    ORDS" :: DISPLAY AT(8,1):"2 - LOAD RECOR
    DS FROM TAPE"
120 DISPLAY AT(10,1):"3 - ENTER NEW RECORDS"
     :: DISPLAY AT(12,1):"4 - DISPLAY/UPDATE
     RECORDS"
130 DISPLAY AT(14,1):"5 - SORT BY SPECIFIED
    FIELD"
140 DISPLAY AT(16,1):"6 - SAVE DATA ON TAPE"
150 DISPLAY AT(23,3):"ENTER SELECTION--->" :
    : ACCEPT AT(23,23)VALIDATE("123456")BEEP
    :CHOICE
170 ON CHOICE GOTO 1000,2000,3000,4000,5000,
    6000
190 GOTO 100
400 FOR X=1 TO 2000 :: NEXT X
410 GOTO 100
1000 CALL CLEAR :: DISPLAY AT(3,1):"DEFINE R
     ECORD FORMAT FOR" :: DISPLAY AT(4,1):"N
     EW DBMS. YOU MAY DEFINE UP"
1005 DISPLAY AT(5,1):"TO 8 FIELDS IN THE REC
     ORD."
```

```
1006 IF RECS>0 THEN DISPLAY AT(8,1):"DELETE
     CURRENT FILE? Y/N"
1007 IF RECS>0 THEN CALL KEY(3,KEY,STATUS)::
      IF STATUS=0 THEN 1007 ELSE IF KEY<>
      89 THEN 100
1010 DISPLAY AT(8,1):"NEW DBMS NAME: .......
     ." :: ACCEPT AT(8,16)SIZE(-8)BEEP:NAME$
1015 DISPLAY AT(9,1):"NUMBER OF FIELDS (1-8)
     " :: ACCEPT AT(9,25)VALIDATE(DIGIT)SIZE
     (2)BEEP:FIELDS
1016 IF FIELDS>8 THEN 1015
1018 DISPLAY AT(11,1):"  FIELD{5 SPACES}LENG
     TH{3 SPACES}TYPE" :: DISPLAY AT(12,1):"
       NAME{6 SPACES}(1-20){3 SPACES}(N/A)"
1020 FOR L=1 TO FIELDS
1022 DISPLAY AT(13+L,1):"  ......{6 SPACES}0
     0{7 SPACES}."
1023 NEXT L
1030 FOR L=1 TO FIELDS
1032 ACCEPT AT(13+L,3)SIZE(-6)BEEP:FNM$(L)
1034 ACCEPT AT(13+L,15)VALIDATE(DIGIT)SIZE(-
     2)BEEP:LN(L):: IF LN(L)<01 OR LN(L)>20
     THEN 1034
1036 ACCEPT AT(13+L,24)VALIDATE("AN")SIZE(-1
     )BEEP:TYP$(L):: IF TYP$(L)="." THEN
     1036
1038 NEXT L
1040 CALL CLEAR :: DISPLAY AT(2,2):"** SET I
     NITIAL VALUES **" :: DISPLAY AT(4,1):"K
     EY IN THE DEFAULT VALUE FOR"
1042 DISPLAY AT(5,1):"EACH FIELD OR PRESS EN
     TER TO" :: DISPLAY AT(6,1):"ACCEPT AS I
     S."
1050 FOR L=1 TO FIELDS
1052 DISPLAY AT(10+L,1):FNM$(L)
1054 IF TYP$(L)="N" THEN GOSUB 1100 ELSE GOS
     UB 1200
1056 NEXT L
1060 FOR L=1 TO FIELDS
1062 IF TYP$(L)="N" THEN GOSUB 1300 ELSE GOS
     UB 1400
1064 NEXT L
1070 TC=0 :: FOR L=1 TO FIELDS
1072 TC=TC+LN(L)
1074 NEXT L
1076 NUMREC=INT(4300/TC):: IF NUMREC>80 THEN
      NUMREC=80
1078 CALL CLEAR :: DISPLAY AT(4,1):"YOUR FIL
     E WILL HOLD ";NUMREC;"RECORDS"
1080 FOR X=1 TO 2000 :: NEXT X
```

```
1090 RECS=0 :: GOTO 100
1100 DISPLAY AT(10+L,8):RPT$("0",LN(L)):: RE
     TURN
1200 DISPLAY AT(10+L,8):RPT$(".",LN(L)):: RE
     TURN
1300 ACCEPT AT(10+L,8)VALIDATE("0123456789,.
     ")SIZE(-LN(L))BEEP:MASK$(L):: RETURN
1400 ACCEPT AT(10+L,8)SIZE(-LN(L))BEEP:MASK$
     (L):: RETURN
2000 CALL CLEAR
2010 OPEN #1:"CS1",INPUT ,INTERNAL,FIXED 192
2020 INPUT #1:NAME$,FIELDS,RECS,NUMREC
2022 PRINT :: PRINT "INPUT FILE - ";NAME$ ::
      PRINT "CONTINUE? Y/N"
2024 CALL KEY(3,KEY,STATUS):: IF STATUS=0 TH
     EN 2024
2026 IF KEY<>89 THEN 2050
2030 FOR L=1 TO FIELDS :: INPUT #1:FNM$(L),T
     YP$(L),LN(L),MASK$(L):: NEXT L
2035 FOR X=1 TO RECS
2040 INPUT #1:A$(X,1),A$(X,2),A$(X,3),A$(X,4
     ),A$(X,5),A$(X,6),A$(X,7),A$(X,8)
2045 NEXT X
2050 CLOSE #1
2060 GOTO 100
3000 IF NUMREC<1 THEN CALL CLEAR :: DISPLAY
     AT(4,1):"NO FILE DEFINED." :: GOTO 400
3004 RECS=RECS+1
3005 IF RECS>NUMREC THEN CALL CLEAR :: DISPL
     AY AT(4,1):"RECORD MAXIMUM EXCEEDED" ::
      GOTO 400
3010 CALL CLEAR :: DISPLAY AT(3,7):"** ADD N
     EW DATA **"
3020 FOR L=1 TO FIELDS
3022 DISPLAY AT(5+L,1)SIZE(LEN(FNM$(L))):FNM
     $(L):: DISPLAY AT(5+L,9)SIZE(LEN(MASK$(
     L))):MASK$(L)
3024 NEXT L
3030 FOR L=1 TO FIELDS
3032 IF TYP$(L)="N" THEN GOSUB 3100 ELSE GOS
     UB 3200
3034 NEXT L
3036 DISPLAY AT(17,1):"# RECORDS:";RECS;"
     {3 SPACES}MAX:";NUMREC
3040 CALL HCHAR(18,1,95,31)
3042 DISPLAY AT(20,1):"1 - TO ENTER ANOTHER
     RECORD" :: DISPLAY AT(21,1):"2 - TO RET
     URN TO MENU" :: DISPLAY AT(23,3):"ENTER
      YOUR CHOICE --->"
3044 ACCEPT AT(23,26)VALIDATE("12")BEEP:OPT
```

```
3046 IF OPT=1 THEN 3000
3048 IF OPT=2 THEN 100
3050 GOTO 3044
3100 ACCEPT AT(5+L,9)VALIDATE("0123456789,."
     )SIZE(-LEN(MASK$(L)))BEEP:A$(RECS,L)::
     RETURN
3200 ACCEPT AT(5+L,9)SIZE(-LEN(MASK$(L)))BEE
     P:A$(RECS,L):: RETURN
4000 CALL CLEAR :: DISPLAY AT(3,2):"** DISPL
     AY/UPDATE DATA **"
4010 DISPLAY AT(6,1):"1 - DISPLAY ALL RECORD
     S" :: DISPLAY AT(7,1):"{4 SPACES}FROM B
     EGINNING OF FILE."
4015 DISPLAY AT(9,1):"2 - DISPLAY BY VALUE I
     N" :: DISPLAY AT(10,1):"{4 SPACES}SPECI
     FIED FIELD"
4020 DISPLAY AT(12,3):"ENTER YOUR CHOICE ---
     >" :: ACCEPT AT(12,26)VALIDATE("12")BEE
     P:OPT
4030 ON OPT GOTO 4100,4200
4050 GOTO 100
4100 HOLD=RECS
4110 FOR RECS=1 TO HOLD
4115 IF RECS>HOLD THEN 4150
4120 GOSUB 4300
4130 CALL KEY(3,KEY,STATUS)
4132 IF STATUS=0 THEN 4130
4134 IF KEY=13 THEN 4150
4136 IF KEY=77 THEN RECS=999 :: GOTO 4150
4138 IF KEY<>85 THEN 4130
4140 FOR L=1 TO FIELDS
4142 IF TYP$(L)="N" THEN GOSUB 3100 ELSE GOS
     UB 3200
4144 NEXT L
4146 IF SEG$(A$(RECS,1),1,4)="$DEL" THEN GOS
     UB 4400
4150 NEXT RECS
4155 RECS=HOLD
4160 GOTO 100
4200 CALL CLEAR :: DISPLAY AT(2,1):"** DISPL
     AY BY FIELD VALUE **"
4205 DISPLAY AT(5,1):"ENTER THE NAME OF THE
     DATA" :: DISPLAY AT(6,1):"FIELD TO BE S
     EARCHED AND THE"
4210 DISPLAY AT(7,1):"SEARCH ARGUMENT (VALUE
     )."
4220 DISPLAY AT(12,1):"FIELD TO BE SEARCHED
     ......" :: ACCEPT AT(12,22)SIZE(-6):FLD
     $
4221 FLD=0
```

```
4222 FOR L=1 TO FIELDS
4224 IF FLD$=FNM$(L)THEN FLD=L :: L=99
4226 NEXT L
4228 IF FLD=Ø THEN DISPLAY AT(14,1):"NO SUCH
      FIELD NAME." :: DISPLAY AT(15,1):"'R'
     TO RETRY - 'M' FOR MENU" ELSE GOTO 425Ø
423Ø CALL KEY(3,KEY,STATUS)
4232 IF STATUS=Ø THEN 423Ø
4234 IF KEY=82 THEN 42ØØ ELSE 1ØØ
425Ø DISPLAY AT(14,1):"ENTER SEARCH VALUE
     {5 SPACES}"
4252 ACCEPT AT(15,1):ARG$
426Ø HOLD=RECS
4262 FOR RECS=1 TO HOLD
4264 IF ARG$=SEG$(A$(RECS,FLD),1,LEN(ARG$))T
     HEN GOSUB 43ØØ ELSE 429Ø
427Ø CALL KEY(3,KEY,STATUS)
4272 IF STATUS=Ø THEN 427Ø
4274 IF KEY=13 THEN 429Ø
4276 IF KEY=77 THEN RECS=999 :: GOTO 429Ø
428Ø IF KEY<>85 THEN 427Ø
4282 FOR L=1 TO FIELDS
4284 IF TYP$(L)="N" THEN GOSUB 31ØØ ELSE GOS
     UB 32ØØ
4286 NEXT L
429Ø NEXT RECS
4292 RECS=HOLD
4294 GOTO 1ØØ
43ØØ CALL CLEAR :: DISPLAY AT(2,1):"** DISPL
     AY/UPDATE RECORDS **"
431Ø FOR L=1 TO FIELDS
432Ø DISPLAY AT(5+L,1):FNM$(L):: DISPLAY AT(
     5+L,9):A$(RECS,L)
433Ø NEXT L
434Ø DISPLAY AT(2Ø,1):"PRESS ENTER FOR NEXT
     RECORD" :: DISPLAY AT(22,1):"PRESS 'U'
     TO UPDATE RECORD" :: DISPLAY AT(24,1):"
     PRESS 'M' FOR MENU"
435Ø RETURN
44ØØ CALL CLEAR :: DISPLAY AT(3,1):"STAND BY
     ..."
441Ø FOR X=RECS TO HOLD :: FOR Y=1 TO FIELDS
442Ø A$(X,Y)=A$(X+1,Y)
443Ø NEXT Y :: NEXT X
444Ø RECS=RECS-1 :: HOLD=HOLD-1 :: RETURN
5ØØØ CALL CLEAR :: DISPLAY AT(3,9):"** SORT
     **" :: DISPLAY AT(6,1):"NAME OF SORT FI
     ELD ......" :: ACCEPT AT(6,2Ø)SIZE(-6)B
     EEP:FLD$
```

```
5005 FLD=0
5010 FOR L=1 TO FIELDS
5012 IF FLD$=FNM$(L)THEN FLD=L :: L=99
5013 NEXT L
5015 IF FLD=0 THEN DISPLAY AT(14,1):"NO SUCH
      FIELD NAME." :: DISPLAY AT(15,1):"'R'
     TO RETRY - 'M' FOR MENU" ELSE GOTO 5040
5020 CALL KEY(3,KEY,STATUS):: IF STATUS=0 TH
     EN 5020
5030 IF KEY=82 THEN 5000 ELSE 100
5040 DISPLAY AT(20,1):"SORTING..."
5050 Y=1 :: HX=0 :: FOR Z=1 TO FIELDS :: A$(
     0,Z)="____" :: NEXT Z
5055 SS=0 :: DISPLAY AT(20,1):"SORTING...";R
     ECS-Y
5060 FOR X=Y TO RECS
5065 IF A$(X,FLD)<A$(0,FLD)THEN GOSUB 5100
5070 NEXT X
5075 IF SS=1 THEN GOSUB 5200
5080 Y=Y+1 :: FOR Z=1 TO FIELDS :: A$(0,Z)=A
     $(Y,Z):: NEXT Z
5085 IF Y<RECS THEN 5055
5090 GOTO 100
5100 FOR Z=1 TO FIELDS :: A$(0,Z)=A$(X,Z)::
     NEXT Z :: HX=X :: SS=1 :: RETURN
5200 FOR Z=1 TO FIELDS
5210 H$=A$(Y,Z):: A$(Y,Z)=A$(HX,Z):: A$(HX,Z
     )=H$
5220 NEXT Z :: RETURN
6000 CALL CLEAR
6010 OPEN #1:"CS1",OUTPUT,INTERNAL,FIXED 192
6020 PRINT #1:NAME$;FIELDS;RECS;NUMREC
6030 FOR L=1 TO FIELDS :: PRINT #1:FNM$(L);T
     YP$(L);LN(L);MASK$(L):: NEXT L
6035 FOR X=1 TO RECS
6040 PRINT #1:A$(X,1);A$(X,2);A$(X,3);A$(X,4
     );A$(X,5);A$(X,6);A$(X,7);A$(X,8)
6045 NEXT X
6050 CLOSE #1 :: GOTO 100
```

Program 2. MINI-REPT

```
1 REM TI MINI-REPT
2 REM
10 CALL CLEAR :: CALL SCREEN(9)
20 DISPLAY AT(3,1):RPT$("*",28):: DISPLAY AT
   (4,1):"*" :: DISPLAY AT(4,28):"*"
22 DISPLAY AT(5,1):"*{4 SPACES}M I N I - R E
    P T{5 SPACES}*"
```

```
24 DISPLAY AT(6,1):"*" :: DISPLAY AT(6,28):"
   *" :: DISPLAY AT(7,1):RPT$("*",28)
30 FOR X=1 TO 2000 :: NEXT X
40 DIM A$(81,8)
100 CALL CLEAR :: CALL SCREEN(8):: DISPLAY A
    T(2,10):"** MENU **"
110 DISPLAY AT(6,1):"1 - LOAD RECORDS FROM T
    APE" :: DISPLAY AT(8,1):"2 - DISPLAY REC
    ORDS"
140 DISPLAY AT(10,1):"3 - SUMMARIZE BY FIELD
    (S)" :: DISPLAY AT(12,1):"4 - PRODUCE PR
    INTED REPORT"
150 DISPLAY AT(23,3):"ENTER SELECTION--->" :
    : ACCEPT AT(23,23)VALIDATE("12345")BEEP:
    CHOICE
170 ON CHOICE GOTO 2000,4000,7000,8000
190 GOTO 100
2000 CALL CLEAR
2010 OPEN #1:"CS1",INPUT ,INTERNAL,FIXED 192
2020 INPUT #1:NAME$,FIELDS,RECS,NUMREC
2022 PRINT :: PRINT "INPUT FILE - ";NAME$ ::
      PRINT "CONTINUE? Y/N"
2024 CALL KEY(3,KEY,STATUS):: IF STATUS=0 TH
     EN 2024
2026 IF KEY<>89 THEN 2050
2030 FOR L=1 TO FIELDS :: INPUT #1:FNM$(L),T
     YP$(L),LN(L),MASK$(L):: NEXT L
2035 FOR X=1 TO RECS
2040 INPUT #1:A$(X,1),A$(X,2),A$(X,3),A$(X,4
     ),A$(X,5),A$(X,6),A$(X,7),A$(X,8)
2045 NEXT X
2050 CLOSE #1
2060 GOTO 100
4000 CALL CLEAR :: DISPLAY AT(3,6):"** DISPL
     AY DATA **"
4010 DISPLAY AT(6,1):"1 - DISPLAY ALL RECORD
     S" :: DISPLAY AT(7,1):"{4 SPACES}FROM B
     EGINNING OF FILE."
4015 DISPLAY AT(9,1):"2 - DISPLAY BY VALUE I
     N" :: DISPLAY AT(10,1):"{4 SPACES}SPECI
     FIED FIELD"
4020 DISPLAY AT(12,3):"ENTER YOUR CHOICE ---
     >" :: ACCEPT AT(12,26)VALIDATE("12")BEE
     P:OPT
4030 ON OPT GOTO 4100,4200
4050 GOTO 100
```

```
4100 HOLD=RECS
4110 FOR RECS=1 TO HOLD
4115 IF RECS>HOLD THEN 4150
4120 GOSUB 4300
4130 CALL KEY(3,KEY,STATUS)
4132 IF STATUS=0 THEN 4130
4134 IF KEY=13 THEN 4150
4136 IF KEY=77 THEN RECS=999 :: GOTO 4150
4138 GOTO 4130
4150 NEXT RECS
4155 RECS=HOLD
4160 GOTO 100
4200 CALL CLEAR :: DISPLAY AT(2,1):"** DISPL
     AY BY FIELD VALUE **"
4205 DISPLAY AT(5,1):"ENTER THE NAME OF THE
     DATA" :: DISPLAY AT(6,1):"FIELD TO BE S
     EARCHED AND THE"
4210 DISPLAY AT(7,1):"SEARCH ARGUMENT (VALUE
     )."
4220 DISPLAY AT(12,1):"FIELD TO BE SEARCHED
     ......" :: ACCEPT AT(12,22)SIZE(-6):FLD
     $
4221 FLD=0
4222 FOR L=1 TO FIELDS
4224 IF FLD$=FNM$(L)THEN FLD=L :: L=99
4226 NEXT L
4228 IF FLD=0 THEN DISPLAY AT(14,1):"NO SUCH
      FIELD NAME." :: DISPLAY AT(15,1):"'R'
     TO RETRY - 'M' FOR MENU" ELSE GOTO 4250
4230 CALL KEY(3,KEY,STATUS)
4232 IF STATUS=0 THEN 4230
4234 IF KEY=82 THEN 4200 ELSE 100
4250 DISPLAY AT(14,1):"ENTER SEARCH VALUE
     {5 SPACES}"
4252 ACCEPT AT(15,1):ARG$
4260 HOLD=RECS
4262 FOR RECS=1 TO HOLD
4264 IF ARG$=SEG$(A$(RECS,FLD),1,LEN(ARG$))T
     HEN GOSUB 4300 ELSE 4290
4270 CALL KEY(3,KEY,STATUS)
4272 IF STATUS=0 THEN 4270
4274 IF KEY=13 THEN 4290
4276 IF KEY=77 THEN RECS=999 :: GOTO 4290
4280 GOTO 4270
4290 NEXT RECS
4292 RECS=HOLD
4294 GOTO 100
4300 CALL CLEAR :: DISPLAY AT(2,6):"** DISPL
     AY RECORDS **"
```

```
4310 FOR L=1 TO FIELDS
4320 DISPLAY AT(5+L,1):FNM$(L):: DISPLAY AT(
     5+L,10):A$(RECS,L)
4330 NEXT L
4340 DISPLAY AT(20,1):"PRESS ENTER FOR NEXT
     RECORD" :: DISPLAY AT(24,1):"PRESS 'M'
     FOR MENU"
4350 RETURN
7000 CALL CLEAR :: DISPLAY AT(2,1):"** SUMMA
     RIZE BY FIELDNAME **"
7001 DISPLAY AT(4,1):"SEARCH 1 OR 2 FIELDS?
     ." :: ACCEPT AT(4,23)VALIDATE("12")BEEP
     :SF
7005 TOT=0 :: TI=0
7010 DISPLAY AT(6,1):"FIELD TO BE SEARCHED .
     ....." :: ACCEPT AT(6,22)SIZE(-6)BEEP:F
     LD$ :: FLD=0
7015 FOR L=1 TO FIELDS
7016 IF FLD$=FNM$(L)THEN FLD=L :: L=99
7017 NEXT L
7020 IF FLD=0 THEN DISPLAY AT(6,1):"NO SUCH
     FIELD NAME" :: DISPLAY AT(7,1):"'R' TO
     RETRY - 'M' FOR MENU" ELSE GOTO 7028
7022 CALL KEY(3,KEY,STATUS)
7024 IF STATUS=0 THEN 7022
7026 IF KEY=82 THEN 7000 ELSE 100
7028 DISPLAY AT(7,1):"ENTER SEARCH VALUE" ::
      ACCEPT AT(8,1):ARG$
7029 IF SF=2 THEN GOSUB 7300 :: IF FLD2=0 TH
     EN 100
7030 DISPLAY AT(17,1):"FIELD TO BE SUMMED ..
     ....." :: ACCEPT AT(17,20)SIZE(-6)BEEP:S
     UM$ :: SUM=0
7035 FOR L=1 TO FIELDS
7036 IF SUM$=FNM$(L)THEN SUM=L :: L=99
7037 NEXT L
7040 IF SUM=0 THEN DISPLAY AT(17,1):"NO SUCH
      FIELD NAME" :: DISPLAY AT(19,1):"'R' T
     O RETRY - 'M' FOR MENU" ELSE GOTO 7050
7042 CALL KEY(3,KEY,STATUS)
7044 IF STATUS=0 THEN 7042
7046 IF KEY=82 THEN 7030 ELSE 100
7050 IF TYP$(SUM)<>"N" THEN DISPLAY AT(17,1)
     :"NOT A NUMERIC FIELD" :: DISPLAY AT(19
     ,1):"'R' TO RETRY - 'M' FOR MENU" ELSE
     GOTO 7060
7052 CALL KEY(3,KEY,STATUS)
7054 IF STATUS=0 THEN 7052
7056 IF KEY=82 THEN 7030 ELSE 100
7060 CALL CLEAR :: HOLD=RECS
```

```
7062 FOR RECS=1 TO HOLD
7064 IF ARG$=SEG$(A$(RECS,FLD),1,LEN(ARG$))T
     HEN GOSUB 7400
7066 NEXT RECS
7068 RECS=HOLD
7070 PRINT :: PRINT :: PRINT USING "ITEMS ##
     {4 SPACES}TOTAL #######.##":TI,TOT
7072 PRINT :: PRINT "PRESS ANY KEY FOR MENU"
7074 CALL KEY(3,KEY,STATUS):: IF STATUS=0 TH
     EN 7074 ELSE 100
7100 H$="" :: FOR X=1 TO LEN(A$(RECS,SUM))
7110 IF SEG$(A$(RECS,SUM),X,1)>="0" AND SEG$
     (A$(RECS,SUM),X,1)<="9" OR SEG$(A$(RECS
     ,SUM),X,1)="." THEN GOSUB 7200
7120 NEXT X
7130 N=VAL(H$):: TOT=TOT+N :: TI=TI+1
7135 IF S=1 THEN RETURN
7140 PRINT A$(RECS,FLD);" ";A$(RECS,SUM)
7145 IF SF=2 THEN PRINT
7150 RETURN
7200 H$=H$&SEG$(A$(RECS,SUM),X,1):: RETURN
7300 DISPLAY AT(10,1):"2ND SEARCH FIELD ....
     .." :: ACCEPT AT(10,18)SIZE(-6)BEEP:FLD
     2$ :: FLD2=0
7310 FOR L=1 TO FIELDS
7320 IF FLD2$=FNM$(L)THEN FLD2=L :: L=99
7325 NEXT L
7330 IF FLD2=0 THEN DISPLAY AT(10,1):"NO SUC
     H FIELD" :: DISPLAY AT(11,1):"'R' TO RE
     TRY - 'M' FOR MENU" :: GOTO 7340
7332 DISPLAY AT(11,1):"ENTER 2ND SEARCH VALU
     E"
7334 ACCEPT AT(12,1)BEEP:ARG2$
7336 GOTO 7350
7340 CALL KEY(3,KEY,STATUS)
7342 IF STATUS=0 THEN 7340
7344 IF KEY=82 THEN 7300
7350 RETURN
7400 IF SF=1 THEN GOSUB 7100 :: RETURN
7405 IF ARG2$="$ALL" THEN 7440
7410 IF ARG2$=SEG$(A$(RECS,FLD2),1,LEN(ARG2$
     ))THEN 7440
7420 RETURN
7440 PRINT A$(RECS,FLD2):: GOSUB 7100
7450 RETURN
8000 CALL CLEAR :: DISPLAY AT(2,1):"** PRODU
     CE PRINTED REPORT **"
8005 TOT=0 :: P(0)=0
8006 OPEN #2:"RS232",OUTPUT,DISPLAY
```

```
8010 DISPLAY AT(5,1):"NUMBER OF FIELDS TO PR
     INT? ." :: ACCEPT AT(5,28)VALIDATE("123
     45678")BEEP:NF
8020 FOR L=1 TO NF
8022 DISPLAY AT(6+(2*L),1):USING "NAME OF FI
     ELD # - ......":L :: ACCEPT AT(6+(2*L),
     19)SIZE(-6)BEEP:P$(L)
8024 P(L)=0 :: FOR Z=1 TO FIELDS
8026 IF P$(L)=FNM$(Z)THEN P(L)=Z
8028 NEXT Z
8030 IF P(L)=0 THEN L=L-1
8032 NEXT L
8036 DISPLAY AT(23,1):"TOTAL A FIELD? Y/N ."
      :: ACCEPT AT(23,20)VALIDATE("YN")SIZE(
     -1)BEEP:OPT$ :: IF OPT$<>"Y" THEN 8050
8040 DISPLAY AT(24,1):"NAME OF FIELD ......"
      :: ACCEPT AT(24,15)SIZE(-6)BEEP:P$(0)
8042 FOR Z=1 TO FIELDS
8044 IF P$(0)=FNM$(Z)THEN P(0)=Z
8045 NEXT Z
8046 IF TYP$(P(0))="N" THEN 8050
8048 DISPLAY AT(24,1):"** INVALID OR NON-NUM
     ERIC **" :: FOR Z=1 TO 2000 :: NEXT Z :
     : GOTO 8040
8050 GOSUB 8100 :: TC=0 :: FOR Z=1 TO NF ::
     TC=TC+LN(P(Z)):: NEXT Z
8052 TC=TC+(2*NF)-2
8054 IF TC<80 THEN 8060
8055 DISPLAY AT(3,1):USING "REPORT WILL OVER
     FLOW BY ##":TC-80 :: DISPLAY AT(5,1):"'
     P' TO PRINT - 'M' FOR MENU"
8056 CALL KEY(3,KEY,STATUS)
8058 IF STATUS=0 THEN 8056
8059 IF KEY<>80 THEN 100
8060 CALL CLEAR :: DISPLAY AT(3,1):"ENTER RE
     PORT TITLE" :: ACCEPT AT(4,1)BEEP:RT$
8062 DISPLAY AT(10,1):"PRINTING..."
8063 PRINT #2:RPT$(" ",(80-LEN(RT$))/2);
8064 PRINT #2:RT$ :: PRINT #2:" " :: PRINT #
     2:" " :: PRINT #2:" "
8070 FOR Q=1 TO RECS
8071 IF OPT$="N" THEN 8074
8072 IF ARG$<>SEG$(A$(Q,FLD),1,LEN(ARG$))THE
     N 8080
8074 L$="" :: FOR L=1 TO NF
8075 L$=L$&A$(Q,P(L)):: IF L<NF THEN L$=L$&"
      "
8076 NEXT L
8078 PRINT #2:L$
8079 IF P(0)<>0 THEN GOSUB 8200
```

```
8080 NEXT Q
8082 IF P(0)=0 THEN 8090
8084 PRINT #2:" " :: PRINT #2:" " :: PRINT #
     2:"TOTAL FOR ";FNM$(P(0));" ";TOT
8090 CLOSE #2 :: GOTO 100
8100 CALL CLEAR :: DISPLAY AT(3,1):"SEARCH B
     Y FIELD NAME? Y/N ." :: ACCEPT AT(3,27)
     VALIDATE("YN")BEEP:OPT$ :: IF OPT$="N"
     THEN RETURN
8110 DISPLAY AT(5,1):"SEARCH FIELD NAME ....
     .." :: ACCEPT AT(5,19)SIZE(-6)BEEP:FLD$
8120 FLD=0 :: FOR L=1 TO FIELDS
8125 IF FLD$=FNM$(L)THEN FLD=L
8130 NEXT L
8140 IF FLD=0 THEN 8110
8150 DISPLAY AT(6,1):"ENTER SEARCH VALUE" ::
      ACCEPT AT(7,1):ARG$
8160 RETURN
8200 RECS=Q :: SUM=P(0):: S=1 :: GOSUB 7100
     :: S=0 :: RETURN
```

TI Word Processor

James D. Baker

This menu-based word processor includes many of the basic features of commercial word processors: text creation, addition, deletion, modification, paragraphs, pagination, margin control, page overflow, and text centering. Written for the TI-99/4A with Extended BASIC, a disk drive, and printer, the program runs with standard 16K memory.

Just like thousands of other TI users, I have added to my system since the original purchase of the computer and a TV set. After I had purchased Extended BASIC, the Peripheral Expansion Box, disk drive and controller, RS-232 interface, and a printer, my next choice was word processing capability. All the word processor packages available required 32K memory expansion, so I decided to write my own word processor.

This program runs with standard 16K memory because of *linked list* access for text files: Only one line of text is in memory at a time, with before and after indices pointing to the previous or following line of text.

With this design, addition and deletion of text lines are possible. The addition of a single line or an entire paragraph of text is also possible and, therefore, updating text after the initial input process is easy.

Automatic pagination, margins (top, bottom, left, and right), page overflow, text centering, and text modification are also included features.

The program is written in two distinct sections: first, the create/edit section, then the print section. If additional features are added, it may be necessary to split the program into two separate programs in order to maintain the objective of minimum memory usage.

Program Initialization

Upon initial execution of the program, the user will be asked for a filename (assumed on DSK1) where text is stored. The subroutine called in line 140 sets characters in lowercase.

Next, a screen menu is displayed with these options:

N—NEW DATA FILE
A—ADD TO END OF EXISTING FILE
C—CHANGE EXISTING FILE
P—PRINT FILE

New Data File

Upon selection of the first option, a header record is written to the opened disk file. This record is used to maintain a pointer to the last text record in the file. Initially, this record does not contain any meaningful information, but will be updated at the end of the program to contain the actual last record number.

Control is then passed to the routine for entering new text (lines 380–470). Original text is entered using the LINPUT statement, which limits the length of a single entry to 128 characters. However, this is not a severe limitation; the program will simply cause wraparound of the text from one record to the next. The computer will beep to remind you that you have exceeded the length of the input string, and you must then press ENTER to cause this record to be written to disk and begin entry of the next record. Also, note that during text entry all the standard control key operations are allowed, including cursor left or right, character delete or insert, erase, etc.

The pointers for previous and next record locations are then updated, and a check for one of the special control functions, /E/, is performed. This is used to indicate the end of text and must be entered as the last record of the text. If the record just entered is not the end marker (/E/), the program writes the text line to disk and returns for the next line of text.

When text entry is complete and the /E/ is entered, lines 490–510 update record 0 with the record number of the last record on file. Finally, the option of printing the text is offered. If you answer Y for yes, control is passed to the print routine (line 2400); otherwise the program ends.

Other special control functions are also included for editing. By entering /C/ as the first three characters of the text line, the print program will automatically center the text that follows on that line. By entering /P/ as the first three characters of a text line, the print program will automatically indent five spaces for a new paragraph. Also, by entering /N/ as the only three characters on a text line, the print program will automatically cause a top-of-page routine to be executed. These special control functions can be entered as upper- or lowercase letters.

Appending

When this second menu option is selected, control is passed to program line 600. This routine simply uses the pointer obtained from the first record on file to retrieve the last record on file (the /E/ record). Then the last actual text record is retrieved by using the previous record pointer from the /E/ record.

The last actual text record on file is then displayed, and control is passed to the routine used for original text entry.

Changing an Existing File

With this option, the program retrieves the first text record, using the pointer obtained from the first record on the file. This line of text and a change menu are then displayed:

```
1=NEXT LINE      5=ADD BEFORE
2=LAST LINE      6=ADD AFTER
3=FWD X LINES    7=CHANGE
4=BKW X LINES    8=DELETE
                 9=QUIT
```

Next Line. This option displays the next text line. If selected, program execution is transferred to line 900. This routine first sets the number-of-records-forward counter to one. The loop in lines 940–980 follows the next record pointer through the file until the requested number of records forward has been read.

A check is made to insure that a READ past the end of file does not occur. If this is attempted, the program displays the last line of text, a warning message, and returns to the main change menu. Upon completion of the loop, program control is returned to the main change menu.

It should be noted that the loop is not necessary in order to display the next line. However, it is also used to advance any number of records by using the third option discussed below.

Last Line. This option displays the previous line of text. The routine starting at line 1000 provides for stepping backward through the text file. This routine is the same as the prior routine except that the previous record pointer is used in order to proceed to the previous record.

FWD X Lines and BKW X Lines. Both of these options (3 and 4) are handled in the routine beginning at line 1100. The program asks for the number of lines to be read either forward or backward. This value is then placed in the appropriate counter, and control is transferred to the Next Line or Last Line routine.

Add Before and Add After. These options (5 and 6), initially handled by the same routine (at line 1100), allow for adding text; option 5 for adding before the current line, option 6 for adding after it. The program displays the current record and, based on which type of add was requested, prompts you to add before or after.

The new line of text is then entered and the record pointers from the current record are saved. The /E/ is retrieved in order to determine the next available location in the file to store a record (next record pointer). This value is saved, and then the /E/ record is rewritten with the next record pointer incremented. Based on the type of add being done, control is transferred to the appropriate routine.

If you select Add Before (option 5), control is passed to line 1350.

If you select Add After (option 6), control is passed to line 1450.

Control is then transferred to line 1430 and processing continues as discussed above.

Change. This option allows you to change an existing line of text. The routine for this option begins at line 1540. The text line is broken into 14 lines of "equal" length. Using the DISPLAY AT and ACCEPT AT statements allows the setting of default values for each of the subtext lines to their initial string value. This eliminates the necessity of retyping the entire line to make a minor correction.

The length of each of the subtext lines is calculated and the first 13 lines are displayed. Note that a special character is added to the end of each line. This is done so a space is not lost at the end of the subtext line.

Line 1650 determines if there is any text remaining for the fourteenth line. This is necessary to avoid an error if the string happens to be less than 13 times the rounded length of a single subtext line length. The fourteenth line is then displayed in preparation for change.

The 14 lines are then "looped" through, allowing any changes desired. Note that the maximum length of any subtext line is limited to 26 characters and that if the special end character is accidentally deleted, the program will restore this character. The length of the new text line is recalculated since this length could now exceed the maximum string length permitted by the computer.

After the text has been changed, the new text length is checked to see if it exceeds 225 characters. If the length is less than 226 characters, the text line is reconstructed and control is transferred to line 2050.

If the length of the new text line exceeds 225 characters, a menu offering two choices is displayed: either update as modified and create a new record on disk or reupdate the line. If the reupdate choice is selected, control is transferred to the beginning of the change routine with no changes made.

If the choice is made to update and create a new record, lines 1900–1940 establish two new text strings consisting of the first seven and last seven subtext lines respectively. The current record being changed is then replaced on disk by the first new text string created. The second new text string is then added to the file using the Add After routine. Note that the return switch has been set in line 1950 causing control to return to this routine after the add is completed.

The first of the new records is retrieved, and control is returned to display this as the current record and to display the main change menu.

If the change process did not cause a new record to be added, lines 2050–2130 display the changed text and offer three choices: perform more updates, update the record as displayed, or exit with no updating.

Delete. The routine for this option, which allows you to delete a line of text, begins at line 2180. You will be asked for confirmation before the delete is executed. If the choice is made not to delete the line, control is passed back to line 780 where the current line is redisplayed and the main menu choices are available.

If you choose to delete the line, the previous and next record pointers from this "to be deleted" record are saved. The previous record is then read and updated with the next record pointer from the deleted record. The record after the deleted record is then read and updated with the previous record pointer from the deleted record. Note that the record just deleted is only deleted from the standpoint that the record pointers no longer allow access to the record.

A check is then made to insure that this delete has not caused all text to be deleted. If this is the case, the program displays a message to that effect and terminates. Otherwise, if a record still exists before the deleted record, control is passed to line 1000 and the previous record is displayed. If the record prior to the deleted record is the header record, control is passed to line 900, and the record following the deleted record is displayed.

Print File

The print routine begins at line 2400. Lines 2480–2540 establish the default values for top margin (TM), bottom margin (BM), left margin (LM), page length (PL), lines per page (LPP), and maximum line length (MAXWID). Print control information is then requested, including mode of print (draft or final), spacing (single or double), and optional page numbering.

The input file is then "restored" to restart from the first record on file, and the printer output file is opened. Note that the parallel port is used in this program. If you are using the serial port for your printer, the OPEN statement in line 2730 will require appropriate changes.

The first record on file is read to retrieve the next record pointer for the first text record. The main print "loop" begins at line 2820 where the next text record is read using the next record pointer from the previous record.

If draft printing was requested, control is passed to that routine (line 2880). If the current record is a forced new page request (/N/), the subroutine at line 3900 causes a page eject and the top margin to be printed. Control is then returned to the main print loop.

Line 2850 passes control to the ending routine if this is the last text record. Otherwise, control is passed to the print final routine (line 2980).

Print Draft. This routine (lines 2870–2930) simply prints the lines of text in sequence exactly as entered. This includes printing any special print commands, but does not effect these commands. This is useful if you want to see what was entered for verification purposes and do not want pagination, etc. This print mode is also faster than final printing as the special print commands are not executed.

Print Final. This routine begins at line 2980 and prints as much text as will fit on the remainder of the print line, then prints character by character until a space is encountered.

The Print Final routine first checks for any special print commands. If a blank line, centered line, or new paragraph is requested, control is passed to the appropriate routine. If the last character on the text line is a period, two spaces are added to the end of the line to insure proper spacing.

The centering routine begins at line 3550 by printing any unfinished print line and checking for overflow. The length of the text to be centered (excluding the centering command) and the number of spaces required to center the text are then calculated. The line is then printed and control is passed to read the next record.

The routine to print a blank line begins at line 3700. This routine simply prints the preceding line, a blank line, checks for overflow and returns to read the next record.

The routines for top and bottom margins begin at line 3800 and simply loop for the necessary number of blank lines. Page numbering is handled on line 3940.

Lowercase Definition

Finally, the DATA statements in lines 3980–4240 represent lowercase letters. These values are assigned according to standard lowercase ASCII characters and are read using the loop in lines 4250–4290.

TI Word Processor

```
100 REM WORD PROCESSING
130 DIM A1$(14)
140 GOSUB 4250
150 CALL CLEAR
160 DISPLAY AT(10,7):"WORD PROCESSING"
170 DISPLAY AT(11,3):"- ENTRY/UPDATE PROGRAM
    -"
180 INPUT "FILENAME -DSK1.":F$
190 DISPLAY AT(6,8)ERASE ALL:"SELECT OPTION"
200 DISPLAY AT(9,6):"N - NEW DATA FILE"
210 DISPLAY AT(11,6):"A - ADD TO END OF"
220 DISPLAY AT(12,10):"EXISTING FILE"
230 DISPLAY AT(14,6):"C - CHANGE EXISTING"
240 DISPLAY AT(15,10):"FILE"
250 DISPLAY AT(17,6):"P - PRINT FILE"
260 DISPLAY AT(20,10):"CHOICE"
```

```
270 ACCEPT AT(20,17)BEEP VALIDATE("NACP"):C$
280 IF LEN(C$)=0 THEN 260
290 OPEN #1:"DSK1."&F$,RELATIVE,INTERNAL,UPD
    ATE,FIXED 250
300 IF C$="P" THEN 2410
310 IF C$="N" THEN 320 ELSE 340
320 PRINT #1,REC 0:"EOF=";0;1
330 NXTREC=1 :: GOTO 400
340 RECNO=0
350 INPUT #1,REC RECNO:A$,EOFREC,NXTREC
360 IF C$="A" THEN 600 ELSE 670
370 REM
380 REM NEW ROUTINE
390 REM
400 CALL CLEAR
410 LINPUT A$
420 LSTREC=CURREC
430 CURREC=NXTREC
440 NXTREC=NXTREC+1
450 IF SEG$(A$,1,3)="/E/" OR SEG$(A$,1,3)="/
    e/" THEN PRINT #1,REC CURREC:A$;LSTREC;N
    XTREC :: EOFREC=CURREC :: GOTO 490
460 PRINT #1,REC CURREC:A$;LSTREC,NXTREC
470 GOTO 410
480 REM UPDATE HEADER
490 RECNO=0
500 INPUT #1,REC RECNO:A$,HRECNO,NXTREC
510 PRINT #1,REC RECNO:A$,EOFREC,NXTREC
520 DISPLAY AT(12,1)ERASE ALL:"DO YOU WANT T
    O PRINT THE"
530 DISPLAY AT(13,1):"REPORT NOW - Y/N"
540 ACCEPT AT(13,18)BEEP SIZE(1)VALIDATE("YN
    yn"):P$
550 IF P$="Y" OR P$="y" THEN 2410
560 CLOSE #1
570 END
580 REM
590 REM ADD ROUTINE
600 REM
610 INPUT #1,REC EOFREC:A$,CURREC,NXTREC
620 INPUT #1,REC CURREC:A$,LSTREC,DUMMY
630 CALL CLEAR
640 DISPLAY AT(10,1):"LAST RECORD ON FILE IS
    :"
650 DISPLAY AT(12,1):A$
660 LINPUT A$ :: LSTREC=CURREC :: CURREC=EOF
    REC :: GOTO 450
670 REM
680 REM UPDATE ROUTINE
690 REM
```

```
700 CALL CLEAR
710 RECNO=NXTREC
720 INPUT #1,REC RECNO:A$,LSTREC,NXTREC
730 DISPLAY AT(2,1):"CURRENT LINE"
740 FOR I=4 TO 13
750 DISPLAY AT(I,1):" "
760 NEXT I
770 DISPLAY AT(4,1):A$
780 DISPLAY AT(14,1):"SELECT CHOICE:"
790 DISPLAY AT(16,1):"1=NEXT LINE{4 SPACES}5
    =ADD BEFORE"
800 DISPLAY AT(17,1):"2=LAST LINE{4 SPACES}6
    =ADD AFTER"
810 DISPLAY AT(18,1):"3=FWD X LINES  7=CHANG
    E"
820 DISPLAY AT(19,1):"4=BWK X LINES  8=DELET
    E"
830 DISPLAY AT(20,16):"9=QUIT"
840 DISPLAY AT(22,1):"YOUR CHOICE:"
850 ACCEPT AT(22,13)BEEP VALIDATE("123456789
    "):C$
860 DISPLAY AT(24,1):" "
870 IF LEN(C$)=0 THEN 840
880 C=VAL(C$)
890 ON C GOTO 900,1000,1100,1100,1180,1180,1
    540,2180,490
900 REM
910 REM DISPLAY NEXT
920 REM
930 NBRFWD=1
940 FOR I=1 TO NBRFWD
950 IF NXTREC=EOFREC THEN DISPLAY AT(24,1):"
    LINE DOES NOT EXIST" :: DISPLAY AT(2,1):
    "LAST LINE OF TEXT" :: GOTO 740
960 RECNO=NXTREC
970 INPUT #1,REC RECNO:A$,LSTREC,NXTREC
980 NEXT I
990 GOTO 730
1000 REM
1010 REM DISPLAY LAST
1020 REM
1030 NBRBACK=1
1040 FOR I=1 TO NBRBACK
1050 IF LSTREC=0 THEN DISPLAY AT(24,1):"LINE
      DOES NOT EXIST" :: DISPLAY AT(2,1):"FI
     RST LINE OF TEXT" :: GOTO 740
1060 RECNO=LSTREC
1070 INPUT #1,REC RECNO:A$,LSTREC,NXTREC
1080 NEXT I
1090 GOTO 730
```

```
1100 REM
1110 REM FOWARD/BACK X LINES
1120 REM
1130 DISPLAY AT(22,16):"NBR LINES"
1140 ACCEPT AT(22,26)BEEP:NBRLNS
1150 IF C=3 THEN NBRFWD=NBRLNS :: GOTO 940
1160 NBRBACK=NBRLNS
1170 GOTO 1040
1180 REM
1190 REM ADD BEFORE/AFTER
1200 REM
1210 CALL CLEAR
1220 IF C=6 THEN PRINT "ADD NEW LINE AFTER:"
      ELSE PRINT "ADD NEW LINE BEFORE:"
1230 PRINT
1240 PRINT A$
1250 PRINT
1260 PRINT "ENTER NEW LINE" :: ::
1270 LINPUT AN$
1280 HREC=RECNO
1290 HLST=LSTREC
1300 HNXT=NXTREC
1310 INPUT #1,REC EOFREC:A$,LSTREC,ADDREC
1320 HADD=ADDREC
1330 PRINT #1,REC EOFREC:A$,LSTREC,ADDREC+1
1340 IF C=6 OR RETSW=1 THEN 1450
1350 REM
1360 REM ADD BEFORE
1370 REM
1380 PRINT #1,REC HADD:AN$,HLST,HREC
1390 INPUT #1,REC HLST:A$,LSTREC,NXTREC
1400 PRINT #1,REC HLST:A$,LSTREC,HADD
1410 INPUT #1,REC HREC:A$,LSTREC,NXTREC
1420 PRINT #1,REC HREC:A$,HADD,NXTREC
1430 NXTREC=HADD
1440 IF RETSW=1 THEN 2010 ELSE GOTO 700
1450 REM
1460 REM ADD AFTER
1470 REM
1480 PRINT #1,REC HADD:AN$,HREC,HNXT
1490 INPUT #1,REC HREC:A$,LSTREC,NXTREC
1500 PRINT #1,REC HREC:A$,LSTREC,HADD
1510 INPUT #1,REC HNXT:A$,LSTREC,NXTREC
1520 PRINT #1,REC HNXT:A$,HADD,NXTREC
1530 GOTO 1430
1540 REM
1550 REM CHANGE
1560 REM
1570 CALL CLEAR
1580 LENA1=INT(LEN(A$)/14)+1
```

```
1590 FOR I=1 TO 13
1600 A1$(I)=SEG$(A$,LENA1*(I-1)+1,LENA1)&"
     {,}"
1610 DISPLAY AT(I,1):"["
1620 DISPLAY AT(I,2):A1$(I)
1630 DISPLAY AT(I,28):"]"
1640 NEXT I
1650 IF LEN(A$)<=13*LENA1 THEN A1$(14)="{,}"
      :: GOTO 1670
1660 A1$(14)=SEG$(A$,LENA1*13+1,LEN(A$)-LENA
     1*13)&"{,}"
1670 DISPLAY AT(14,1):"["
1680 DISPLAY AT(14,2):A1$(14)
1690 DISPLAY AT(14,28):"]"
1700 LENA=0
1710 FOR I=1 TO 14
1720 ACCEPT AT(I,2)BEEP SIZE(-26):A1$(I)
1730 IF LEN(A1$(I))=0 THEN A1$(I)="{,}" ELSE
      IF SEG$(A1$(I),LEN(A1$(I)),1)<>"{,}" T
     HEN A1$(I)=A1$(I)&"{,}"
1740 LENA=LENA+(LEN(A1$(I))-1)
1750 NEXT I
1760 IF LENA>225 THEN 1820
1770 A$=""
1780 FOR I=1 TO 14
1790 A$=A$&SEG$(A1$(I),1,POS(A1$(I),"{,}",1)
     -1)
1800 NEXT I
1810 GOTO 2050
1820 DISPLAY AT(16,1):"NEW LINE TOO LONG"
1830 DISPLAY AT(18,1):"SELECT CHOICE:"
1840 DISPLAY AT(19,1):"1=UPDATE/CREATE NEW L
     INE"
1850 DISPLAY AT(20,1):"2=RE-UPDATE"
1860 DISPLAY AT(22,1):"YOUR CHOICE"
1870 ACCEPT AT(22,13)BEEP VALIDATE("12"):C$
1880 IF LEN(C$)=0 THEN 1860
1890 IF C$="2" THEN 1540
1900 A2$="" :: A3$=""
1910 FOR I=1 TO 7
1920 A2$=A2$&SEG$(A1$(I),1,POS(A1$(I),"{,}",
     1)-1)
1930 A3$=A3$&SEG$(A1$(I+7),1,POS(A1$(I+7),"
     {,}",1)-1)
1940 NEXT I
1950 RETSW=1
1960 HLDCUR=RECNO
1970 A$=A2$
1980 PRINT #1,REC RECNO:A$,LSTREC,NXTREC
1990 AN$=A3$
```

```
2000 GOTO 1280
2010 INPUT #1,REC HLDCUR:A$,LSTREC,NXTREC
2020 RETSW=0
2030 CALL CLEAR
2040 GOTO 720
2050 CALL CLEAR
2060 DISPLAY AT(2,1):"CURRENT LINE"
2070 DISPLAY AT(4,1):A$
2080 DISPLAY AT(14,1):"SELECT CHOICE:"
2090 DISPLAY AT(16,1):"1=MORE UPDATES"
2100 DISPLAY AT(17,1):"2=UPDATE AS IS"
2110 DISPLAY AT(18,1):"3=EXIT-NO UPDATE"
2120 DISPLAY AT(22,1):"YOUR CHOICE:"
2130 ACCEPT AT(22,13)BEEP VALIDATE("123"):C$
2140 IF LEN(C$)=0 THEN 2080
2150 ON VAL(C$)GOTO 1540,2160,720
2160 PRINT #1,REC RECNO:A$,LSTREC,NXTREC
2170 GOTO 720
2180 REM
2190 REM DELETE LINE
2200 REM
2210 DISPLAY AT(24,1):"CONFIRM DELETE - Y/N"
2220 ACCEPT AT(24,22)BEEP VALIDATE("YyNn"):D
     $
2230 IF D$="N" OR D$="n" THEN DISPLAY AT(24,
     1):"LINE NOT DELETED" :: GOTO 780
2240 HLST=LSTREC
2250 HNXT=NXTREC
2260 INPUT #1,REC HLST:A$,LSTREC,NXTREC
2270 PRINT #1,REC HLST:A$,LSTREC,HNXT
2280 INPUT #1,REC HNXT:A$,LSTREC,NXTREC
2290 PRINT #1,REC HNXT:A$,HLST,NXTREC
2300 LSTREC=HLST
2310 NXTREC=HNXT
2320 DISPLAY AT(24,1):" "
2330 IF LSTREC>0 THEN GOTO 1000
2340 IF NXTREC=EOFREC THEN 2350 ELSE 900
2350 CALL CLEAR
2360 PRINT "TEXT NO LONGER EXISTS"
2370 PRINT
2380 CLOSE #1
2390 END
2400 REM
2410 REM WORD PROCESSING
2420 REM PRINT PROGRAM
2430 REM
2440 CALL CLEAR
2450 REM
2460 REM SET-UP DEFAULTS
2470 REM
```

```
2480 TM=6
2490 BM=6
2500 LM=1
2510 PL=66
2520 LC=0
2530 LPP=PL-BM
2540 MAXWID=68
2550 DISPLAY AT(10,7):"WORD PROCESSING"
2560 DISPLAY AT(11,6):"- PRINT PROGRAM -"
2570 DISPLAY AT(18,1):"FILENAME - DSK1.";F$
2580 DISPLAY AT(20,1):"PRINT MODE - D/F"
2590 DISPLAY AT(22,1):"SPACING - S/D"
2600 DISPLAY AT(24,1):"PAGE NUMBER (Y/N)"
2610 ACCEPT AT(20,20)SIZE(1)BEEP VALIDATE("D
     Fdf"):M$
2620 IF LEN(M$)=0 THEN 2610
2630 IF M$="d" THEN M$="D"
2640 IF M$="f" THEN M$="F"
2650 ACCEPT AT(22,20)SIZE(1)BEEP VALIDATE("S
     Dsd"):SPG$
2660 IF LEN(SPG$)=0 THEN 2650
2670 IF SPG$="s" THEN SPG$="S"
2680 IF SPG$="d" THEN SPG$="D"
2690 ACCEPT AT(24,20)SIZE(1)BEEP VALIDATE("Y
     Nyn"):PGNO$
2700 IF LEN(PGNO$)=0 THEN 2690
2710 IF PGNO$="y" THEN PGNO$="Y"
2720 RESTORE #1
2730 OPEN #2:"PIO"
2740 GOSUB 3800
2750 REM
2760 REM READ INITIAL RECORD
2770 REM
2780 INPUT #1:A$,LSTREC,NXTREC
2790 REM
2800 REM READ INPUT FILE
2810 REM
2820 INPUT #1,REC NXTREC:A$,LSTREC,NXTREC
2830 IF M$="D" THEN 2850
2840 IF SEG$(A$,1,3)="/N/" OR SEG$(A$,1,3)="
     /n/" THEN PRINT #2 :: LC=LC+1 :: GOSUB
     3900 :: GOTO 2800
2850 IF SEG$(A$,1,3)="/E/" OR SEG$(A$,1,3)="
     /e/" THEN 2940
2860 IF M$="F" THEN 2980
2870 REM
2880 REM PRINT DRAFT
2890 REM
2900 PRINT #2:A$
2910 LC=LC+1
```

```
2920 IF LC=LPP THEN GOSUB 3900
2930 GOTO 2800
2940 PRINT #2
2950 GOSUB 3910
2960 CLOSE #1 :: CLOSE #2
2970 END
2980 REM
2990 REM PRINT FINAL
3000 REM
3010 IF LEN(A$)=0 THEN 3690
3020 IF SEG$(A$,LEN(A$),1)="." THEN A$=A$&"
     "
3030 IF SEG$(A$,1,3)="/P/" OR SEG$(A$,1,3)="
     /p/" THEN 3140
3040 IF SEG$(A$,1,3)="/C/" OR SEG$(A$,1,3)="
     /c/" THEN 3540
3050 IF PC+LEN(A$)<=MAXWID THEN 3110
3060 NPOS=MAXWID-PC
3070 STRT=1
3080 INIT=NPOS+1
3090 IF INIT<1 THEN INIT=1
3100 GOTO 3300
3110 PRINT #2:A$;
3120 PC=PC+LEN(A$)
3130 GOTO 2800
3140 REM
3150 REM **NEW PARAGRAPH**
3160 REM
3170 IF PC>LM THEN PRINT #2 :: LC=LC+1 :: PR
     INT #2:RPT$(" ",LM);
3180 IF SPG$="D" AND PC>LM THEN PRINT #2 ::
     LC=LC+1 :: PRINT #2:RPT$(" ",LM);
3190 PC=LM
3200 IF LC>=LPP THEN GOSUB 3900
3210 PRINT #2:"{5 SPACES}";
3220 IF LEN(A$)+LM+2>MAXWID THEN 3260
3230 PRINT #2:SEG$(A$,4,LEN(A$)-3);
3240 PC=LEN(A$)+2+LM
3250 GOTO 2800
3260 NPOS=MAXWID-5-LM
3270 STRT=4
3280 INIT=NPOS+4
3290 REM
3300 REM **PRINT PARTIAL LINE**
3310 REM
3320 IF PC>MAXWID THEN 3380
3330 PRINT #2:SEG$(A$,STRT,NPOS);
3340 PC=MAXWID
3350 REM
3360 REM
```

```
3370 REM
3380 FOR I=INIT TO LEN(A$)
3390 PC=PC+1
3400 A2$=SEG$(A$,I,1)
3410 IF PC=1+LM AND A2$=" " THEN PC=LM :: GO
     TO 3440
3420 IF A2$=" " THEN 3460
3430 PRINT #2:A2$;
3440 NEXT I
3450 GOTO 2800
3460 INIT=I :: PRINT #2 :: LC=LC+1 :: PRINT
     #2:RPT$(" ",LM);
3470 IF SPG$="D" THEN PRINT #2 :: LC=LC+1 ::
      PRINT #2:RPT$(" ",LM);
3480 IF LC>=LPP THEN GOSUB 3900
3490 PC=LM
3500 IF INIT=LEN(A$)THEN 2800
3510 IF SEG$(A$,INIT,1)=" " THEN INIT=INIT+1
      :: GOTO 3500
3520 A$=SEG$(A$,INIT,LEN(A$)-INIT+1)
3530 GOTO 3050
3540 REM
3550 REM CENTERING ROUTINE
3560 REM
3570 IF PC>LM THEN PRINT #2 :: LC=LC+1 :: PR
     INT #2:RPT$(" ",LM);
3580 IF PC>LM AND SPG$="D" THEN PRINT #2 ::
     LC=LC+1 :: PRINT #2:RPT$(" ",LM);
3590 PC=LM
3600 IF LC>=LPP THEN GOSUB 3900
3610 CLEN=LEN(A$)-3
3620 SP=INT((MAXWID-LM-CLEN)/2)
3630 PRINT #2:RPT$(" ",SP+LM);
3640 PRINT #2:SEG$(A$,4,LEN(A$))
3650 LC=LC+1 :: PRINT #2:RPT$(" ",LM);
3660 IF SPG$="D" THEN PRINT #2 :: LC=LC+1 ::
      PRINT #2:RPT$(" ",LM);
3670 IF LC>=LPP THEN GOSUB 3900
3680 GOTO 2800
3690 REM
3700 REM PRINT BLANK LINE
3710 REM
3720 IF PC=LM THEN 3750
3730 PRINT #2 :: LC=LC+1
3740 IF SPG$="D" THEN PRINT #2 :: LC=LC+1
3750 PRINT #2 :: LC=LC+1 :: PRINT #2:RPT$("
     ",LM);
3760 IF SPG$="D" THEN PRINT #2 :: LC=LC+1 ::
      PRINT #2:RPT$(" ",LM);
```

```
3770 IF LC>=LPP THEN GOSUB 3900
3780 PC=LM
3790 GOTO 2800
3800 REM
3810 REM PRINT TOP MARGIN
3820 REM
3830 FOR LC=1 TO TM
3840 PRINT #2
3850 NEXT LC
3860 LC=TM
3870 PRINT #2:RPT$(" ",LM);
3880 PC=LM
3890 RETURN
3900 REM
3910 REM PRINT BOTTOM & TOP MARGINS
3920 REM
3930 FOR LCT=LC+1 TO PL
3940 IF PGNO$="Y" AND LCT=PL-3 THEN PGNO=PGN
     O+1 :: PRINT #2:RPT$(" ",38);"PAGE ";PG
     NO ELSE PRINT #2
3950 NEXT LCT
3960 GOSUB 3800
3970 RETURN
3980 REM RE-DEFINE LOWER CASE CHARACTERS
3990 DATA 0000003804 3C443C
4000 DATA 0040407844444478
4010 DATA 0000003C4040403C
4020 DATA 0004043C4444443C
4030 DATA 000000384478403C
4040 DATA 0018242020702020
4050 DATA 0000304838082810
4060 DATA 0040404078444444
4070 DATA 0010001010101010
4080 DATA 0004000404042418
4090 DATA 0040485060504848
4100 DATA 0010101010101010
4110 DATA 0000002854444444
4120 DATA 0000007844444444
4130 DATA 0000003844444438
4140 DATA 0000704870404040
4150 DATA 00001C241C040404
4160 DATA 0000005864404040
4170 DATA 0000003C40380478
4180 DATA 0000207020202418
4190 DATA 0000004444444438
4200 DATA 0000004444442810
4210 DATA 0000004444546C44
4220 DATA 0000004428102844
4230 DATA 0000442418102040
4240 DATA 0000007C0810207C
```

```
4250 FOR I=97 TO 122
4260 READ A$
4270 CALL CHAR(I,A$)
4280 NEXT I
4290 RETURN
```

4
Recreation

4

Trap

Larry Michalewicz

Each player must avoid the walls while trying to force his opponent to collide with him or a wall. It gets tricky. A two-player game, joysticks required.

"Trap," written for the TI-99/4 and 4A, runs in TI or Extended BASIC. The object is to force your opponent to collide with a wall while you avoid hitting any walls yourself. If you cause your opponent to crash into your wall, his own wall, or a boundary wall, you receive a point. The first player to get five points wins the game.

Program Description

The playing field for the game is set up in lines 200–280. Lines 250 and 260 draw the top and bottom barriers, and lines 270 and 280 draw the left and right walls.

The variables for player movement are initialized in lines 290–380. The beginning coordinates for player 1 are C1 and D1; for player 2, C2 and D2.

Lines 410 and 470 examine input from the two joysticks. If a joystick has not been moved or has been moved in a diagonal direction, the player will continue to move in the direction he or she was last going. The CALL GCHAR statements in lines 520 and 570 determine the ASCII value of the character in the next screen location. If this value is anything but a 32 (which is a space), then I1 or I2 is assigned the value of 1 depending on which player has collided.

Line 620 checks for a collision between players and walls. If I1 (but not I2) is equal to 1, meaning the player on the left side has crashed, the right side wins and is awarded a point. Likewise, if I2 (but not I1) is 1, the left side wins and receives a point. If both players collide (I1 and I2 = 1) with a wall simultaneously, each player is awarded a point.

When either player gets five points, the game is over. Lines 830–840 then prompt for another game.

Trap

```
100 P=0
110 Q=0
120 CALL CHAR(120,"")
130 CALL CHAR(135,"")
140 CALL CHAR(136,"")
150 CALL CLEAR
160 CALL SCREEN(3)
170 CALL COLOR(13,1,7)
180 CALL COLOR(14,1,5)
190 REM SET UP PLAYING FIELD
200 PRINT "PLAYER #1 ";P,"PLAYER #2 ";Q
210 FOR O=1 TO 22
220 PRINT
230 NEXT O
240 CALL COLOR(12,2,2)
250 CALL HCHAR(2,2,120,30)
260 CALL HCHAR(24,2,120,30)
270 CALL VCHAR(2,2,120,23)
280 CALL VCHAR(2,31,120,23)
290 C1=12
300 C2=12
310 D1=4
320 D2=28
330 R1=0
340 S1=1
350 R2=0
360 S2=-1
370 I1=0
380 I2=0
390 REM MAIN LOOP(MOVEMENT)
400 CALL JOYST(1,B1,A1)
410 IF ABS(A1)-ABS(B1)=0 THEN 440
420 R1=A1/4
430 S1=B1/4
440 C1=C1-R1
450 D1=D1+S1
460 CALL JOYST(2,B2,A2)
470 IF ABS(A2)-ABS(B2)=0 THEN 500
480 R2=A2/4
490 S2=B2/4
500 C2=C2-R2
510 D2=D2+S2
520 CALL GCHAR(C1,D1,G)
530 IF G=32 THEN 550
540 I1=1
550 CALL VCHAR(C1,D1,135)
560 CALL SOUND(1,262,0)
570 CALL GCHAR(C2,D2,G)
```

```
580 IF G=32 THEN 600
590 I2=1
600 CALL VCHAR(C2,D2,136)
610 CALL SOUND(1,290,0)
620 IF I1+I2=0 THEN 400
630 IF ((I1=1*I2=1))+((I2=1)*(D1+D2=32))THEN
     710
640 IF I1=1 THEN 680
650 H$="LEFT SIDE WINS"
660 P=P+1
670 GOTO 740
680 H$="RIGHT SIDE WINS"
690 Q=Q+1
700 GOTO 740
710 H$="IT'S A TIE"
720 P=P+1
730 Q=Q+1
740 PRINT H$
750 FOR I=1 TO 200
760 NEXT I
770 FOR H=1 TO 23
780 PRINT
790 NEXT H
800 IF (P<>5)*(Q<>5)THEN 150
810 REM PLAY AGAIN?
820 PRINT "PLAYER 1";P,"PLAYER 2";Q
830 PRINT "DO YOU WANT TO PLAY AGAIN (Y/N)?"
840 CALL KEY(0,KEY,ST)
850 IF ST=0 THEN 840
860 IF (KEY=89)+(KEY=121)THEN 100
870 END
```

Duck Leader

Douglas E. Smith and Douglas W. Smith

This is no time to be feather-brained or daffy. There are hunters lurking in the maze of reeds ahead, and if you make a wrong turn, you and your friends will be duck soup. Two skill levels, ten difficulty ratings.

This game will challenge your skill and memory. Your assignment, as the leader of a squadron of 30 ducks, is to direct them through a series of marsh mazes to the safety of a duck sanctuary.

You must swim through five different mazes, each of which has invisible reed patches and hidden hunters. The reeds will send you back to the beginning of the maze. If you find a hunter, you will lose a duck. Save as many ducks as possible for a high score. Lose your squadron and it's all over.

Favorite Duck

After typing in the program, it's a good idea to list the program, check for errors, and then save a copy to tape or disk before running the program.

When you start the game, the title graphics will appear, followed by several questions.

1. LEVEL=?(1=HELP/ 2=NO HELP)

Enter 1 until you have gained confidence in your memory and problem-solving ability. With level 1 you may use the H key to quickly view the location of the reeds and hunters in the marsh (up to five times during one game).

Level 2 will double your possible score, but you cannot use the H key for help.

2. DIFFICULTY=(1−10)

For your first game, enter 1. After some practice try the other difficulty ratings. Ratings 1–4 are easy, 5–7 are hard, and 8–10 are very challenging. The difficulty rating determines the complexity of the maze.

3. INSTRUCTIONS ? (Y/N)

Enter Y to read the instructions before the game begins. This screen briefly describes the game and the functions of the appropriate keys used for the game. It also shows the graphic characters used for the reeds, hunters, and the marsh exit.

4. FIRST NAME OF THE LEAD DUCK?

Type in your first name or the name of your favorite duck, and press ENTER. The game will then begin.

If you entered Y for instructions, they will appear first. Hit the 1 key to begin play.

At first the outline of the marsh appears, and then the positions of the reeds and hunters are indicated. You have 15 seconds to study the locations before the reeds and hunters become invisible.

Successful Maneuvers

The duck on the left side of the marsh represents your squadron. Move the duck by using the arrow keys (E, S, D, and X). You do not have to press ENTER or use the FCTN key.

If you hit the sides of the marsh or the invisible reed, your ducks will bounce back to the starting position, but you do not lose any ducks.

Meeting a hunter will result in gunfire and the loss of a duck, with the survivors returning to the start again.

The positions of the reeds and hunters do not change until you reach the exit to the next marsh.

Your goal is to maneuver your squadron to the right side of the marsh and out the exit to the next marsh. Once you have passed through the five marshes to safety, the program will congratulate you, show you the remaining ducks, and print your score.

The highest possible score is 6000 and can be achieved only at level 2 with the difficulty rating 10. Nobody has achieved this score to date.

The marsh border color changes to red if the squadron is depleted to ten or fewer ducks. Losing all the squadron will put you in Duck Soup, and give you a zero score.

After the score is printed, the program will ask PLAY AGAIN? (Y/N). Enter Y to play again at the last selected level and difficulty. Enter N to choose a different level and difficulty.

The H key may be used if you selected level 1. Pushing the H key while playing the game will give you a quick look at the marsh. Using this key does not change the position of the duck. You are limited to only five helps per game. Remember the highest score using level 1 will be half that of level 2.

On occasion, the program will generate a maze which is impossible to cross. (Ducks don't always have it easy, do they?) Press the N key and you will move to a new marsh. The change in marshes will cost the squadron five ducks, so use the N key only if there is no way out.

Some Noteworthy Routines

The "Duck Leader" program employs several very useful TI BASIC routines. Creation of the maze is accomplished using the RND function to place the reeds and hunters (lines 620–670). The CALL GCHAR in line 800 tests the randomly determined position for characters already present on the marsh.

Line 800 checks to see if an empty space is present and, if not, calls for a new set of coordinates to be generated.

The CALL KEY routine (line 890) is used to move the duck through the marsh. Keys 68, 69, 83, and 88 determine the direction of the move. Once again the CALL GCHAR is used to test for an empty space. If found, the duck is printed in that position. If a reed or hunter is found, lines 1250–1260 execute the proper action.

Several loops and counters pause the program and keep track of the ducks. Lines 690–720 give you time to view the maze before the characters become invisible. The H key sends the program to lines 1140–1220, making the maze visible. The ducks left are counted in lines 980 and 1310. The number of marshes traversed is counted in line 1380. The score is calculated in line 1550 and the ducks saved in line 1540.

Changing the Difficulty

You may wish to make the game less difficult by making one or more of the following changes:

1. Set the final value in the FOR-NEXT loop in line 690 to a higher value to increase the length of time you have to view the maze.

2. Increase the final value for X in the FOR-NEXT loop beginning in line 1170 to give you a longer look when you use the H key.
3. Change the 5 in line 1150 to a greater number to increase the number of times you can use the H key.

Program Summary

Lines	
120–170	Reset random generator, define graphic characters and colors.
180–320	Print title graphics and questions.
330–410	Duck animation GOSUB. Routine for title and game end.
420–500	Instructions.
510–610	Marsh and borderline.
620–670	Print reeds and hunters.
680–720	Allow view of the maze.
730	Make the reeds and hunters invisible.
740	Transfer control of the program to the call key routine.
750–820	Subroutine which randomly selects the positions for reeds and hunters.
830–980	Use the call key routine to read the keyboard and to branch the program for the desired action.
910	Check for the H key.
920	Check for the N key.
980–1030	Reset game for a new marsh.
980	Add five to total ducks lost (DL).
1040–1050	Sound for hitting reeds.
1060–1130	Reset value for position of the duck.
1140–1220	Reveal maze when H key is used.
1140	Check for level (1) input.
1150	Check for HELP limit.
1190	Count H key use.
1230–1270	Check the duck position for contact with reeds or hunters or the Exit.
1280–1360	Print gunfire graphic, call sound, and increase DL by one.
1320	Check for DL=20 and change border color if true.
1370–1410	Return program for creation of a new marsh, and count marshes completed.
1420–1460	Screen color change routine used to signify the beginning and end of the game and of the completion of a marsh.
1470–1710	Print end-of-game message and play again prompt.
1540	Calculate ducks saved (DS).
1550	Calculate score.
1610	Print saved ducks.
1670–1690	Set DL, HELP, and MARSH to 0.

1720–1800 Subroutine which defines the graphic characters.

128,129	Duck.
136	Border.
137	Exit.
112	Reeds.
113	Hunter.
120	Gun Shot.

Duck Leader

```
120 RANDOMIZE
130 GOSUB 1720
140 CALL COLOR(13,2,1)
150 CALL COLOR(14,5,16)
160 CALL COLOR(11,13,1)
170 CALL COLOR(12,10,1)
180 CALL CLEAR
190 CALL SCREEN(12)
200 FOR X=1 TO 19
210 PRINT "  DUCK LEADER  DUCK LEADER"
220 NEXT X
230 PRINT
240 GOSUB 330
250 GOSUB 1420
260 INPUT "LEVEL=?(1=HELP/2=NØ HELP)":LEVEL
270 INPUT "DIFFICULTY=?(1-10)":DIF
280 IF (DIF<1)+(DIF>10)THEN 270
290 INPUT "INSTRUCTIONS?(Y/N)":INS$
300 INPUT "FIRST NAME OF LEAD DUCK?":NAME$
310 IF INS$="Y" THEN 420
320 GOTO 520
330 FOR Y=12 TO 14 STEP 2
340 FOR X=1 TO 32
350 CALL HCHAR(Y,X,129)
360 CALL SOUND(25,-5,15)
370 CALL HCHAR(Y,X,128)
380 CALL HCHAR(Y,X,32)
390 NEXT X
400 NEXT Y
410 RETURN
420 CALL CLEAR
430 PRINT NAME$: :
440 PRINT "YOU ARE THE LEADER OF A": :"SQUAD
    RON OF THIRTY DUCKS.": :"PADDLE THROUGH
    FIVE MARSHES": :
450 PRINT "TO SAFETY!!": :"USE ARROW KEYS TO
     MOVE": :"H-KEY FOR HELP(ONLY FIVE)": :"
    N-KEY=NEW MARSH(-5 DUCKS)": :
```

```
460 PRINT "WATCH OUT FOR REEDS..": :"AND HUN
    TERS!!": :"REEDS":"HUNTERS":"EXIT"
470 CALL HCHAR(21,12,112)
480 CALL HCHAR(22,12,113)
490 CALL HCHAR(23,12,137)
500 INPUT "ENTER 1 TO START":SRT
510 GOTO 520
520 REM MARSH
530 CALL SCREEN(15)
540 CALL CLEAR
550 PRINT TAB(10);"MARSH #";MARSH+1
560 CALL HCHAR(2,3,136,28)
570 CALL HCHAR(22,3,136,28)
580 CALL VCHAR(2,3,136,20)
590 CALL VCHAR(2,30,136,20)
600 CALL VCHAR(6,7,136,12)
610 CALL VCHAR(9,30,137,7)
620 FOR X=1 TO (DIF*5)
630 GOSUB 750
640 CALL HCHAR(ROW,COL,112)
650 GOSUB 750
660 CALL HCHAR(ROW,COL,113)
670 NEXT X
680 CALL HCHAR(12,5,128)
690 FOR X=1 TO 50
700 CALL COLOR(13,9,1)
710 CALL COLOR(13,2,1)
720 NEXT X
730 CALL SCREEN(13)
740 GOTO 830
750 REM RAN ROW+COL
760 ROW=INT(22*RND)
770 IF ROW<3 THEN 760
780 COL=INT(30*RND)
790 IF COL<4 THEN 780
800 CALL GCHAR(ROW,COL,GRC)
810 IF GRC<>32 THEN 760
820 RETURN
830 REM MOVE DUCK
840 R=12
850 C=5
860 CALL HCHAR(R,C,129)
870 CALL SOUND(25,-5,15)
880 CALL HCHAR(R,C,128)
890 CALL KEY(0,KEY,ST)
900 IF (KEY=68)+(KEY=69)+(KEY=72)+(KEY=78)+(
    KEY=83)+(KEY=88)THEN 910 ELSE 890
910 IF KEY=72 THEN 1140
920 IF KEY=78 THEN 980
930 CALL HCHAR(R,C,32)
```

```
940 IF KEY=68 THEN 1060
950 IF KEY=69 THEN 1080
960 IF KEY=83 THEN 1100
970 IF KEY=88 THEN 1120
980 DL=DL+5
990 IF DL>19 THEN 1000 ELSE 1040
1000 CALL COLOR(14,5,9)
1010 IF DL>=30 THEN 1020 ELSE 1040
1020 DL=30
1030 GOTO 1470
1040 CALL SOUND(100,500,0)
1050 GOTO 520
1060 C=C+1
1070 GOTO 1230
1080 R=R-1
1090 GOTO 1230
1100 C=C-1
1110 GOTO 1230
1120 R=R+1
1130 GOTO 1230
1140 IF LEVEL<>1 THEN 890
1150 IF HELP=5 THEN 1210
1160 CALL SCREEN(15)
1170 FOR X=1 TO 50
1180 NEXT X
1190 HELP=HELP+1
1200 CALL SCREEN(13)
1210 CALL SOUND(5,1000,1)
1220 GOTO 890
1230 CALL GCHAR(R,C,GR)
1240 IF GR=32 THEN 860
1250 IF (GR=136)+(GR=112)THEN 1350
1260 IF GR=113 THEN 1280
1270 IF GR=137 THEN 1370
1280 CALL HCHAR(R,C,120)
1290 CALL SOUND(1,200,1)
1300 CALL HCHAR(R,C,113)
1310 DL=DL+1
1320 IF DL=20 THEN 1330 ELSE 1340
1330 CALL COLOR(14,5,9)
1340 IF DL=30 THEN 1470 ELSE 840
1350 CALL SOUND(50,-1,10)
1360 GOTO 840
1370 REM NEW MARSH
1380 MARSH=MARSH+1
1390 IF MARSH=5 THEN 1470
1400 GOSUB 1420
1410 GOTO 520
1420 FOR SC=4 TO 16
1430 CALL SCREEN(SC)
```

```
1440 CALL SOUND(1,3000,1)
1450 NEXT SC
1460 RETURN
1470 REM END OF GAME
1480 CALL CLEAR
1490 CN=INT((28-LEN(NAME$))/2)
1500 PRINT TAB(CN);NAME$: :
1510 IF DL=30 THEN 1640
1520 GOSUB 1420
1530 GOSUB 1420
1540 DS=30-DL
1550 SCORE=LEVEL*DIF*DS*10
1560 PRINT "CONGRATULATIONS YOU SAVED ": :TA
     B(10);DS;"DUCKS!": :TAB(5);"FOR A SCORE
      OF ";SCORE
1570 FOR X=1 TO 12
1580 PRINT
1590 NEXT X
1600 CL=INT((32-DS)/2)
1610 CALL HCHAR(6,CL,128,DS)
1620 GOSUB 330
1630 GOTO 1660
1640 REM DUCK SOUP
1650 PRINT "OH NO!! YOU'RE DUCK SOUP": :TAB(
     11);"SCORE=0": :
1660 INPUT "PLAY AGAIN?(Y/N)":PLAY$
1670 DL=0
1680 HELP=0
1690 MARSH=0
1700 CALL COLOR(14,5,16)
1710 IF PLAY$="Y" THEN 520 ELSE 1810
1720 REM CALL CHAR
1730 CALL CHAR(128,"00040B04FE7C8800")
1740 CALL CHAR(129,"00081608FE7C4400")
1750 CALL CHAR(136,"00AAAAAAAAAAAAAA")
1760 CALL CHAR(137,"00080C7E0C080000")
1770 CALL CHAR(112,"00AAAAAAAAAAAAAA")
1780 CALL CHAR(113,"0000609090677C64")
1790 CALL CHAR(120,"8142241818244281")
1800 RETURN
1810 END
```

Freeway 2000

John B. Dorff

Dare you cross the freeway of the future? You better have all your wits together, for this is one grueling highway. It will take all the cunning and speed you can muster to cross this ten-lane roadway. Requires Extended BASIC and joysticks. A Speech Synthesizer is optional.

If you've been trying to write games in BASIC, you have probably found out that it can be difficult to design fast-action games. Creating a game with many moving objects on the screen, moving in all directions, is next to impossible; BASIC is just too slow. Still, with TI's great graphic and sprite capabilities, there are ways to create fun and exciting games once you learn to work with BASIC's limitations. Extended BASIC is the best way to create such a game.

"Freeway 2000" is just such a game. It takes advantage of TI's graphics and sprites. To save program space, there are no REM statements or instructions for the game included in the program. For the same reason, and to increase speed, almost all the lines in the program are multistatement lines. Save the program after you have typed it in and before you run it.

Some speech has been added to enhance the game, so if you have a speech synthesizer, connect it before you play. There is nothing like a game that compliments you when you've made a good run, or chides you when you goof.

Crossing the Road

The object of the game is to get across the ten-lane highway, using a joystick to guide your runner, without getting hit. Each time you make it across successfully, the level of difficulty increases. At the start of the game, you score ten points for each lane passed, one thousand points for making it all the way. As the levels increase, the points per lane increase. You start off with six runners, gaining an extra one at six thousand point intervals. The game is for one or two players, so challenge a friend! Remember to have the ALPHA-LOCK key up when playing.

Here's a short explanation of the program:

Line #	Comment
10–30	Call up needed speech words, construct the suffix "s" and add it to certain words.
40–150	Title screen, definition of characters, and initialization of variables. Many variables are used to save space and to increase program speed. The more important ones are: L(1), L(2)—Player levels; E(1), E(2)—Score that must be reached to gain an extra runner; W(1), W(2)—Number of runners the players have left; Z(1), Z(2)—Players' scores; B(1), B(2)—Bonus points; P—Player number.
160	Input one- or two-player game.
170–220	Set up the playing screen.
230–320	Define sprites (cars and runners).
330	Randomly select the cars' speeds for each lane, dependent on the variable L(P)—player level.
340–430	Set cars in motion.
440–460	Main control loop.
470–560	Sorry, you got hit! These lines play appropriate sound effects and find the runner's position for scoring.
580–690	You made it across! These lines add the appropriate points and check to see if an extra runner should be awarded. Also increase the player's level.
700–720	Main control loop. (This is used when the runner is on top of the screen and must come down to cross the freeway.)
730–780	Same as 470–560.
790–800	Input to play again; reinitialize.
810–820	Input to continue the same game or start a new one; reinitialize.
830	This subroutine waits for you to press a key to answer.
840–850	This subroutine creates varied car sounds, dependent on the variable O.
860–920	These lines check to see if the game is over. If not, they subtract a runner and change the player number in a two-player game. They also award an extra runner at 6000 point intervals.
930	Data for constructing the suffix "s".

Freeway 2000

```
10 RANDOMIZE :: CALL SPGET("SET",S$):: CALL
   SPGET("GOOD",G$):: CALL SPGET("MOVE",M$):
   : CALL SPGET("WELL",W$)
```

```
20 CALL SPGET("WHAT",WH$):: FOR I=1 TO 29 ::
    READ A :: SS$=SS$&CHR$(A):: NEXT I
30 J=LEN(M$)-13 :: M$=SEG$(M$,1,2)&CHR$(J)&S
   EG$(M$,4,J):: J=LEN(WH$)-13 :: WH$=SEG$(W
   H$,1,2)&CHR$(J)&SEG$(WH$,4,J)
40 CALL CLEAR :: O=1800 :: GOSUB 840 :: DISP
   LAY AT(3,4):"****FREEWAY 2000****" :: L(1
   ),L(2)=5 :: V1=-2.5 :: V2=2.5 :: Q=33 ::
   Q1=138
50 CALL SOUND(80,570,5,356,5):: E(1),E(2)=60
   00 :: W(1),W(2)=5
60 SND1(0)=430 :: SND1(1)=514 :: SND1(2)=470
    :: SND1(3)=395 :: SND2(0)=300 :: SND2(1)
   =359 :: SND2(2)=390 :: SND2(3)=241
70 CALL CHAR(96,"003C"):: CALL CHAR(97,"0000
   003C"):: CALL CHAR(104,"000000FFFFFFFFFF"
   ):: CALL CHAR(99,"00000000003C")
80 CALL SOUND(150,570,5,356,5):: CALL CHAR(1
   05,""):: CALL CHAR(100,"000000000000003C"
   ):: CALL CHAR(130,"191AFEBC983C4484")
90 CALL CHAR(128,"0066C9DDDDDC966006693BBBB
   BB9366"):: CALL CHAR(132,"000CE43F3FE40C0
   0"):: P=1 :: Z(1),Z(2)=0 :: B(1),B(2)=100
   0 :: U=110
100 CALL SOUND(50,430,3,300,3):: DISPLAY AT(
    12,1):"DARE YOU CROSS THE FREEWAY" :: DI
    SPLAY AT(13,1):"OF THE FUTURE??????"
110 CALL SOUND(80,430,3,300,3):: O=2400 :: G
    OSUB 840 :: DISPLAY AT(13,1):"
    {18 SPACES}" :: DISPLAY AT(12,1):"ARE YOU
     INTREPID ENOUGH..."
120 O=1600 :: GOSUB 840 :: DISPLAY AT(13,1):
    "ADROIT ENOUGH...." :: CALL SOUND(150,47
    0,5,390,5):: FOR X=1 TO 150 :: NEXT X
130 CALL SOUND(150,470,5,390,5):: FOR X=1 TO
     200 :: NEXT X :: DISPLAY AT(14,1):"INSA
    NE ENOUGH TO TRY?!" :: O=2200 :: GOSUB 8
    40
140 CALL CLEAR :: DISPLAY AT(12,5):"GOOD LUC
    K,FRIEND...." :: FOR D=1 TO 100 :: NEXT
    D :: CALL SOUND(400,514,3,359,3)
150 FOR D=1 TO 300 :: NEXT D :: DISPLAY AT(1
    4,5):"YOU'LL NEED IT!!!!!!" :: O=1800 ::
     GOSUB 840
160 CALL CLEAR :: DISPLAY AT(12,8)BEEP SIZE(
    15):"1 OR 2 PLAYERS?" :: GOSUB 830 :: IF
     K=49 THEN A=1 ELSE A=0
170 CALL CLEAR :: CALL COLOR(2,2,13):: CALL
    COLOR(3,2,13):: CALL COLOR(4,2,13):: CAL
    L COLOR(8,2,13):: CALL COLOR(13,2,13)
```

```
180 CALL COLOR(10,13,16):: CALL COLOR(9,15,16):: CALL COLOR(7,2,13):: CALL COLOR(1,2,13):: CALL COLOR(5,2,13):: CALL COLOR(6,2,13)
190 CALL HCHAR(1,1,32,160):: CALL HCHAR(19,1,32,192):: CALL HCHAR(6,1,105,32):: CALL HCHAR(7,1,96,32):: CALL HCHAR(8,1,97,32)
200 CALL HCHAR(9,1,99,32):: CALL HCHAR(10,1,100,32):: CALL HCHAR(11,1,105,32):: CALL HCHAR(12,1,96,32):: CALL HCHAR(13,1,97,32)
210 CALL HCHAR(14,1,99,32):: CALL HCHAR(15,1,100,32):: CALL HCHAR(16,1,105,32):: CALL HCHAR(17,1,96,32):: CALL HCHAR(18,1,104,32)
220 DISPLAY AT(1,1)SIZE(8):"PLAYER 1" :: IF A=0 THEN DISPLAY AT(2,1)SIZE(8):"PLAYER 2"
230 CALL SPRITE(#1,128,2,41,12,#2,128,5,41,63,#4,128,3,41,187)
240 CALL SPRITE(#5,129,6,51,100,#6,129,7,51,200,#7,129,15,51,224)
250 CALL SPRITE(#8,129,2,61,60,#9,129,3,61,188)
260 CALL SPRITE(#10,128,14,71,90,#11,128,7,71,190,#12,128,9,71,220)
270 CALL SPRITE(#13,129,5,81,79,#14,129,13,81,109,#3,129,2,81,235)
280 CALL SPRITE(#15,128,7,91,123,#19,128,15,91,250,#16,128,4,101,30,#17,128,7,101,60,#18,128,6,101,179)
290 CALL SPRITE(#20,129,5,111,115,#21,129,2,111,145,#22,129,14,111,175)
300 CALL SPRITE(#23,128,15,121,84,#24,128,9,121,168,#25,128,7,121,235)
310 CALL SPRITE(#26,129,5,131,68,#27,129,2,131,184)
320 DISPLAY AT(22,1):"PLAYER";P :: CALL SPRITE(#28,130,2,160,127)
330 DISPLAY AT(5,10):"GET READY!!" :: CALL HCHAR(24,3,130,W(P)):: CALL SAY("GET",S$) :: FOR N=1 TO 10 :: S(N)=INT(RND*L(P))+10 :: NEXT N
340 HH=INT(RND*4):: CALL MOTION(#1,0,-S(1),#2,0,-S(1),#4,0,-S(1))
350 CALL MOTION(#5,0,S(2),#6,0,S(2),#7,0,S(2))
360 CALL MOTION(#8,0,S(3),#9,0,S(3))
370 CALL MOTION(#10,0,-S(4),#11,0,-S(4),#12,0,-S(4))
```

```
380 CALL MOTION(#13,Ø,S(5),#14,Ø,S(5),#3,Ø,S
    (5))
390 CALL MOTION(#15,Ø,-S(6),#19,Ø,-S(6))
400 CALL MOTION(#16,Ø,-S(7),#17,Ø,-S(7),#18,
    Ø,-S(7)):: DISPLAY AT(5,1Ø):"{4 SPACES}G
    O!!" :: CALL SAY("GO")
410 CALL MOTION(#2Ø,Ø,S(8),#21,Ø,S(8),#22,Ø,
    S(8))
420 CALL MOTION(#23,Ø,-S(9),#24,Ø,-S(9),#25,
    Ø,-S(9))
430 CALL MOTION(#26,Ø,S(1Ø),#27,Ø,S(1Ø)):: D
    ISPLAY AT(5,1Ø):"{12 SPACES}" :: IF O=1 T
    HEN 7ØØ ELSE O=Ø
440 CALL JOYST(P,X,Y):: CALL COINC(ALL,C)::
    IF C THEN 47Ø
450 CALL MOTION(#28,Y*V1,X*V2):: CALL POSITI
    ON(#28,R,V):: CALL JOYST(P,X,Y):: CALL C
    OINC(ALL,C):: IF C THEN 47Ø
460 CALL MOTION(#28,Y*V1,X*V2):: IF R>Q THEN
     44Ø ELSE 57Ø
470 CALL SOUND(8ØØ,SND1(HH),5,SND2(HH),5)::
    CALL MOTION(#28,Ø,Ø):: CALL PATTERN(#28,
    132)
480 CALL SOUND(1Ø,554,1):: CALL SOUND(1Ø,523
    ,2):: CALL SOUND(1Ø,494,3):: CALL SOUND(
    1Ø,466,4):: CALL SOUND(1Ø,44Ø,5)
490 CALL SOUND(1Ø,415,6):: CALL SOUND(1Ø,392
    ,7):: CALL SOUND(1Ø,37Ø,8):: CALL SOUND(
    1Ø,349,9):: CALL SOUND(1Ø,33Ø,1Ø)
500 CALL POSITION(#28,R,V):: ON HH+1 GOTO 51
    Ø,52Ø,53Ø,54Ø
510 CALL SAY(,WH$&SS$,"THAT"):: GOTO 55Ø
520 CALL SAY("SORRY"):: GOTO 55Ø
530 CALL SAY("OH",W$):: GOTO 55Ø
540 CALL SAY(,WH$&SS$,"THAT")
550 FOR D=133 TO 43 STEP -1Ø :: IF R<D THEN
    Z(P)=Z(P)+L(P)+L(P):: U=U+8 :: CALL SOUN
    D(5,U,Ø):: DISPLAY AT(P,9):Z(P)ELSE 87Ø
560 NEXT D :: GOTO 87Ø
570 CALL MOTION(#28,Ø,Ø):: IF R>17Ø OR R<2Ø
    THEN Z(P)=Ø :: DISPLAY AT(2Ø,12):"NO FAI
    R!" :: CALL SOUND(5ØØ,-3,Ø):: GOTO 86Ø
580 DISPLAY AT(2Ø,9)SIZE(13):"NICE RUNNING!"
     :: DISPLAY AT(21,6)SIZE(19):STR$(B(P));
    " BONUS POINTS!!" :: O=O+1
590 ON HH+1 GOTO 6ØØ,61Ø,62Ø,63Ø
600 CALL SAY("MEAN",M$&SS$):: GOTO 64Ø
610 CALL SAY("VARY",G$):: GOTO 64Ø
620 CALL SAY(,W$,"DONE"):: GOTO 64Ø
630 CALL SAY(,G$,"GOING")
```

```
640 FOR D=1 TO 10 :: U=U+8 :: Z(P)=Z(P)+L(P)
    +L(P):: CALL SOUND(5,U,0):: DISPLAY AT(P
    ,9):Z(P):: NEXT D :: U=110
650 L(P)=L(P)+1 :: Z(P)=Z(P)+B(P):: CALL SOU
    ND(50,SND1(HH),3,SND2(HH),3):: CALL SOUN
    D(100,SND1(HH),3,SND2(HH),3)
660 DISPLAY AT(P,9):Z(P):: IF Z(P)<E(P)THEN
    680
670 W(P)=W(P)+1 :: E(P)=E(P)+6000 :: CALL HC
    HAR(24,W(P)+2,130):: CALL SOUND(10,3
49,0):: CALL SOUND(10,523,0)
680 CALL HCHAR(20,9,32,50):: IF L(P)=11 OR L
    (P)=16 OR L(P)=21 THEN B(P)=B(P)+1000
690 GOTO 330
700 CALL JOYST(P,X,Y):: CALL COINC(ALL,C)::
    IF C THEN 730
710 CALL MOTION(#28,Y*V1,X*V2):: CALL POSITI
    ON(#28,R,V):: CALL JOYST(P,X,Y):: CALL C
    OINC(ALL,C):: IF C THEN 730
720 CALL MOTION(#28,Y*V1,X*V2):: IF R<Q1 THE
    N 700 ELSE 570
730 CALL SOUND(800,SND1(HH),5,SND2(HH),5)::
    CALL MOTION(#28,0,0)
740 CALL PATTERN(#28,132):: CALL SOUND(10,55
    4,1):: CALL SOUND(10,523,2):: CALL SOUND
    (10,494,3):: CALL SOUND(10,466,4):: CALL
     SOUND(10,440,5)
750 CALL SOUND(10,415,6):: CALL SOUND(10,392
    ,7):: CALL SOUND(10,370,8):: CALL SOUND(
    10,349,9):: CALL SOUND(10,330,10)
760 CALL SAY("SORRY"):: CALL POSITION(#28,R,
    V)
770 FOR D=40 TO 130 STEP 10 :: IF R>D THEN Z
    (P)=Z(P)+L(P)+L(P):: U=U+8 :: CALL SOUND
    (5,U,0):: DISPLAY AT(P,9):Z(P)ELSE 870
780 NEXT D :: GO TO 870
790 CALL SAY("TRY AGAIN"):: DISPLAY AT(5,7)B
    EEP:"PLAY AGAIN?(Y,N)" :: GOSUB 830
800 IF K=110 THEN CALL SAY("GOODBYE"):: CALL
     CLEAR :: STOP
810 W(1),W(2)=5 :: P=1 :: O,Z(1),Z(2)=0 :: D
    ISPLAY AT(5,7)BEEP:" CONTINUE GAME?" ::
    GOSUB 830
820 IF K=121 THEN 160 ELSE E(1),E(2)=6000 ::
     B(1),B(2)=1000 :: L(1),L(2)=5 :: GOTO 1
    60
830 CALL KEY(0,K,SS):: IF SS=0 THEN 830 ELSE
     RETURN
840 FOR X=18 TO 0 STEP -1 :: CALL SOUND(60,2
    00,30,200,30,0,30,-8,X):: NEXT X
```

```
850 FOR X=Ø TO 18 :: CALL SOUND(6Ø,2ØØ,3Ø,2Ø
    Ø,3Ø,O,3Ø,-8,X):: NEXT X :: RETURN
860 DISPLAY AT(P,9):Z(P):: FOR D=1 TO 1ØØ ::
     NEXT D :: CALL HCHAR(2Ø,14,32,8)
870 U=11Ø :: IF Z(P)>E(P)-1 THEN W(P)=W(P)+1
     :: CALL HCHAR(24,W(P)+2,13Ø):: E(P)=E(P
    )+6ØØØ :: CALL SOUND(1Ø,349,Ø):: CALL SO
    UND(1Ø,523,Ø)
880 O=Ø :: IF W(P)THEN W(P)=W(P)-1 :: GOTO 9
    1Ø
890 DISPLAY AT(22,11):"GAME OVER" :: FOR D=1
     TO 2ØØ :: NEXT D :: IF A THEN 79Ø
900 CALL HCHAR(22,11,32,9):: IF P=1 THEN P=2
     :: A=1 ELSE P=1 :: A=1
910 CALL HCHAR(24,3,32,1Ø):: IF A THEN 32Ø
920 IF P=1 THEN P=2 :: GOTO 32Ø ELSE P=1 ::
    GOTO 32Ø
930 DATA 96,Ø,26,14,56,13Ø,2Ø4,Ø,223,177,26,
    224,1Ø3,85,3,252,1Ø6,1Ø6,128,95,44,4,24Ø
    ,35,11,2,126,16,121
```

The Chase

Dennis M. Reddington

"The Chase" is a challenging action game. It's a relatively fast-moving game written in TI BASIC.

Watch out for those ghosts. If they catch your jewel collectors the price can be quite costly: Once all six of your collecters are caught, the game ends.

The object of "The Chase" is to collect jewels. The playfield for The Chase is a 7 × 11 grid. Move your jewel collectors around by moving the joystick or by using the keyboard's arrow keys (E, up; X, down; S, left; and D, right). If you manage to gather all the jewels you'll move to the next level of play. Be careful—don't let a ghost catch your jewel collector, for if he's caught all the jewels will be placed back onto the playing grid.

Design Considerations

The Chase is of interest to a TI-99/4A programmer because it demonstrates some ways to develop a relatively fast moving game in BASIC that pressures the player to keep moving. Several of the game design and programming considerations used in The Chase can be used in other BASIC games to speed up the action. They include:

- The use of color changes to give the appearance of fast game action;
- Limiting the playing grid's size to a relatively small portion of the screen so that, in a game like The Chase, captures and escapes can take place quickly;
- Randomly generating each game to add variety to the game's play;
- Checking first for the more common joystick movements and thus reducing the amount of time required to react to the player's request to move;
- Changing character patterns on the screen to give the player the feeling of action;
- Minimizing the time-consuming task of displaying the score and other text;
- Coordinating the sound with joystick movement;
- Increasing the difficulty level as the game progresses to higher levels.

Game Scoring

Play continues until all six jewel collectors are captured. Each time you clear the playing board, you will advance to the next level of play. Scoring is based on the level of play: For each jewel collected you'll get a number of points equal to ten times the level. For example, level 1 scores 10 points for each jewel collected, level 8 scores 80, and so on.

The Chase

```
100 CALL CLEAR
110 PRINT "T H E  C H A S E"
120 PRINT
130 PRINT "Joystick 1?  1/ENTER"
140 PRINT "Joystick 2?  2/ENTER"
150 PRINT "Keyboard  ?  3/ENTER"
160 PRINT
170 INPUT JTSW
180 PRINT
190 PRINT "=============="
200 IF JTSW=1 THEN 230
210 IF JTSW=2 THEN 230
220 IF JTSW<>3 THEN 100 ELSE 250
230 PRINT "ALPHA LOCK Off"
240 GOTO 260
250 PRINT "ALPHA LOCK On"
260 PRINT "=============="
270 PRINT
280 PRINT "Please Press ENTER"
290 PRINT "To BEGIN ..."
300 CALL KEY(0,X,Y)
310 Z=Z+1
320 IF Y=0 THEN 300
330 IF Z<100 THEN 360
340 Z=Z-90
350 GOTO 330
360 RANDOMIZE Z
370 CALL CLEAR
380 H$="FF"
390 V$="1010101010101010"
400 P$="00001818"
410 PO$="00"
420 G1$="183C5AFFFF919191"
430 G2$="183C5AFFFF242424"
440 S1$="003C5A5A5A7E3C"
450 S2$="003C7E42427E3C"
460 E1$="0000001818"
470 E2$="000024181824"
480 E3$="0200249818240048"
```

```
490 E4$="4A8142810024814A"
500 E5$="0004008000040081"
510 CALL SCREEN(2)
520 CALL COLOR(14,16,1)
530 CALL COLOR(13,14,1)
540 CALL COLOR(12,5,1)
550 CALL CHAR(136,H$)
560 CALL CHAR(128,H$)
570 CALL CHAR(120,H$)
580 CALL CHAR(137,V$)
590 CALL CHAR(129,V$)
600 CALL CHAR(121,V$)
610 CALL CHAR(138,P$)
620 CALL CHAR(130,P$)
630 CALL CHAR(122,P$)
640 CALL COLOR(11,11,1)
650 CALL CHAR(112,S1$)
660 SM=112
670 CALL CHAR(113,PO$)
680 PO=113
690 CALL COLOR(10,8,1)
700 CALL CHAR(104,G$)
710 G=104
720 CALL CHAR(105,E1$)
730 CALL CHAR(106,E2$)
740 CALL CHAR(107,E3$)
750 CALL CHAR(108,E4$)
760 CALL CHAR(109,E5$)
770 EB=105
780 CALL HCHAR(1,3,136,28)
790 CALL HCHAR(2,4,128,26)
800 CALL HCHAR(3,5,120,24)
810 CALL HCHAR(22,5,120,24)
820 CALL HCHAR(23,4,128,26)
830 CALL HCHAR(24,3,136,28)
840 CALL VCHAR(2,2,137,22)
850 CALL VCHAR(2,31,137,22)
860 CALL VCHAR(3,3,129,20)
870 CALL VCHAR(3,30,129,20)
880 CALL VCHAR(4,4,121,18)
890 CALL VCHAR(4,29,121,18)
900 FOR X=11 TO 21 STEP 2
910 CALL HCHAR(16,X,SM)
920 NEXT X
930 S1=11
940 S2=21
950 FOR X=3 TO 8
960 CALL COLOR(X,14,1)
970 NEXT X
980 CALL HCHAR(19,9,83)
```

```
990 CALL HCHAR(19,10,67)
1000 CALL HCHAR(19,11,79)
1010 CALL HCHAR(19,12,82)
1020 CALL HCHAR(19,13,69)
1030 CALL HCHAR(19,14,58)
1040 MX=2
1050 LX=LX+1
1060 IF MX=5 THEN 1080
1070 MX=MX+1
1080 PILLS=76
1090 S=S+(10*LX)
1100 M1=0
1110 M2=0
1120 O1=138
1130 O2=138
1140 FOR X=8 TO 14
1150 FOR Y=11 TO 21
1160 Z=(INT(3*RND)+1)*8+114
1170 CALL HCHAR(X,Y,Z)
1180 CALL SOUND(150,-4,1)
1190 GOSUB 2700
1200 NEXT Y
1210 NEXT X
1220 CALL HCHAR(11,16,SM)
1230 L=11
1240 C=16
1250 CALL HCHAR(8,11,G)
1260 L1=8
1270 C1=11
1280 CALL HCHAR(14,21,G)
1290 L2=14
1300 C2=21
1310 CALL SOUND(300,-5,0)
1320 SMC=SMC+1
1330 IF SMC=1 THEN 1380
1340 SMC=0
1350 CALL CHAR(112,S1$)
1360 CALL CHAR(104,G1$)
1370 GOTO 1400
1380 CALL CHAR(112,S2$)
1390 CALL CHAR(104,G2$)
1400 CALL SOUND(1,3000,5)
1410 NL=L
1420 NC=C
1430 IF JTSW=3 THEN 1440 ELSE 1500
1440 XX=0
1450 CALL KEY(0,XX,YY)
1460 IF XX<>ASC("S")THEN 1470 ELSE 1530
1470 IF XX<>ASC("D")THEN 1480 ELSE 1560
1480 IF XX<>ASC("E")THEN 1490 ELSE 1590
```

```
1490 IF XX<>ASC("X")THEN 1860 ELSE 1620
1500 CALL JOYST(JTSW,XX,YY)
1510 XX=(10*XX)+YY
1520 IF XX<>-40 THEN 1550
1530 NC=C-1
1540 GOTO 1630
1550 IF XX<>40 THEN 1580
1560 NC=C+1
1570 GOTO 1630
1580 IF XX<>4 THEN 1610
1590 NL=L-1
1600 GOTO 1630
1610 IF XX<>-4 THEN 1860
1620 NL=L+1
1630 IF NL<8 THEN 1860
1640 IF NL>14 THEN 1860
1650 IF NC<11 THEN 1860
1660 IF NC>21 THEN 1860
1670 CALL GCHAR(NL,NC,X)
1680 FOR Z=122 TO 138 STEP 8
1690 IF X<>Z THEN 1730
1700 S=S+(10*LX)
1710 PILLS=PILLS-1
1720 GOTO 1750
1730 NEXT Z
1740 IF X=PO THEN 1750 ELSE 2440
1750 CALL HCHAR(L,C,PO)
1760 CALL HCHAR(NL,NC,SM)
1770 L=NL
1780 C=NC
1790 IF PILLS<>0 THEN 1850
1800 NLSW=1
1810 FOR X=3500 TO 3300 STEP -20
1820 CALL SOUND(1,X,0)
1830 NEXT X
1840 GOTO 2520
1850 GOSUB 2700
1860 Z=INT(LX*3*RND)+1
1870 IF Z=1 THEN 1320
1880 X1=(ABS(L1-L)+ABS(C1-C))
1890 X2=(ABS(L2-L)+ABS(C2-C))
1900 HIT=1
1910 IF X1=1 THEN 2010
1920 IF X2=1 THEN 2230
1930 HIT=0
1940 IF M1<>MX THEN 1970
1950 M1=0
1960 GOTO 2230
1970 IF M2<>MX THEN 2000
1980 M2=0
```

```
1990 GOTO 2010
2000 IF X1>X2 THEN 2230
2010 M1=M1+1
2020 NL1=L1
2030 NC1=C1
2040 IF L1=L THEN 2100
2050 IF L1>L THEN 2080
2060 NL1=L1+1
2070 GOTO 2140
2080 NL1=L1-1
2090 GOTO 2140
2100 IF C1<C THEN 2130
2110 NC1=C1-1
2120 GOTO 2140
2130 NC1=C1+1
2140 CALL GCHAR(NL1,NC1,O1X)
2150 IF O1X=G THEN 1320
2160 CALL HCHAR(L1,C1,O1)
2170 O1=O1X
2180 CALL HCHAR(NL1,NC1,G)
2190 L1=NL1
2200 C1=NC1
2210 IF HIT=1 THEN 2440
2220 GOTO 1320
2230 M2=M2+1
2240 NL2=L2
2250 NC2=C2
2260 IF C2=C THEN 2320
2270 IF C2<C THEN 2300
2280 NC2=C2-1
2290 GOTO 2360
2300 NC2=C2+1
2310 GOTO 2360
2320 IF L2>L THEN 2350
2330 NL2=L2+1
2340 GOTO 2360
2350 NL2=L2-1
2360 CALL GCHAR(NL2,NC2,O2X)
2370 IF O2X=G THEN 1320
2380 CALL HCHAR(L2,C2,O2)
2390 O2=O2X
2400 CALL HCHAR(NL2,NC2,G)
2410 L2=NL2
2420 C2=NC2
2430 GOTO 2210
2440 CALL SOUND(500,-5,1)
2450 FOR X=0 TO 4
2460 CALL SOUND(150,-4,0)
2470 CALL HCHAR(16,S1,EB+X)
2480 NEXT X
```

```
2490 CALL SOUND(150,-4,0)
2500 CALL HCHAR(16,S1,PO)
2510 S1=S1+2
2520 T$=STR$(S)
2530 Z=LEN(T$)
2540 FOR X=1 TO Z
2550 TX$=SEG$(T$,X,1)
2560 Y=VAL(TX$)
2570 CALL HCHAR(19,16+X,Y+48)
2580 NEXT X
2590 FOR X=8 TO 14
2600 CALL HCHAR(X,11,PO,11)
2610 NEXT X
2620 IF S1=S2+2 THEN 2660
2630 IF NLSW<>1 THEN 1080
2640 NLSW=0
2650 GOTO 1050
2660 FOR X=1 TO 3000
2670 NEXT X
2680 CALL CLEAR
2690 END
2700 Z=INT(3*RND)+1
2710 ON Z GOTO 2720,2740,2760
2720 Z1=14
2730 GOTO 2770
2740 Z1=5
2750 GOTO 2770
2760 Z1=16
2770 Z=INT(3*RND)+12
2780 CALL COLOR(Z,Z1,1)
2790 RETURN
```

Thinking

Andy VanDuyne
TI Version by Patrick Parrish

"Thinking"—and its advanced version, "Thinking Harder"—is a game of pattern recognition and memory that tests your ability to think logically.

You have nine black boxes labeled from 1 to 9 in front of you. Your job is to make them all light up with a purple glow.

The trouble is, you can't get to them directly. Instead, you have a set of six switches, numbered from 1 to 6. Each switch controls *three* of the boxes. When you choose switch 1, for example, boxes 1, 4, and 8 might change condition. If they were all dark, then they'll all glow; if they were all glowing purple, then they'll go dark. And if 1 and 4 were purple and 8 was black, then 1 and 4 will go dark and 8 will glow purple.

The trouble is figuring out which switches control certain boxes. You know that there is a correct combination—three of the switches, toggled at once, will make all nine boxes glow. But which three? That's where luck and genius combine. It's possible to guess right with your first three choices. But if you aren't concentrating, it's also possible to get such a mishmash of purple and black boxes that it could take a hundred tries before the puzzle is solved.

How to Play

After you have typed in "Thinking" and saved it on tape or disk, run it and the game will begin. A title screen and two screens of instructions appear first. Press any key to go on.

Nine black boxes lettered from 1 to 9 appear in the center of the screen. Below the boxes you can see the number of purple boxes, which is 0 at the beginning of the game. At the top of the screen is the number of turns you have taken, which is 1 at the start of the game.

The input line just above the black boxes asks you for a number from 1 to 6. Hit a number and press ENTER. Three boxes will immediately turn purple. The turn number will change to 2 and the count of purple boxes will change to 3.

Suppose you enter the number 5, and the 1, 2, and 8 boxes glow purple. You don't know about any of the other numbers, but you know that from then on, in that game, number 5 will toggle boxes 1, 2, and 8. The pattern for each switch is randomly assigned at the beginning of each game, so that each time you play there'll be a new set of patterns. But the pattern for a particular switch will never change *during* a game.

If you choose a number and don't like what it did, choosing the same number again toggles the same three boxes and restores them to the way they were originally. It will cost you a turn each time, though, just as if you had entered a new number.

When all nine boxes turn purple, the computer congratulates you, tells you how many turns you took, and asks if you want another game. If you choose to play again, a new set of patterns is randomly created.

Strategy and Frustration

At the beginning of every game there are always two perfect solutions. The puzzle can always be solved. Winning in three or five tries is entirely a matter of luck. Students in my school average between 9 and 25 turns—slightly better than the teachers. If you become totally lost, however, it can take dozens or even a hundred tries to solve the puzzles.

But if you think logically, you should soon become quite good at the game. I won't give away the whole strategy, but you might keep in mind that any two patterns that overlap (that change the condition of the same box) cannot possibly be in the same winning combination. And in the last turn before you win, you must always have exactly six purple boxes and three black ones.

Is It Too Easy?

If you become a master at Thinking, you might want to try "Thinking Harder." In this version of the game, you have *nine* possible patterns instead of six. This makes it possible to get much more confused, and getting it right by luck alone is much less likely.

To play Thinking Harder, remove the word REM in lines 210–240. If Thinking Harder is too difficult, you can always reverse the changes and go back to Thinking again.

Thinking

```
100 GOTO 150
110 FOR U=1 TO LEN(D$)
120 CALL HCHAR(ROW,COL+U,ASC(SEG$(D$,U,1)))
130 NEXT U
140 RETURN
150 CALL CLEAR
160 CALL SCREEN(6)
170 PRINT TAB(7);"T H I N K I N G": :
180 G=6
190 B1=2
200 B2=17
210 REM G=9
220 REM B1=3
230 REM B2=26
240 REM PRINT TAB(9);"H A R D E R"
250 PRINT : : : : : : : : : : :
260 FOR I=1 TO 250
270 NEXT I
280 G$=STR$(G)
290 GOSUB 1640
300 CALL CLEAR
310 CALL SCREEN(14)
320 GOSUB 2030
330 DD=1
340 CALL CLEAR
350 FOR N=1 TO G
360 CH(N)=0
370 NEXT N
380 FOR N=1 TO 9
390 C(N)=0
400 CALL COLOR(N+5,2,2)
410 NEXT N
420 CO=0
430 GOSUB 2210
440 FOR N=1 TO G
450 RANDOMIZE
460 Z=INT(RND*G)+1
470 IF CH(Z)<>0 THEN 460
480 CH(Z)=N
490 NEXT N
500 FOR B=1 TO B1
510 FOR N=1 TO 9
520 RANDOMIZE
530 Z=INT(RND*9)+1
540 IF Y(Z)<>0 THEN 530
550 Y(Z)=N
560 NEXT N
570 FOR N=1 TO 9
```

```
580  X=Y(N)
590  X$=SEG$(STR$(X),1,1)
600  P$(B)=P$(B)&X$
610  NEXT N
620  FOR N=1 TO 9
630  Y(N)=0
640  NEXT N
650  GOSUB 2210
660  NEXT B
670  H$=P$(1)&P$(2)
680  IF G<>9 THEN 700
690  H$=P$(1)&P$(2)&P$(3)
700  FOR N=1 TO B2 STEP 3
710  P$(INT(N/3)+1)=SEG$(H$,N,3)
720  NEXT N
730  CALL SCREEN(15)
740  FOR I=9 TO 23
750  CALL VCHAR(4,I,64,15)
760  NEXT I
770  C1=72
780  R=6
790  FOR S=1 TO 3
800  J=11
810  FOR Q=C1 TO C1+16 STEP 8
820  FOR I=R TO R+2
830  CALL HCHAR(I,J,Q,3)
840  NEXT I
850  J=J+4
860  NEXT Q
870  R=R+4
880  C1=C1+24
890  NEXT S
900  KH=49
910  FOR T=12 TO 20 STEP 4
920  CALL HCHAR(7,T,KH)
930  CALL HCHAR(11,T,KH+3)
940  CALL HCHAR(15,T,KH+6)
950  KH=KH+1
960  NEXT T
970  Q=0
980  ROW=2
990  COL=10
1000 D$="# (1-"&G$&") ?"
1010 GOSUB 110
1020 ROW=20
1030 COL=10
1040 D$=",.*& # :"
1050 GOSUB 110
1060 ROW=22
```

```
1070 D$="'.*'%"&CHR$(34)&"+  :"
1080 GOSUB 110
1090 FOR N=1 TO 9
1100 IF C(N)<>14 THEN 1130
1110 CALL COLOR(5+N,14,14)
1120 GOTO 1140
1130 CALL COLOR(5+N,2,2)
1140 NEXT N
1150 FOR I=1 TO 9
1160 IF C(I)<>14 THEN 1180
1170 CO=CO+1
1180 NEXT I
1190 CALL HCHAR(22,21,CO+48)
1200 IF CO=9 THEN 1450
1210 CO=0
1220 Q=Q+1
1230 D$=STR$(Q)
1240 ROW=20
1250 COL=19
1260 GOSUB 110
1270 CALL HCHAR(2,21,30)
1280 CALL KEY(0,K,ST)
1290 IF ST=1 THEN 1310
1300 CALL HCHAR(2,21,32)
1310 IF (K<49)+(K>48+G)THEN 1270
1320 CALL SOUND(50,440,4)
1330 CALL HCHAR(2,21,K)
1340 SE=CH(K-48)
1350 FOR N=1 TO 3
1360 W=VAL(SEG$(P$(SE),N,1))
1370 IF C(W)<>0 THEN 1400
1380 C(W)=14
1390 GOTO 1420
1400 IF C(W)<>14 THEN 1420
1410 C(W)=0
1420 NEXT N
1430 GOTO 1090
1440 REM YOU WIN!
1450 L1=2
1460 L2=15
1470 S1=1
1480 FOR U=1 TO 3
1490 FOR I=L1 TO L2 STEP S1
1500 CALL SOUND(-1,110+I*10,3)
1510 CALL SCREEN(I)
1520 NEXT I
1530 S1=S1*-1
1540 NEXT U
1550 ROW=24
1560 COL=12
```

```
1570 D$="!/!$& ?"
1580 GOSUB 110
1590 CALL KEY(0,K,ST)
1600 IF ST=0 THEN 1590
1610 IF (K<>78)*(K<>89)THEN 1590
1620 IF K=89 THEN 330 ELSE 2240
1630 REM INSTRUCTIONS
1640 CALL CLEAR
1650 CALL SCREEN(11)
1660 PRINT "YOU WILL SEE 9 BLACK BLOCKS."
1670 PRINT
1680 PRINT "BY ENTERING A NUMBER BETWEEN"
1690 PRINT
1700 PRINT "1 AND ";G$;", YOU CAN CHANGE"
1710 PRINT
1720 PRINT "SOME OF THEM TO PURPLE."
1730 PRINT
1740 PRINT
1750 PRINT "BUT, SOME PURPLE ONES MIGHT"
1760 PRINT
1770 PRINT "TURN BACK TO BLACK !"
1780 PRINT
1790 PRINT
1800 PRINT "EACH NUMBER YOU ENTER WILL"
1810 PRINT
1820 PRINT "CHANGE THE COLORS IN ITS OWN"
1830 PRINT
1840 PRINT "WAY."
1850 PRINT
1860 GOSUB 1970
1870 CALL CLEAR
1880 PRINT
1890 PRINT "TRY TO CHANGE ALL THE BLOCKS"
1900 PRINT
1910 PRINT "TO PURPLE IN AS FEW TRIES AS"
1920 PRINT
1930 PRINT "YOU CAN."
1940 FOR I=1 TO 10
1950 PRINT
1960 NEXT I
1970 PRINT
1980 PRINT TAB(3);"PRESS A KEY TO CONTINUE";
1990 CALL KEY(0,K,ST)
2000 IF ST=0 THEN 1990
2010 RETURN
2020 REM DEFINE COLORS AND CHARS
2030 FOR I=72 TO 136 STEP 8
2040 CALL CHAR(I,"")
2050 NEXT I
2060 FOR I=1 TO 12
```

```
2070 READ LL,L$
2080 CALL CHAR(LL,L$)
2090 NEXT I
2100 CALL COLOR(5,5,1)
2110 FOR I=6 TO 14
2120 CALL COLOR(I,2,2)
2130 NEXT I
2140 RETURN
2150 DATA 33,003844447C444444,34,007C4040784
     0407C
2160 DATA 47,003C40405C444438,36,00381010101
     01038
2170 DATA 37,004040404040407C,38,0044646454 4
     C4C44
2180 DATA 39,0078444478404040,42,00784444785
     04844
2190 DATA 43,0038444038044438,44,007C1010101
     01010
2200 DATA 46,0044444444444438,64,FFFFFFFFFFF
     FFFFF
2210 DD=DD+2
2220 CALL SCREEN(DD)
2230 RETURN
2240 END
```

Bowling Champp

Joseph Ganci
TI Translation by Patrick Parrish

Now you can go bowling without the expense of renting special shoes or suffering the embarrassment of rolling a gutter ball in front of dozens of people. "Bowling Champ" is a game for one to four players.

Some games, such as *Pac-Man* or *Adventure*, create their own unique fantasy worlds, while others are simulations of reality. "Bowling Champ" is an example of the latter.

It's not easy to take a game with countless physical variables (such as bowling) and reduce it to numbers so it can be re-created by a computer—especially a microcomputer. Compromises must be made. Usually the game must be modified in major ways to make it possible to program. The result is a hybrid game, an approximation of reality, that resembles the original but has new aspects of its own.

Bowling Champ is a reasonable simulation of a game of ten pins, given the limitations imposed by a BASIC program which must remain short enough to publish. The elements of skill and luck have been preserved, and the scoring is authentic.

Up to Four Players

When you first run Bowling Champ, the program asks for the number of players. Up to four people can play.

Next you enter the players' names. All names of more than eight characters long will be truncated to eight characters.

Now you're ready to bowl the first frame. The bowling ball rapidly moves up and down across the alley until you press the space bar. This rolls the ball down the alley and knocks over the pins—unless you've thrown a gutter ball. The trick is to time your release so the ball rolls down the center of the alley to score a strike.

In case you're unfamiliar with how a game of ten pins is scored, here's a brief summary:

A game consists of ten frames or turns. Each player gets one or two balls per frame. If you roll a strike—knocking down all ten pins with the first ball—you don't get a second ball, but the current ball's score is ten plus the total of your next two throws.

If some pins are left standing after your first ball, you get a second ball. If you knock down all the remaining pins, it counts as a spare, and the current ball's score is ten plus your next throw.

If any pins remain after your second ball (no strike or spare), the number of pins knocked down in that frame is added to your previous score.

Rolling a spare in the tenth (last) frame gains you one extra ball; rolling a strike in the tenth frame gains two extra balls.

Therefore, a perfect game—ten strikes during regular play plus two strikes with the extra bowling balls—scores 300 points. Needless to say, this doesn't happen very often, either in real bowling or in Bowling Champ.

Is It Too Hard?

You can make the game easier with just two simple changes. Remove STEP 2 from line 1660 and delete line 1740 entirely.

Bowling Champ

```
100 GOTO 150
110 FOR I=1 TO LEN(H$)
120 CALL HCHAR(R,C+I,ASC(SEG$(H$,I,1)))
130 NEXT I
140 RETURN
150 GOSUB 2440
160 DIM NAME$(3),SS(3),TT(3)
170 G=15
180 H=23
190 CALL CLEAR
200 CALL SCREEN(6)
210 PRINT TAB(8);"B O W L I N G": :
220 PRINT TAB(9);"C H A M P !": : : : : : :
    : : : : :
230 PRINT TAB(3);"HOW MANY PLAYERS (1-4) ?";
240 CALL KEY(0,A,S)
250 IF S=0 THEN 240
260 IF (A<49)+(A>52)THEN 240
270 A=A-48
280 CALL CLEAR
290 CALL SCREEN(13)
```

```
300 X$="NAMES"
310 IF A<>1 THEN 330
320 X$="NAME"
330 PRINT "TYPE IN YOUR ";X$;":": : :
340 FOR I=0 TO A-1
350 PRINT :
360 PRINT TAB(4);"PLAYER #";I+1;" ";
370 INPUT NAME$(I)
380 NAME$(I)=SEG$(NAME$(I),1,8)
390 NEXT I
400 REM DRAW GAME SCREEN
410 CALL CLEAR
420 CALL SCREEN(12)
430 H$="1  2  3  4  5  6  7  8  9  10"
440 R=1
450 C=1
460 GOSUB 110
470 R=2
480 H$="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
490 GOSUB 110
500 FOR J=1 TO A
510 H$="  y  y  y  y  y  y  y  y  y"
520 R=2*J+1
530 GOSUB 110
540 H$="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
550 R=2*J+2
560 GOSUB 110
570 NEXT J
580 R=13+(A>2)*2
590 FOR J=1 TO A
600 C=1-((J=2)+(J=4))*15
610 R=R-(J=3)*2
620 H$=NAME$(J-1)&":"
630 GOSUB 110
640 NEXT J
650 REM INITIALIZE SCORE STATE
660 FOR J=0 TO A-1
670 SS(J)=1
680 TT(J)=0
690 NEXT J
700 REM PUT DOWN ALLEY
710 CALL COLOR(13,1,1)
720 FOR J=G TO H
730 CALL HCHAR(J,2,E,30)
740 NEXT J
750 CALL HCHAR(14,2,120,30)
760 CALL HCHAR(24,2,120,30)
770 REM MAIN LOOP
780 FOR Q=1 TO 10
790 FOR RR=0 TO A-1
```

```
800 CC=(RR+1)*3
810 IF RR<>3 THEN 830
820 CC=14
830 CALL COLOR(13,2,CC)
840 CALL COLOR(11,15,CC)
850 B1=0
860 GOSUB 1320
870 IF J1=10 THEN 900
880 B1=1
890 GOSUB 1450
900 IF Q<>10 THEN 920
910 ON S GOTO 920,1040,1040,920,1160
920 NEXT RR
930 NEXT Q
940 R=19
950 C=7
960 H$="PLAY AGAIN (Y/N) ?"
970 GOSUB 110
980 CALL KEY(0,K,ST)
990 IF ST=0 THEN 980
1000 IF (K<>89)*(K<>78)THEN 980
1010 IF K=89 THEN 170
1020 STOP
1030 REM 10TH FRAME-EXTRA BALLS
1040 R=19
1050 C=2
1060 H$="TAKE 2 MORE BALLS, "&NAME$(RR)
1070 GOSUB 110
1080 FOR I=1 TO 300
1090 NEXT I
1100 CALL HCHAR(19,2,E,29)
1110 SS(RR)=S-1
1120 B1=1
1130 GOSUB 1320
1140 IF J=10 THEN 1230
1150 GOTO 1270
1160 C=3
1170 R=19
1180 H$="TAKE 1 MORE BALL, "&NAME$(RR)
1190 GOSUB 110
1200 FOR I=1 TO 300
1210 NEXT I
1220 CALL HCHAR(19,3,E,28)
1230 SS(RR)=1
1240 B1=2
1250 GOSUB 1320
1260 GOTO 920
1270 SS(RR)=1
1280 B1=2
1290 GOSUB 1450
```

```
1300 GOTO 920
1310 REM FIRST BALL
1320 FOR I=16 TO 22 STEP 2
1330 CALL VCHAR(I,30,112)
1340 NEXT I
1350 FOR I=17 TO 21 STEP 2
1360 CALL VCHAR(I,29,112)
1370 NEXT I
1380 CALL HCHAR(18,28,112)
1390 CALL HCHAR(20,28,112)
1400 CALL HCHAR(19,27,112)
1410 PS=-1
1420 J1=0
1430 GOTO 1460
1440 REM SECOND BALL
1450 PS=0
1460 GOSUB 1580
1470 T=TT(RR)
1480 S=SS(RR)
1490 T=T+J
1500 ON SS(RR)GOSUB 2200,2250,2300,2340,2390
1510 TT(RR)=T
1520 SS(RR)=S
1530 R=13+(A>2)*2-(RR>1)*2
1540 C=10-((RR=1)+(RR=3))*15
1550 H$=STR$(TT(RR))
1560 GOSUB 110
1570 RETURN
1580 IF (Q=1)*(PS=-1)*(RR=0)THEN 1650
1590 C=30
1600 FOR HH=C TO 3 STEP -1
1610 CALL HCHAR(15,HH+1,E)
1620 CALL HCHAR(15,HH,B)
1630 NEXT HH
1640 CALL HCHAR(15,HH+1,E)
1650 C=3
1660 FOR R=G TO H STEP 2
1670 CALL HCHAR(R,C,B)
1680 CALL KEY(0,K,ST)
1690 CALL HCHAR(R,C,E)
1700 IF ST=0 THEN 1730
1710 ROW=R
1720 R=H
1730 NEXT R
1740 G=15-(G=15)
1750 IF ST=0 THEN 1660
1760 R=ROW
1770 J=0
1780 FOR C=3 TO 25
1790 CALL HCHAR(R,C,E)
```

```
1800 CALL HCHAR(R,C+1,B)
1810 CALL SOUND(-1,130,2)
1820 NEXT C
1830 CALL GCHAR(R,C+1,X)
1840 IF (X<>112)*(C<>31)THEN 2020
1850 IF C=31 THEN 2060
1860 IF X<>112 THEN 2020
1870 CALL SOUND(10,-7,5)
1880 J=J+1
1890 C=C+1
1900 FOR D=-1 TO 1 STEP 2
1910 Y1=R
1920 X1=C
1930 X1=X1+1
1940 Y1=Y1+D
1950 CALL GCHAR(Y1,X1,X)
1960 IF X<>112 THEN 2010
1970 J=J+1
1980 CALL HCHAR(Y1,X1,E)
1990 CALL SOUND(10,-7,5)
2000 GOTO 1930
2010 NEXT D
2020 CALL HCHAR(R,C-1,E,2)
2030 C=C+1
2040 CALL HCHAR(R,C,B)
2050 GOTO 1830
2060 CALL HCHAR(R,C,E)
2070 J1=J1+J
2080 R=3+RR*2
2090 C=-2+3*Q+B1
2100 G1=J+48
2110 IF J1<>10 THEN 2150
2120 G1=47
2130 IF PS=0 THEN 2150
2140 G1=88
2150 IF B1=0 THEN 2170
2160 G1=G1+50
2170 H$=CHR$(G1)
2180 GOSUB 110
2190 RETURN
2200 IF J1<>10 THEN 2240
2210 S=5
2220 IF PS=0 THEN 2240
2230 S=2
2240 RETURN
2250 T=T+J
2260 S=4
2270 IF J<>10 THEN 2290
2280 S=3
2290 RETURN
```

```
2300 T=T+J*2
2310 IF J=10 THEN 2330
2320 S=4
2330 RETURN
2340 T=T+J
2350 S=1
2360 IF J1<>10 THEN 2380
2370 S=5
2380 RETURN
2390 T=T+J
2400 S=1
2410 IF J<>10 THEN 2430
2420 S=2
2430 RETURN
2440 FOR I=97 TO 107
2450 READ C$
2460 CALL CHAR(I,C$)
2470 NEXT I
2480 FOR I=112 TO 128 STEP 8
2490 READ C$
2500 CALL CHAR(I,C$)
2510 NEXT I
2520 CALL CHAR(121,"00101010101000")
2530 CALL CHAR(138,"FFBBBBD7EFD7BBBB")
2540 E=129
2550 CALL CHAR(129,"")
2560 B=128
2570 RETURN
2580 DATA FFFFBF7EFDFBFFF,FFC7BBBBBBBBC7,
     FFEFCFEFEFEFEFC7,FFC7BBFBF7EFDF83
2590 DATA FFC7BBFBE7FBBBC7,FFF7E7D7B783F7F7,
     FF83BF87FBFBBBC7
2600 DATA FFE7DFBF87BBBBC7,FF83FBF7EFDFDFDF,
     FFC7BBBBC7BBBBC7
2610 DATA FFC7BBBBC3FBF7CF
2620 DATA 1C1C081C3E3E3E1C,000000FF00000000,
     003C7E7E7E7E7E3C
```

Worm of Bemer

Stephen D. Fultz
TI Translation by Patrick Parrish

Nerm the worm is lost in Bemer Castle and needs your help to get home. You must guide him through 11 rooms and help him find magic mushrooms to eat along the way. The journey is a navigator's nightmare, because you never know where the next mushroom will grow, and if Nerm hits a wall or gets trapped by his tail, he loses one of his lives.

"Worm of Bemer" is a fast-paced arcade game in which Nerm the Worm travels through rooms eating magic mushrooms. Nerm is lost in Bemer Castle and wants to return home. Guide Nerm to a mushroom using the keyboard arrow keys (E, S, D, and X) so he can keep up his strength for the journey. After eating five mushrooms in a room, Nerm can exit to the next room. You must guide Nerm through 11 rooms before he finds his home. You start out with four lives. If you touch anything besides a mushroom you will lose a life.

At the top of the screen you will see the current score, what room Nerm is in, how many mushrooms Nerm must eat to open the exits, and how many lives Nerm has left, including the current life. You get 100 points, plus bonus points, for every mushroom you eat. Nerm gets a bonus life after completing the first two rooms and another for every third room thereafter.

Adding More Features

You can learn a lot about programming and games by modifying the action and settings in Worm of Bemer. Some features you might add include a routine to save the high score to disk, adding more players, or having Nerm go to a different room depending on which exit he takes. Simpler enhancements would be changing the number of mushrooms that Nerm must eat or changing his speed.

Worm of Bemer

```
3 DIM NN(29),RANK$(12)
5 GOSUB 11000
```

```
10 GOTO 5000
20 FOR I=1 TO LEN(H$)
30 CALL HCHAR(ROW,COL+I,ASC(SEG$(H$,I,1)))
35 NEXT I
40 RETURN
100 CALL KEY(0,K,ST)
105 IF (K<>68)+(OD=2)THEN 110
106 DX=1
107 DY=0
108 DI=1
110 IF (K<>83)+(OD=1)THEN 115
111 DX=-1
112 DY=0
113 DI=2
115 IF (K<>69)+(OD=3)THEN 120
116 DY=-1
117 DX=0
118 DI=4
120 IF (K<>88)+(OD=4)THEN 140
125 DY=1
130 DX=0
135 DI=3
140 CALL HCHAR(YA,XA,136)
145 OD=DI
150 XA=XA+DX
152 YA=YA+DY
154 L=LEN(XA$)
156 XA$=XA$&CHR$(XA)
158 YA$=YA$&CHR$(YA)
160 CALL GCHAR(YA,XA,Z)
162 IF Z<>32 THEN 200
164 CALL HCHAR(YA,XA,128)
166 CALL SOUND(1,622,2)
168 IF L<WO THEN 100
170 CALL HCHAR(ASC(YA$),ASC(XA$),32)
172 LL=LEN(XA$)-1
174 XA$=SEG$(XA$,2,LL)
176 YA$=SEG$(YA$,2,LL)
180 GOTO 100
200 CALL SOUND(100,311,2)
201 CALL HCHAR(YA,XA,128)
203 GOSUB 6600
205 IF Z<>MUSH THEN 260
210 WO=WO+15+2*LO
212 IF WO<185 THEN 215
214 WO=185
215 RANDOMIZE
216 XX=RND*28+3
218 X=RND*19+4
220 CALL GCHAR(X,XX,H1)
```

```
222 IF H1<>32 THEN 216
224 SC=SC+100+LO*7
228 HI=HI-1
230 GOSUB 6600
232 IF HI>0 THEN 245
234 CALL HCHAR(3,17,104)
236 CALL HCHAR(13,2,104)
238 CALL HCHAR(13,31,104)
240 CALL HCHAR(23,17,104)
241 FOR I=3 TO 30 STEP 3
242 CALL SOUND(100,1900,I)
243 NEXT I
244 GOTO 100
245 CALL HCHAR(X,XX,MUSH)
250 GOTO 100
260 IF Z=104 THEN 270
261 IF LI=1 THEN 7500
264 GOSUB 7500
266 GOTO 290
270 CALL HCHAR(YA,XA,136)
272 GOSUB 7000
275 FOR DE=110 TO 880 STEP 32
277 PRINT
279 CALL SOUND(1,DE,2)
280 CALL SOUND(-1,DE,2)
281 NEXT DE
282 LO=LO+1
283 IF LO=12 THEN 1200
284 WO=5
285 L1=L1+1
286 IF LO>EX THEN 9100
287 CALL COLOR(14,L1,1)
288 CALL CLEAR
289 GOSUB 1300
290 GOSUB 6600
300 ON LO GOTO 5080,400,500,550,600,700,800,
    450,550,1000,1100,1200
399 GOTO 5080
400 REM SECOND SCREEN
410 CALL HCHAR(13,5,120,24)
420 GOTO 5080
449 REM SCREEN
450 CALL VCHAR(7,15,120,16)
455 CALL HCHAR(9,6,120,22)
460 GOTO 5080
499 REM FOURTH SCREEN
500 CALL HCHAR(6,5,120,24)
505 CALL HCHAR(20,5,120,24)
510 GOTO 5080
549 REM FIFTH SCREEN
```

```
550 CALL HCHAR(7,6,120,22)
555 CALL VCHAR(8,15,120,16)
560 GOTO 5080
599 REM FRAME 6
600 CALL HCHAR(12,3,120,13)
610 CALL HCHAR(12,19,120,12)
620 GOTO 5080
699 REM FRAME 7
700 FOR I=8 TO 18
710 CALL HCHAR(I,7,120,7)
715 CALL HCHAR(I,18,120,8)
720 NEXT I
725 GOTO 5080
799 REM FRAME 8
800 CALL HCHAR(8,3,120,13)
805 CALL HCHAR(14,12,120,19)
810 CALL HCHAR(18,3,120,13)
815 GOTO 5080
999 REM FRAME 9
1000 GOSUB 1400
1015 FOR T=5 TO 21
1020 CALL HCHAR(T,4,32,16)
1025 NEXT T
1030 GOTO 5080
1100 GOSUB 1400
1110 FOR T=5 TO 21
1115 CALL HCHAR(T,4,32,20)
1120 NEXT T
1125 GOTO 400
1199 REM YOU WIN!!
1200 CALL CLEAR
1205 CALL SCREEN(3)
1206 FOR I=4 TO 8
1207 CALL COLOR(I,2,1)
1208 NEXT I
1210 PRINT TAB(9);"NERM'S HOME!"
1220 PRINT
1230 PRINT
1240 PRINT TAB(10);"THANK YOU!"
1250 FOR T=1 TO 9
1260 PRINT
1270 NEXT T
1275 FOR T=1 TO 3
1280 FOR I=110 TO 880 STEP 30
1283 CALL SOUND(1,I,2)
1284 CALL SOUND(-1,I,2)
1285 NEXT I
1286 FOR I=880 TO 110 STEP -30
1287 CALL SOUND(1,I,2)
1288 CALL SOUND(-1,I,2)
```

```
1289 NEXT I
129Ø NEXT T
1291 CALL SCREEN(2)
1293 GOTO 77ØØ
13ØØ CALL CLEAR
13Ø5 PRINT "SCORE :";TAB(2Ø);"ROOM :"
131Ø PRINT "MUSHROOMS :";TAB(2Ø);"LIVES :"
132Ø FOR T=1 TO 21
133Ø PRINT
134Ø NEXT T
135Ø RETURN
14ØØ FOR T=5 TO 21
141Ø CALL HCHAR(T,4,12Ø,26)
142Ø NEXT T
143Ø RETURN
4999 REM UP THE GAME
5ØØØ GOSUB 1ØØØØ
5ØØ5 MUSH=112
5Ø1Ø LI=4
5Ø15 SC=Ø
5Ø2Ø LO=1
5Ø35 HI=5
5Ø4Ø WO=5
5Ø45 EX=2
5Ø5Ø L1=3
5Ø55 GOSUB 55ØØ
5Ø6Ø CALL CLEAR
5Ø65 CALL SCREEN(2)
5Ø66 FOR I=3 TO 8
5Ø67 CALL COLOR(I,16,1)
5Ø68 NEXT I
5Ø7Ø GOSUB 13ØØ
5Ø75 GOSUB 66ØØ
5Ø8Ø XA$=""
5Ø81 YA$=""
5Ø85 XA=17
5Ø86 YA=18
5Ø91 DX=Ø
5Ø93 DY=-1
51Ø3 IF HI<6 THEN 51Ø7
51Ø5 HI=5
51Ø7 IF HI>-1 THEN 511Ø
51Ø9 HI=Ø
511Ø DI=4
5115 FOR I=2 TO 31 STEP 29
512Ø CALL VCHAR(3,I,12Ø,21)
5125 NEXT I
513Ø FOR I=3 TO 23 STEP 2Ø
5135 CALL HCHAR(I,3,12Ø,28)
514Ø NEXT I
```

```
5145 CALL HCHAR(24,3,137,28)
5150 IF HI>0 THEN 5174
5155 CALL HCHAR(3,17,104)
5160 CALL HCHAR(12,2,104)
5165 CALL HCHAR(12,31,104)
5167 CALL HCHAR(23,17,104)
5171 GOTO 150
5174 RANDOMIZE
5175 XX=RND*28+3
5178 X=RND*19+4
5180 CALL GCHAR(X,XX,H1)
5185 IF H1<>32 THEN 5174
5190 CALL HCHAR(X,XX,MUSH)
5200 GOTO 150
5500 CALL CLEAR
5505 PRINT TAB(10);"GET READY!"
5510 FOR T=1 TO 12
5515 PRINT
5520 NEXT T
5525 FOR I=1 TO 14
5530 CALL SOUND(100,NN(I),2)
5535 NEXT I
5540 RETURN
6599 REM PRINT SCORE
6600 H$=STR$(SC)
6603 ROW=1
6604 COL=10
6605 GOSUB 20
6607 H$=STR$(LO)
6608 COL=28
6609 GOSUB 20
6610 H$=STR$(HI)
6611 ROW=2
6620 COL=14
6625 GOSUB 20
6630 H$=STR$(LI)
6635 COL=29
6640 GOSUB 20
6650 RETURN
6999 REM NERM LEAVES
7000 SP=SP-5
7005 GOSUB 6600
7010 HI=5
7015 L=LEN(XA$)
7020 FOR I=1 TO L
7025 CALL SOUND(2,110+I*2,2)
7030 CALL HCHAR(ASC(YA$),ASC(XA$),32)
7035 LL=LEN(XA$)-1
7040 XA$=SEG$(XA$,2,LL)
7045 YA$=SEG$(YA$,2,LL)
```

```
7050 NEXT I
7060 RETURN
7499 REM OOP!!
7500 CALL CLEAR
7505 PRINT TAB(13);"OOPS"
7510 FOR I=1 TO 12
7515 PRINT
7520 NEXT I
7525 LI=LI-1
7547 FOR I=14 TO 24
7549 CALL SOUND(10,I*40,2)
7551 NEXT I
7553 FOR I=1 TO 30
7555 NEXT I
7560 IF LI<1 THEN 7700
7575 GOSUB 1300
7600 RETURN
7699 REM THE GAME ENDS
7700 CALL CLEAR
7704 FOR I=3 TO 8
7705 CALL COLOR(I,16,1)
7706 NEXT I
7710 IF HS>SC THEN 7750
7720 HS=SC
7721 FOR I=1 TO 5
7722 PRINT
7723 NEXT I
7725 PRINT TAB(8);"NEW HIGH SCORE"
7728 FOR T=110 TO 1760 STEP 50
7729 CALL SOUND(2,T,2)
7730 NEXT T
7740 FOR I=1 TO 5
7743 PRINT
7745 NEXT I
7750 PRINT TAB(7);"YOUR SCORE: ";SC
7755 PRINT
7760 PRINT TAB(7);"HIGH SCORE: ";HS
7770 FOR I=1 TO 3
7775 PRINT
7780 NEXT I
7785 PRINT TAB(5);"YOUR NEW RANK IS :"
7790 PRINT
7795 PRINT TAB(9);RANK$(LO)
7796 FOR I=15 TO 29
7797 CALL SOUND(100,NN(I),2)
7798 NEXT I
7800 PRINT
7805 PRINT
7806 PRINT
7810 PRINT "(C TO CONTINUE  Q TO QUIT)"
```

```
7815 FOR T=1 TO 4
7816 PRINT
7817 NEXT T
7820 CALL KEY(0,K,ST)
7830 IF ST=0 THEN 7820
7840 IF (K<>67)*(K<>81)THEN 7820
7845 IF K=67 THEN 5000
7850 STOP
9099 REM EXTRA LIFE
9100 CALL CLEAR
9110 PRINT TAB(11);"BONUS LIFE"
9120 FOR I=1 TO 12
9125 PRINT
9130 NEXT I
9132 FOR I=1 TO 30 STEP 2
9134 CALL SOUND(100,1175,I)
9136 NEXT I
9140 EX=EX+3
9145 LI=LI+1
9150 GOTO 287
10000 CALL CLEAR
10001 FOR T=3 TO 8
10003 CALL COLOR(T,2,1)
10006 NEXT T
10010 CALL COLOR(14,3,1)
10015 CALL SCREEN(15)
10020 PRINT TAB(10);"WELCOME TO"
10021 FOR T=1 TO 4
10022 PRINT
10023 NEXT T
10025 PRINT TAB(8);"NERM OF BEMER"
10028 FOR T=1 TO 9
10030 PRINT
10032 NEXT T
10034 PRINT "USE E,S,D, & X KEYS TO MOVE"
10036 PRINT
10040 CALL HCHAR(21,3,136,4)
10042 CALL HCHAR(21,8,128)
10045 FOR I=1 TO 22
10047 CALL HCHAR(21,6+I,136)
10050 CALL HCHAR(21,7+I,128)
10052 CALL SOUND(10,622,2)
10055 CALL HCHAR(21,2+I,32)
10057 FOR T=1 TO 20
10058 NEXT T
10060 NEXT I
10065 FOR T=1 TO 100
10070 NEXT T
10075 RETURN
10999 REM REDEFINE CHARS
```

```
11000 FOR I=104 TO 136 STEP 8
11015 READ A$
11020 CALL CHAR(I,A$)
11025 NEXT I
11030 DATA FFFFFFFFFFFFFFFF,187EFFFF18181818
      ,FF81BDA5A5BD81FF
11032 DATA 8142243C7E5A3C18,387CFEFEFEFE7C38
11033 CALL COLOR(10,2,2)
11035 CALL COLOR(11,14,1)
11040 CALL COLOR(12,2,10)
11045 CALL COLOR(13,7,1)
11050 CALL CHAR(137,"FFFFFFFFFFFFFFFF")
11060 FOR I=1 TO 9
11065 READ RANK$(I)
11070 NEXT I
11075 FOR I=10 TO 12
11080 RANK$(I)="HALL OF FAME"
11085 NEXT I
11090 DATA ZERO,ROOKIE,NOVICE,AVERAGE
11092 DATA MASTER,GRAND MASTER,WIZARD,GRAND
      WIZARD
11094 DATA SUPER STAR
11100 FOR I=1 TO 29
11110 READ NN(I)
11120 NEXT I
11130 DATA 262,349,40000,349,392,40000,392,4
      40,523,440,523,440,349,40000
11135 DATA 349,40000,40000,262,247,262,294,2
      94,262,40000,40000,40000,330,330,349
11140 RETURN
```

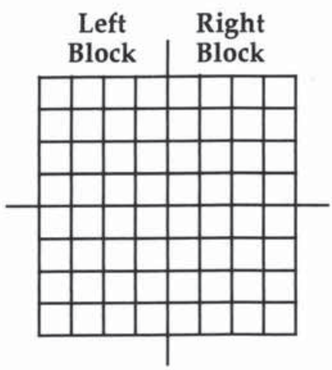
5
Sound and Graphics

5
TI Graphics Made Easy

Lyle O. Haga

There is a better way of figuring out pattern-identifier code than that presented in the TI manual.

The TI screen is divided up into a giant grid of 24 rows and 32 columns for graphics. This grid, shown in your TI manual in the CALL CHAR section, makes 768 positions, or squares, for you to put your graphics in. Each square of the grid is divided into an 8 × 8 grid consisting of 64 dots to be turned on or off. Each 8 × 8 grid is divided into a "left block" and a "right block."



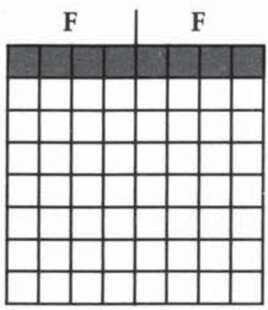
Each time you define a pattern-identifier, you use all 64 dots whether or not you so stipulate. Thus, the statement CALL CHAR(100,"FF") covers all 64 dots even though you stipulated only the top row of eight dots to be turned off; the remaining dots stay turned on. This can be seen by a simple little exercise. Make a box outline, 4 × 4.

On the surface this sounds like a pretty simple exercise, and it is. The problem is that many people probably won't think it through, and will come up with the following:

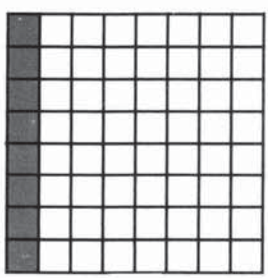
```
10 CALL CLEAR
20 CALL CHAR(100,"FF")
30 CALL CHAR(101,"8080808080808080")
```

```
40 CALL HCHAR(12,8,100,4)
50 CALL HCHAR(16,8,100,4)
60 CALL HCHAR(12,8,101,4)
70 CALL HCHAR(12,12,101,4)
80 GOTO 80
```

No matter what you do, this won't work; there will always be a gap somewhere. Remember that even though you didn't stipulate all 64 dots in CHAR 100, you still have them to deal with.



On top of this you put the following:



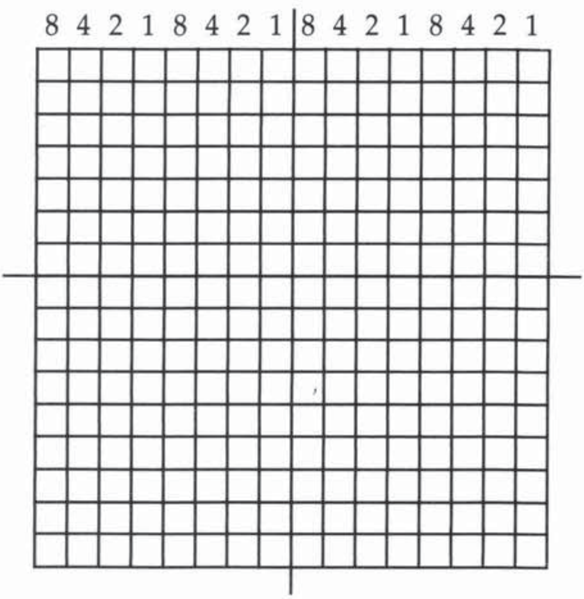
You should be able to see where the gap comes in now. When you put CHAR 101 on top of CHAR 100, the dots you left turned on cover the dots you turned off, thus the gap.

Here's one solution to the problem:

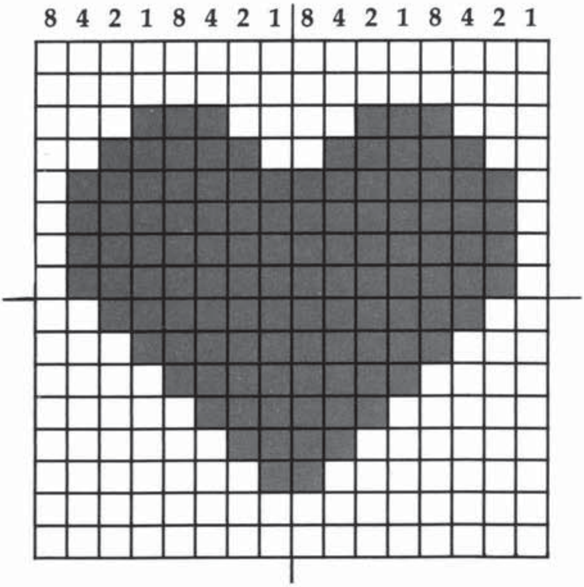
```
10 CALL CLEAR
20 CALL CHAR(100,"00000000000000FF")
30 CALL CHAR(101,"FF")
40 CALL CHAR(102,"8080808080808080")
50 CALL CHAR(103,"0101010101010101")
60 CALL VCHAR(12,8,102,4)
70 CALL VCHAR(12,11,103,4)
80 CALL HCHAR(11,8,100,4)
90 CALL HCHAR(16,8,101,4)
100 GOTO 100
```

There's an easier way of defining graphics? The new method is one your kids learned in school, called base 16. Using base 16, you write the numbers 8,4,2,1,8,4,2,1 across the top of each 8 × 8 grid. Let's see how this works in defining the heart; we will make it two positions high and two wide.

If you are planning to do many graphics, you should get some graph paper—this will make it easier. Let each square on the graph paper represent one dot; this gives you 16 squares wide and 16 squares high. Make the outline with a heavy line. Count horizontally from the left 4, 8, and 12 lines; make these heavier than the other lines, and make the eighth line even heavier and have it extend beyond the outline. This will mark off your left and right blocks and one position from another. Now, counting vertically, go down eight and darken this line, going beyond the outline. Across the top, put your base 16 numbers 8, 4, 2, 1, 8, 4, 2, 1, and your paper should look like this:



With this, let's make our heart. First, color in all the squares making your heart. Then, starting at the top row, add up the numbers over the squares you darkened. If the total is under ten, your pattern code will be that number, and if it is over nine, you see the letters A–F. You do the one complete grid and then move to the right; when you are through, move down to the next line. You should come up with the following results:



A = 10
B = 11
C = 12
D = 13
E = 14
F = 15

Row one has no darkened squares, so the code is zero for both left and right blocks. You get the same results with row two. In row three, a square under the number 1 is darkened in the left block of grid one, so the code is 1. In the right block, squares under the 8 and 4 are darkened, so the code is C. In row four, the squares under the 2 and 1 are darkened; the code is 3. Row four of the right block has darkened squares under 8, 4, and 2, so the code is E. Just keep this up, and you will come up with the following:

```
CALL CHAR(100,"00001C3E7F7F7F7F")
CALL CHAR(101,"0000387CFEFEFEFE")
CALL CHAR(102,"3F1F0F0703010000")
CALL CHAR(103,"FCF8F0E0C0800000")
```

Using base 16 is easier.

Animating TI Displays Without Sprites

Jim Schlegel

Fast animation is possible with TI BASIC through efficient coding and the use of a few tricks. "Marbles," a game written in TI BASIC, demonstrates some of these techniques.

Sprites can be used to create very smooth moving animation. The problem with sprites is that they require the Extended BASIC module. If you don't already have Extended BASIC, it can be a very difficult item to find. It's possible, though, to write animated games using just TI BASIC.

BASIC's CALL and the Hardware

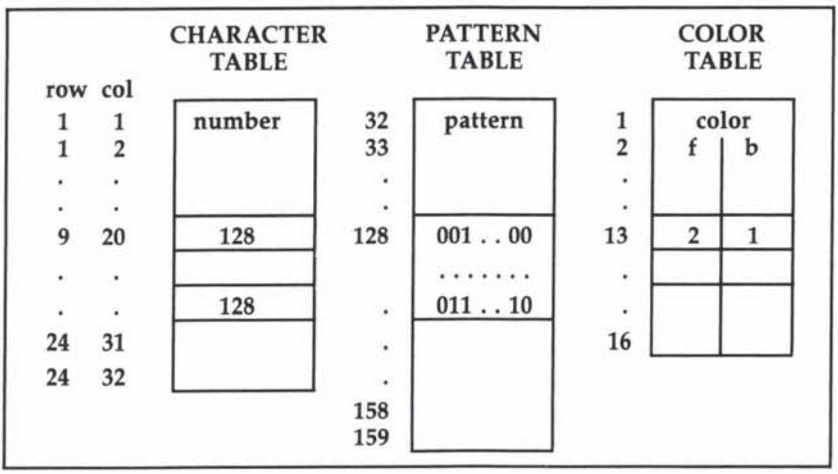
When writing animated programs for the TI-99/4A home computer, an understanding of its architecture will lead to easier coding and faster program execution. In particular, the relationship between the TI's display hardware and the BASIC language CALL instructions used to control this hardware is important. The 99/4A uses Texas Instruments's TMS9918 video display processor to generate the screen display. The display processor functions independently of the TMS9900, the 16-bit microprocessor used in the 99/4A, but is controlled by the TMS9900. This removes the job of generating the display from the microprocessor, allowing it to execute the BASIC program faster.

The TMS9918 allows more flexible displays than the owner of the TI has access to without purchasing additional software modules. Sprites are 8 × 8, 16 × 16, or 32 × 32 pixel patterns created and controlled by the Extended BASIC program. A pixel is the smallest point that can be changed on the display. The sprites are then moved by the TMS9918 independent of, but under control of, the BASIC program. Animated displays can be created without sprites, but it takes a little more work. Here is where the knowledge of the TMS9918 architecture comes in handy!

The display created by the TMS9918 is controlled by three tables which are modified by the BASIC program. These tables and their interrelationships are shown in Figure 1.

Figure 1. TI-99/4A Display Mapping

These tables control the display generated by the TMS9918 video display processor.



Character Table

The first table, the Character Table, is a list of the 768 characters (24 rows by 32 columns) to be displayed. The numbers stored in this table represent the characters to be displayed at each row and column position. The letter A is represented by the number 65, B by 66, C by 67, etc. Numbers 32–127 are defined by the ASCII character set but can be redefined by the BASIC program. ASCII characters 128–159 are also available for defining special characters. This table is accessed by four CALL instructions:

CALL CLEAR
CALL HCHAR(row, column, character[,repetitions])
CALL VCHAR(row, column, character[,repetitions])
CALL GCHAR(row, column, character)

CALL CLEAR sets all numbers in the table to 32 (a space character). CALL HCHAR and CALL VCHAR are used to put numbers into the Character Table while CALL GCHAR is used to get numbers from the table. Note that the repetitions argument for the CALL HCHAR and CALL VCHAR instructions is optional. If this argument is omitted, one character is written to the position defined by the row and column arguments. If this argument is used, a row or column of characters is written to the display. The argument "repetitions" defines the length of the row or column. For example, CALL HCHAR(1,1,65,10) will print ten letter A's horizontally starting at row one, column one.

Pattern Table

The second table, the Pattern Table, is a list of 128 8-byte character patterns. The first entry in the list represents the pattern for character number 32, the second entry is for character number 33, and so on. The last pattern, entry 128, represents the pattern for character number 159. Each character is an 8 × 8 pixel, 2-color pattern where each 1-pixel represents the foreground color and each 0-pixel represents the background color. This table is modified by one CALL instruction:

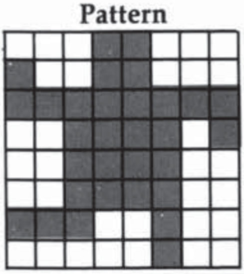
CALL CHAR(character, pattern)

CALL CHAR defines which pixels are to be displayed as the foreground color and which are to be displayed as the background color. An example of a CALL CHAR instruction is shown in Figure 2.

Figure 2. CALL CHAR Instruction

CALL CHAR(128,"1898FF3D3C3CE404")

Pattern



Binary	Hexadecimal
0001 1000	18
1001 1000	98
1111 1111	FF
0011 1101	3D
0011 1100	3C
0011 1100	3C
1110 0100	E4
0000 0100	04

Color Table

The color table is a list of 16 foreground and background color combinations to be used when displaying the characters. The characters defined in the Pattern Table are arranged in sets of eight for determining which colors to use. The first eight characters use the first foreground/background color combination, the second eight characters use the second combination, etc. This table is modified by the CALL COLOR instruction:

CALL COLOR(set, foreground-color, background-color)

Fifteen colors plus transparency are available. Any combination of these colors can be selected by the CALL COLOR instruction.

Table 1. Colors Available on the TI-99/4A

Number	Color	Number	Color
1	Transparent	9	Medium red
2	Black	10	Light red
3	Medium green	11	Dark yellow
4	Light green	12	Light yellow
5	Dark blue	13	Dark green
6	Light blue	14	Magenta
7	Dark red	15	Gray
8	Cyan	16	White

Creating Animation

Most computer games use animated players to liven up the action during play. To do this, the program running the game must change the pattern of the player to make them move. *Munchman* and *TI Invaders* are good examples of games using animated players. Two or more patterns representing different positions of the player are built using the CALL CHAR instruction. The patterns are then alternately displayed creating animation. Also, using and changing colors can add to the effect of animation.

By using the BASIC instructions for creating displays, several different methods can be used to create the same display. Some methods, however, are preferable because they are easier to write and run faster. The faster a program can run, the better the animated display will be.

Many games display the same type of player several times and move each of these players simultaneously. TI Invaders is a good example. Several rows of about ten aliens move about, each moving its legs and/or arms. Each row is made of only one type of alien; all of the aliens in a row move their arms and legs the same way. This type of animation can be created two different ways on the 99/4A.

Both methods will use a common subroutine to animate the players:

```
800 REM * N = Number of Players
810 REM * RP = Array of Row Positions of Players
820 REM * CP = Array of Col Positions of Players
830 REM * C = Character Number of Player Pattern
840 FOR I = 1 TO N
850 CALL HCHAR(RP(I),CP(I),C)
860 NEXT I
870 RETURN
```

The first method uses this subroutine when the player changes their row and column positions and when the players move their arms and/or legs:

```
100 REM * Define Player Patterns
110 CALL CHAR(128,"1898FF3D3C3CE404")
120 CALL CHAR(129,"1819FFBC3C3C2720")
 .
 .
330 REM * Erase Players
340 C=32
350 GOSUB 800
360 REM * Calculate New Rows/Cols
 .
 .
400 REM * Display New Positions
410 C=128
420 GOSUB 800
 .
 .
520 REM * Move Arms/Legs
530 C=129
540 GOSUB 800
 .
 .
610 C=128
620 GOSUB 800
 .
 .
680 IF whatever THEN 520
690 GOTO 330
```

The second method uses subroutine 800 only to change the row and column positions of the players. To move the arms and legs, the character pattern defining the player is changed. Lines 120, 530, and 610 are deleted, lines 540 and 620 are changed to:

```
540 CALL CHAR(128,"1898FF3D3C3CE404")
620 CALL CHAR(128,"1819FFBC3C3C2720")
```

In the first method, characters 128 and 129 are used, while in the second method only character 128 is used. Referring back to Figure 1, the differences in these methods can be seen. Method one changes the Character Table while method two changes the Pattern Table when moving the arms and legs. Method one changes each player's location in the Character Table to point to a new pattern entry in the Pattern Table. Method two just changes the pattern. If ten players were displayed, method one would execute 66 instructions to move the arms and legs while method two would execute only 2 instructions. Method one uses so many more instructions because the loop in subroutine 800 must be executed once for each player.

Using Color

In addition to moving players to create animation, changing colors adds to the visual effect. Again, different approaches will produce the same display but the programming and execution time will vary. The CALL COLOR instruction lets the program change the foreground/background color combination for any character. It's important to remember that each CALL COLOR changes colors for eight character patterns. Care must be used to insure that players and objects are grouped properly for coloring.

Making players and objects appear and disappear can be accomplished three different ways.

First, move the character number of the player or object to the Character Table to make it appear. Overwriting the player or object with a space character would make it disappear. If several players/objects needed to be changed, this would mean executing many instructions.

Second, the CALL CHAR instruction could be used to change the Pattern Table to create this effect. Setting all the pixels in the pattern to 0 would make the object disappear.

Defining the object pattern would make the object reappear. This requires execution of only one instruction.

Third, the CALL COLOR instruction could be used to change the Color Table. By defining both the foreground and background colors the same, the object is no longer visible. If the object is on a game board, the color of the board should be used. Setting both the foreground and background colors to transparent (1), the color defined by the CALL SCREEN instruction would be used. One advantage of using the CALL COLOR instruction is that up to eight distinct objects could be made to appear and disappear with one instruction, while the CALL CHAR instruction would have to be executed once for each distinct object. A single object composed of up to eight character patterns could be changed with a single CALL COLOR instruction.

Example Animated Program

The following BASIC program uses the techniques described in this article to produce an animated game. The object of the game is to maneuver the marble into the hole at the opposite corner of the display. Between the marble and the hole are two to five kids trying to catch the marble. The kids can only be seen at the start of the game or when one is close to the marble. The arrow keys on the keyboard are used to maneuver the marble.

Marble

```
100 REM *
110 REM * DEFINE PLAYERS
120 REM *
130 BGC=8
140 SQUARE=128
150 SQR$="0000000000000000"
160 CALL CHAR(SQUARE,SQR$)
170 CALL COLOR(13,1,BGC)
180 KID=136
190 KD1$="1898FF3D3C3CE404"
200 KD2$="1819FFBC3C3C2720"
210 CALL CHAR(KID,KD1$)
220 CALL COLOR(14,2,BGC)
230 MARBLE=144
240 MRB$="003C7E7E7E7E3C00"
250 CALL CHAR(MARBLE,MRB$)
260 CALL COLOR(15,16,BGC)
```

```
270 HOLE=152
280 HOL$="FFC381818181C3FF"
290 CALL CHAR(HOLE,HOL$)
300 CALL COLOR(16,2,1)
310 REM *
320 REM * DISPLAY BOARD
330 REM *
340 CALL CLEAR
350 CALL SCREEN(10)
360 C=7
370 L=20
380 FOR R=3 TO 22
390 CALL HCHAR(R,C,SQUARE,L)
400 NEXT R
410 REM *
420 REM * POSITION KIDS{3 SPACES}
430 REM *
440 DIM KR(10),KC(10)
450 RANDOMIZE
460 KN=INT(4*RND)+2
470 FOR N=1 TO KN
480 KR(N)=INT(20*RND)+3
490 KC(N)=INT(20*RND)+7
500 CALL HCHAR(KR(N),KC(N),KID)
510 NEXT N
520 REM *
530 REM * POSITION HOLE{3 SPACES}
540 REM *
550 HR=4
560 HC=8
570 CALL HCHAR(HR,HC,HOLE)
580 REM *
590 REM * POSITION MARBLE
600 REM *
610 MR=21
620 MC=25
630 CALL HCHAR(MR,MC,MARBLE)
640 REM *{3 SPACES}
650 REM * WAIT FOR KEY{5 SPACES}
660 REM *{3 SPACES}
670 CALL KEY(1,KEY,STATUS)
680 IF STATUS=0 THEN 670
690 CALL COLOR(14,BGC,BGC)
700 REM *
710 REM * BEGIN GAME
720 REM *
730 CALL CHAR(KID,KD1$)
740 CALL KEY(1,KEY,STATUS)
750 IF STATUS=0 THEN 970
760 J=1
```

```
770 IF STATUS>0 THEN 790
780 J=2
790 IF KEY>5 THEN 970
800 REM *
810 REM * MOVE MARBLE
820 REM *
830 CALL HCHAR(MR,MC,SQUARE)
840 ON KEY+1 GOTO 850,970,870,890,970,910
850 MR=MR+J
860 GOTO 920
870 MC=MC-J
880 GOTO 920
890 MC=MC+J
900 GOTO 920
910 MR=MR-J
920 IF (MR=HR)*(MC=HC)THEN 1210
930 CALL HCHAR(MR,MC,MARBLE)
940 REM *
950 REM * MOVE KIDS
960 REM *
970 CALL CHAR(KID,KD2$)
980 FOR I=1 TO KN
990 CALL HCHAR(KR(I),KC(I),SQUARE)
1000 IF KR(I)=MR THEN 1050
1010 IF KR(I)<MR THEN 1040
1020 KR(I)=KR(I)-1
1030 GOTO 1050
1040 KR(I)=KR(I)+1
1050 IF KC(I)=MC THEN 1100
1060 IF KC(I)<MC THEN 1090
1070 KC(I)=KC(I)-1
1080 GOTO 1100
1090 KC(I)=KC(I)+1
1100 CALL HCHAR(KR(I),KC(I),KID)
1110 IF (KR(I)=MR)*(KC(I)=MC)THEN 1320
1120 R=ABS(KR(I)-MR)
1130 C=ABS(KC(I)-MC)
1140 IF (R+C>4)THEN 1160
1150 CALL COLOR(14,2,BGC)
1160 NEXT I
1170 GOTO 730
1180 REM *
1190 REM * PLAYER WINS
1200 REM *
1210 CALL COLOR(16,2,16)
1220 FOR I=0 TO 1
1230 FOR J=-1 TO -4 STEP -1
1240 CALL SCREEN(I*8-J*2)
1250 CALL SOUND(500,J,1)
1260 NEXT J
```

```
1270 NEXT I
1280 GOTO 100
1290 REM *
1300 REM * PLAYER LOSES
1310 REM *
1320 CALL COLOR(15,7,BGC)
1330 CALL HCHAR(MR,MC,MARBLE)
1340 FOR J=-5 TO -7 STEP -1
1350 CALL SOUND(100,J,1)
1360 NEXT J
1370 GOTO 100
```

SuperFont

Patrick Parrish

A powerful feature of the TI-99/4A is its ability to redefine the character set. With "SuperFont," a comprehensive character definition program, you can harness this capability. Requires Extended BASIC and Memory Expansion.

The character graphics capabilities of the TI-99/4A are well known. To redefine a character on the TI by the usual means (see the *TI User's Reference Guide,* pages II-76 to II-79), a tedious, multistep procedure must be followed. First, you plot the prospective character in an 8 × 8 grid. Next, you convert each row of the grid into a two-digit hexadecimal number and then sequentially combine the numbers from each row to generate a *pattern-identifier,* or coded representation of the character. To complete this task, you place this pattern-identifier along with a chosen ASCII value for the character in a CALL CHAR statement. Anyone who has repeatedly endured this process can attest to its drudgery.

Fortunately, this process is easily computerized, and several character definition programs have been written for the TI. Most character definition programs, though, have not taken full advantage of the TI's capabilities. By using "SuperFont" (Program 1) the task of character manipulations can now be undertaken with ease.

Nineteen Commands

The original SuperFont was written for the Atari by Charles Brannon. The Atari version first appeared in the January 1982 issue of *COMPUTE!* magazine and featured 18 commands for redefining characters. After using this outstanding program on several occasions, I was convinced that the TI user deserved the pleasure and convenience it provided. So, I set about converting the program for the TI.

In converting SuperFont, a few commands with less value to the TI user were eliminated while certain more practical commands were added. The final product offers the following 19 commands or modes:

O DOODLE
E EDIT
N INPUT
R RESTORE CH
H RESTORE CHSET
F COPY
X SWITCH
M MIRROR
V REVERSE
A ROTATE
C CLEAR
I INSERT
D DELETE
W WRITE DATA
Y QUIT
L LOAD FONT
S SAVE FONT
P PRINT CH
T PRINT CHSET

When the program is run, these commands are displayed in menu form on the screen. Above the menu is an 8 × 8 grid which serves as a work space for redefining each character. To the right of the grid, the current mode and, in some cases, a prompt will be displayed. Below this is printed the entire TI character set (codes 32–143) with each color subset (eight characters) depicted by a different background color. (The colors can be toggled off and on with the Z key.)

Several commands require that you pick a character from the character set. In these instances, a box-shaped sprite, CHR$(143), will appear over the last character referenced from the set (defaults to space). To choose a character move the joystick over the desired character and press the fire button.

Unless indicated otherwise, each command will return you to the EDIT mode upon completion. Let's now examine each command beginning with EDIT (the ALPHA-LOCK key should be up).

EDIT is the basic editing command. When selected from the menu, you will be requested to choose a character from the character set. The character selected is copied into the grid and the box-shaped sprite will be homed in the grid. Move this sprite about the grid with the joystick. Pressing the fire button will set or clear the point depending on its present state. You can draw lines by holding down the button while moving the joystick. When you're pleased with the appearance of the character in the grid, press ENTER to redefine the chosen character. You'll then be prompted for another command. To completely redesign a character from scratch, use the CLEAR command.

INPUT lets you type in a pattern-identifier and assign it to a particular character code. When INPUT is selected, choose a replaceable character from the set with the joystick and then type in the hexadecimal code for the proposed character. The hexadecimal code can be typed in upper- or lowercase. A routine at line 1260 automatically converts the code to uppercase. The INPUT command is handy when attempting to associate a pattern-identifier with a CHR$ in someone's BASIC code.

RESTORE CH restores the current character to its original configuration. This command is useful if you wish to start over defining a character or if you changed the wrong one.

RESTORE CHSET restores the entire character set to its initial appearance.

COPY copies a character to a second location in the character set. You will be prompted for the first (character to be moved) and second (destination) character. This command is handy for arranging your customized characters to fit the various color codes.

SWITCH swaps the location of two characters in the set. As with COPY, you will be prompted for two characters.

MIRROR produces a mirror image of the current character in the grid.

REVERSE puts the current character in the grid in reverse field: all dots become blanks, and all blanks become dots.

ROTATE turns the current character 90 degrees clockwise.

CLEAR completely clears out the current character.

INSERT places a row of blanks in the current character. Move the sprite in the grid with the joystick to the row where you wish to insert the blanks and press ENTER. All rows below that will scroll down and the bottom row will be lost.

DELETE is the opposite of INSERT. Position the sprite on a row in the grid and press ENTER. The row will be eliminated and all other rows will scroll upward. DELETE and INSERT can be used in conjunction with ROTATE to scroll characters left or right in the grid (of course, one row will be lost in both cases).

WRITE DATA displays the pattern-identifier for each selected character along with its ASCII value. When finished, a prompt for another command will be issued. This is handy when comparing characters or for providing a few character codes for another program.

QUIT simply terminates the program.

LOAD FONT loads a previously SAVEd character set (a font) from tape or disk. You will be prompted for the device and filename. Be sure that this is typed in the standard format (CS1 or DSK1.FILENAME). Again, capital letters need not be used. The routine that converts from lower- to uppercase lettering takes care of this for you. If you're using a cassette, the screen will be restored after the tape system messages have been printed (the same occurs with SAVE FONT discussed below). When loading is complete, a command prompt is given.

SAVE FONT saves to tape or disk in a data file format only those characters in the set which have been altered since the program was run. Since each character code is saved as a separate record, you may need 30 minutes of tape to save a large set if you use cassette. As with LOAD FONT, you will be prompted for the device and filename. If you accidentally hit L (for LOAD FONT) or S from the main menu, simply press ENTER to abort the errant command when prompted for the device and filename.

Once saved, character sets can be loaded into any program where they're needed (we'll consider this in greater detail shortly). As with LOAD FONT, you'll see a prompt for another command when the SAVE is complete.

PRINT CH prints the current character in an 8 × 8 grid along with its ASCII and pattern-identifier codes, then returns you to the main menu. *Be sure that you modify line 1660 to correspond to the specifications of your printer.*

PRINT CHSET is the same as the previous command except that it prints every character which has been modified.

Just For the Fun of It

The first command in the menu, which we overlooked until now, is the *DOODLE* mode. By choosing this mode, you can use your redefined character set to design a playfield or simply draw for the fun of it. Your completed playfield or drawing can even be saved and loaded back in from tape or disk for further modification.

After redefining some characters, go into the DOODLE mode by typing O. The screen will clear except for the character set at the bottom. The following one-line menu will be displayed at the top of the screen:

C F B M=MENU L S=SAVE

First select the character you wish to locate somewhere on the screen by positioning the box-like sprite with the joystick over this character and pressing the fire button. The chosen character will become a sprite and automatically scroll up to the row above the displayed character set. You can move this character sprite to a desired location with the joystick and print it there by hitting the fire button. If you hold the fire button down while moving the character sprite, a line of characters will be printed.

Now, referring to the above one-line menu, press C to change the screen color, F to change the foreground color of the current character subset, and B to change its background color (as before, all character colors can be toggled off or on with the Z key). When you wish to draw with another character, just press ENTER. The box-like sprite will once again be placed in the character set at the bottom of the screen for another selection. When you've finished drawing, type M to return to the main menu, or if you wish to save the screen (actually, the program saves only rows 2–20), type S. (L lets you load a screen and will wipe out any existing screen.)

Typing L or S while in the DOODLE mode results in a prompt for the device and filename. As with font LOAD and SAVE FONT, carefully type in the device and filename. If you use tape for storage, the screen will be restored (stored in the array Z1) after the tape system messages scroll the screen. If you hit L or S by mistake, just press ENTER to return to the above one-line menu.

When a screen is saved from the DOODLE mode, the screen color, and all foreground and background colors are saved as well.

The commands offered by SuperFont are versatile, but you may want to add others. Since the program is modular in structure (just follow the branching IF statements from line 520 to 1220 for the current commands), you can insert additional command routines following line 1220.

Retrieving a Font or Screen

After you have saved a newly created character set or a set and a screen, how do you go about recovering these for use in another program? Program 2 is a sample program showing how to do this.

Since line 120 dimensions for the screen array (Z1), the foreground colors (FR), and background colors (B), it must be included in your retrieval program. In line 130, the device and filename for the character set file and the screen file are defined as B$ and C$, respectively (the filenames used here are font and screen). If you used tape to store these files, line 130 should read B$,C$="CS1". When loading these from tape, be sure to load them in the proper order.

Lines 140–160 load in the new character set while lines 180–210 load the screen and color codes. In line 220, the screen previously SAVEd from SuperFont is recreated. The delay in line 230 allows you to see it.

If you only wish to retrieve a font, modify lines 120 and 130 to:

120 CALL CLEAR
130 B$="DSK1.FONT"

and eliminate lines 170–220. Of course, you may wish to recover the font along with its foreground and background colors. If so, change line 120 and 190 to:

120 DIM FR(14),B(14) :: CALL CLEAR
190 FOR I=2 TO 20 :: INPUT #1:P$ :: NEXT I

and delete line 220.

A Super Utility

With SuperFont, you can perform many chores with ease. You can customize your character set (ever wished for a true lowercase?), create graphics characters and animated figures (space creatures!), create composite pictures from characters, design playfields, or just play around. The uses of this utility are endless. I'm sure you'll find discovering them as much fun as I have.

Program 1. SuperFont

```
100 !MEMORY EXPANSION REQUIRED
110 DIM A$(111),C$(15),N$(112),D(15),V(8,8),
    FR(14),B(14),Z1(20,32):: L=32
```

```
120 TT=2 :: E=15 :: Q$="DEVICE(DSK1.FILE OR
    CS1)?" :: GOSUB 1630 :: GOTO 410
130 !ERASE
140 F=0 :: GOSUB 150 :: GOTO 490
150 CALL HCHAR(5,14,L,16):: RETURN
160 CALL HCHAR(3,17,L,E):: CALL HCHAR(7,17,L
    ,16):: RETURN
170 FOR I=5 TO 7 :: CALL HCHAR(I,13,L,17)::
    NEXT I :: RETURN
180 CALL HCHAR(8,14,L,E):: CALL HCHAR(20,2,L
    ,27):: RETURN
190 !DISPLAY A GRID CHAR
200 Z$=N$(W-L)
210 FOR I=0 TO 15 :: D(I)=ASC(SEG$(Z$,I+1,1)
    )-48 :: D(I)=D(I)+(D(I)>9)*7
220 NEXT I :: J=0 :: FOR I=0 TO 7 :: DISPLAY
     AT(2+I,1):C$(D(J));:: DISPLAY AT(2+I,5)
    :C$(D(J+1));:: J=J+2 :: NEXT I :: RETURN
230 !CONVERT GRID PAT TO HEX STRING
240 CALL DELSPRITE(#1):: DISPLAY AT(5,15):"P
    LEASE WAIT"
250 FOR R=1 TO 8 :: FOR C=1 TO 8
260 IF M=109 THEN CALL GCHAR(R+1,11-C,H):: G
    OTO 290
270 IF M=97 THEN CALL GCHAR(10-C,R+2,H):: GO
    TO 290
280 CALL GCHAR(R+1,2+C,H)
290 V(R,C)=H-141 :: NEXT C :: NEXT R
300 H$="0123456789ABCDEF" :: IF M=118 THEN H
    $="FEDCBA9876543210"
310 Z$="" :: FOR R=1 TO 8 :: LO=V(R,5)*8+V(R
    ,6)*4+V(R,7)*2+V(R,8)+1
320 HI=V(R,1)*8+V(R,2)*4+V(R,3)*2+V(R,4)+1
330 Z$=Z$&SEG$(H$,HI,1)&SEG$(H$,LO,1):: NEXT
     R
340 IF (M<>100)*(M<>105)THEN 390
350 IF M<>100 THEN 370
360 Z$=SEG$(Z$,1,ROW*2-2)&SEG$(Z$,ROW*2+1,14
    )&"00" :: GOTO 380
370 Z$=SEG$(Z$,1,ROW*2-2)&"00"&SEG$(Z$,ROW*2
    -1,16-ROW*2)
380 CALL CHAR(W,Z$):: N$(W-L)=Z$ :: IF (M=10
    0)+(M=105)THEN GOSUB 200
390 GOSUB 150 :: RETURN
400 !CREATE BLOCK CODES
410 F$="0000000100100011010001010110011110000
    1001101010111100110111101111"
420 FOR I=0 TO 15 :: Z$=SEG$(F$,I*4+1,4):: D
    $=""
```

```
430 FOR J=1 TO 4 :: T=VAL(SEG$(Z$,J,1))+141
    :: D$=D$&CHR$(T):: NEXT J :: C$(I)=D$ ::
     NEXT I
440 CALL CHAR(141,"",142,RPT$("F",16),143,"F
    F818181818181FF"):: FOR I=141 TO 143 ::
    CALL CHARPAT(I,A$(I-L)):: N$(I-L)=A$(I-L
    ):: NEXT I
450 CALL DELSPRITE(#1):: CALL CLEAR :: FOR I
    =2 TO 14 :: FR(I)=2 :: B(I)=I+2 :: CALL
    COLOR(I,2,I+2):: NEXT I :: FR(1)=2 :: B(
    1)=1
460 FOR I=L TO 143 :: PRINT CHR$(I);:: NEXT
    I :: DISPLAY AT(1,11):"SUPERFONT" :: GOS
    UB 1420 :: IF W5=1 THEN CALL COLOR(14,2,
    16)
470 FOR R=1 TO 8 :: CALL HCHAR(R+1,3,141,8):
    : NEXT R
480 BR=20 :: BC=2 :: W=L
490 CALL SOUND(100,800,2):: DISPLAY AT(3,15)
    :"WHICH MODE?"
500 CALL KEY(0,M,S):: IF S=0 THEN 500
510 IF M<>122 THEN 520 ELSE GOSUB 1780 :: GO
    TO 490
520 IF M<>101 THEN 670
530 D$="EDIT MODE" :: T=1 :: GOSUB 1580 :: G
    OSUB 1290 :: IF (F=1)*(K<>112)THEN 140 E
    LSE IF K=112 THEN M=K :: GOSUB 150 :: GO
    TO 1200
540 GOSUB 200 :: Z=1
550 CALL SPRITE(#1,143,10,9,17):: R=1 :: C=2
     :: CALL GCHAR(R+1,C+1,CAR)
560 CALL KEY(0,K,S):: IF (K=13)+(K=112)THEN
    ROW=R :: GOSUB 240 :: GOSUB 1540 :: IF K
    <>112 THEN ON Z GOTO 490,760
570 IF (K>13)*(K<>122)THEN M=K :: GOTO 520 E
    LSE IF K=122 THEN GOSUB 1780
580 CALL JOYST(1,X,Y):: IF ABS(X)+ABS(Y)=8 T
    HEN 580
590 CALL KEY(1,KK,S):: IF (KK<>18)*(ABS(X)+A
    BS(Y)=0)THEN 560
600 OK=0 :: IF ABS(X)+ABS(Y)=4 THEN OK=1
610 C=C-(X=4)+(X=-4):: R=R-(Y=-4)+(Y=4)
620 C=C-(C=1)*8+(C=10)*8 :: R=R-(R=0)*8+(R=9
    )*8
630 CALL LOCATE(#1,8*R+1,8*C+1)
640 IF (KK=18)*(OK=0)THEN CALL GCHAR(R+1,C+1
    ,CAR):: CAR=283-CAR
650 IF (OK=1)*(KK<>18)THEN CALL GCHAR(R+1,C+
    1,CAR)
```

```
660 CALL HCHAR(R+1,C+1,CAR):: CALL SOUND(-1,
    294,3):: GOTO 560
670 IF M<>110 THEN 740
680 T=1 :: D$="INPUT MODE" :: GOSUB 1580 ::
    GOSUB 1290 :: IF F=1 THEN 140
690 IF W5=0 THEN CALL COLOR(3,2,15,4,2,15,9,
    2,15)
700 DISPLAY AT(5,12):"CHAR HEX CODE?" :: ACC
    EPT AT(6,11)SIZE(16)BEEP:D$ :: IF LEN(D$
    )<>16 THEN 700
710 GOSUB 170 :: GOSUB 1260
720 N$(W-L)=Z$ :: GOSUB 210 :: CALL CHAR(W,Z
    $)
730 GOSUB 150 :: IF W5=0 THEN CALL COLOR(3,F
    R(3),B(3),4,FR(4),B(3),9,FR(9),B(9)):: G
    OTO 760 ELSE 760
740 IF M<>114 THEN 770
750 D$="RESTORE CHAR" :: GOSUB 1580 :: CALL
    CHAR(W,A$(W-L)):: N$(W-L)=A$(W-L)
760 Z=1 :: GOSUB 150 :: GOSUB 200 :: M=101 :
    : GOSUB 1540 :: DISPLAY AT(3,15):"EDIT M
    ODE" :: CALL SOUND(50,880,3):: GOTO 550
770 IF M<>104 THEN 810
780 D$="RESTORING SET" :: GOSUB 1580
790 DISPLAY AT(5,15):"PLEASE WAIT"
800 FOR I=L TO 143 :: CALL CHAR(I,A$(I-L))::
     N$(I-L)=A$(I-L):: NEXT I :: GOTO 760
810 IF M<>102 THEN 860
820 D$="COPY MODE" :: GOSUB 1580
830 DISPLAY AT(5,15):"FIRST CHAR?" :: GOSUB
    1290 :: IF F=1 THEN 140 ELSE TM=W
840 GOSUB 200 :: DISPLAY AT(5,15):"SECOND CH
    AR?" :: GOSUB 1290 :: IF F=1 THEN 140 EL
    SE CALL DELSPRITE(#1)
850 CALL CHARPAT(TM,Z$):: CALL CHAR(W,Z$)::
    N$(W-L)=Z$ :: GOTO 760
860 IF M<>120 THEN 920
870 D$="SWITCH MODE" :: GOSUB 1580
880 DISPLAY AT(5,15):"FIRST CHAR?" :: GOSUB
    1290 :: IF F=1 THEN 140 ELSE TM=W
890 GOSUB 200 :: DISPLAY AT(5,15):"SECOND CH
    AR?" :: GOSUB 1290 :: IF F=1 THEN 140 EL
    SE TM2=W :: CALL DELSPRITE(#1)
900 CALL CHARPAT(TM,D$):: CALL CHARPAT(TM2,F
    $):: CALL CHAR(TM2,D$):: CALL CHAR(TM,F$
    )
910 N$(TM-L)=F$ :: N$(TM2-L)=D$ :: GOTO 760
920 IF M<>109 THEN 940
```

```
930 D$="MIRROR MODE" :: GOSUB 1580 :: GOSUB
    240 :: GOTO 760
940 IF M<>118 THEN 960
950 D$="REVERSE MODE" :: GOSUB 1580 :: GOSUB
     240 :: GOTO 760
960 IF M<>97 THEN 1000
970 D$="ROTATE MODE" :: GOSUB 1580
980 GOSUB 240 :: GOSUB 200 :: GOSUB 1540 ::
    T=0 :: D$="AGAIN (Y/N)?" :: GOSUB 1600 :
    : GOSUB 150 :: IF T=1 THEN 980
990 GOTO 760
1000 IF M=99 THEN D$="CLEAR MODE" :: GOSUB 1
     580 :: D$=RPT$("0",16):: CALL CHAR(W,D$
     ):: N$(W-L)=D$ :: GOTO 760
1010 IF M=105 THEN D$="INSERT MODE" :: GOSUB
      1580 :: Z=2 :: GOTO 550
1020 IF M=100 THEN D$="DELETE MODE" :: GOSUB
      1580 :: Z=2 :: GOTO 550
1030 IF M<>119 THEN 1100
1040 IF W5=0 THEN CALL COLOR(3,2,15,4,2,15,5
     ,2,15)
1050 D$="WRITE MODE" :: T=1 :: GOSUB 1580 ::
      GOSUB 1290 :: IF F=1 THEN F=0 :: GOTO
     1090 ELSE GOSUB 200
1060 DISPLAY AT(7,16):"CHAR=";W :: DISPLAY A
     T(9,11):N$(W-L)
1070 D$="AGAIN(Y/N) ?" :: GOSUB 1600
1080 CALL HCHAR(9,11,L,18):: IF T=1 THEN GOS
     UB 150 :: GOTO 1050
1090 GOSUB 170 :: IF W5=0 THEN CALL COLOR(3,
     FR(3),B(3),4,FR(4),B(4),5,FR(5),B(5))::
      GOTO 490 ELSE 490
1100 IF M=121 THEN STOP
1110 IF M<>108 THEN 1150
1120 D$="LOAD FONT" :: GOSUB 1580
1130 GOSUB 1230 :: OPEN #1:D$,INTERNAL,INPUT
      ,FIXED
1140 INPUT #1:T,N$(T):: IF T<>112 THEN CALL
     CHAR(T+L,N$(T)):: GOTO 1140 ELSE CLOSE
     #1 :: GOSUB 180 :: IF ASC(D$)=67 THEN 4
     50 ELSE 490
1150 IF M<>115 THEN 1200
1160 D$="SAVE FONT" :: GOSUB 1580 :: GOSUB 1
     230
1170 OPEN #1:D$,INTERNAL,OUTPUT,FIXED :: FOR
      I=L TO 143
1180 IF N$(I-L)<>A$(I-L)THEN PRINT #1:I-L,N$
     (I-L)
```

```
1190 NEXT I :: T=112 :: F$="" :: PRINT #1:T,
     F$ :: CLOSE #1 :: GOSUB 180 :: IF ASC(D
     $)=67 THEN 450 ELSE 490
1200 IF M=112 THEN H=1 :: GOSUB 1660 :: GOTO
      490
1210 IF M=116 THEN H=0 :: GOSUB 1660 :: GOTO
      490
1220 IF M<>111 THEN 490 ELSE CALL DELSPRITE(
     #1):: GOSUB 1850 :: GOTO 490
1230 DISPLAY AT(20,2):Q$ :: ACCEPT AT(8,14):
     D$ :: IF D$="" THEN GOSUB 180 :: GOTO 4
     90 ELSE GOSUB 1260
1240 RETURN
1250 !CONVERT TO CAPS
1260 Z$="" :: FOR I=1 TO LEN(D$):: F$=SEG$(D
     $,I,1):: IF (ASC(F$)>96)*(ASC(F$)<123)T
     HEN F$=CHR$(ASC(F$)-L)
1270 Z$=Z$&F$ :: NEXT I :: D$=Z$ :: RETURN
1280 !GET CHAR
1290 CALL SPRITE(#1,143,FR(14),BR*8+1,BC*8+1
     )
1300 CALL JOYST(1,X,Y):: IF ABS(X)+ABS(Y)=8
     THEN 1300
1310 BC=BC-(X=4)+(X=-4):: W=W-(X=4)+(X=-4)
1320 BR=BR-(Y=-4)+(Y=4):: W=W-(Y=-4)*28+(Y=4
     )*28
1330 IF BC<2 THEN BC=29 :: BR=BR-1
1340 IF BC>29 THEN BC=2 :: BR=BR+1
1350 IF BR<20 THEN BR=23 :: W=W+112
1360 IF BR>23 THEN BR=20 :: W=W-112
1370 CALL KEY(1,KK,ST):: CALL KEY(0,K,S):: I
     F K=122 THEN GOSUB 1780 :: GOTO 1290
1380 IF S<>0 THEN F=1 :: IF M=111 THEN RETUR
     N ELSE CALL DELSPRITE(#1):: RETURN
1390 IF KK=18 THEN CALL SOUND(10,110,2):: IF
      M=111 THEN RETURN ELSE GOSUB 150 :: CA
     LL DELSPRITE(#1):: RETURN
1400 GOTO 1290
1410 !MENU
1420 DISPLAY AT(10,14):"O DOODLE"
1430 DISPLAY AT(11,1):"E EDIT";TAB(14):"N IN
     PUT"
1440 DISPLAY AT(12,1):"R RESTORE CH";TAB(14)
     :"H RESTORE CHSET"
1450 DISPLAY AT(13,1):"F COPY";TAB(14);"X SW
     ITCH"
1460 DISPLAY AT(14,1):"M MIRROR";TAB(14);"V
     REVERSE"
```

```
1470 DISPLAY AT(15,1):"A ROTATE";TAB(14);"C
     CLEAR"
1480 DISPLAY AT(16,1):"I INSERT":TAB(14);"D
     DELETE"
1490 DISPLAY AT(17,1):"W WRITE DATA";TAB(14)
     :"Y QUIT"
1500 DISPLAY AT(18,1):"L LOAD FONT";TAB(14);
     "S SAVE FONT"
1510 DISPLAY AT(19,1):"P PRINT CH";TAB(14):"
     T PRINT CHSET"
1520 RETURN
1530 !DRAW A FEW CHARS
1540 FOR I=0 TO 5 STEP 2 :: CALL HCHAR(7,17+
     I,W):: NEXT I :: RETURN
1550 !POSITION CURSOR
1560 R=20 :: C=2 :: W=L :: CALL SPRITE(#1,14
     3,2,R*8+1,C*8+1):: RETURN
1570 !DISPLAY MODE
1580 GOSUB 160 :: DISPLAY AT(3,15):D$ :: IF
     T=1 THEN DISPLAY AT(5,15):"CHOOSE A CHA
     R" :: T=0
1590 RETURN
1600 DISPLAY AT(5,15):D$ :: ACCEPT AT(5,27)B
     EEP VALIDATE("yn")SIZE(1):Z$ :: IF Z$="
     y" THEN T=1
1610 RETURN
1620 !SAVE ORIG CHAR PATS
1630 CALL CLEAR :: CALL SCREEN(E):: DISPLAY
     AT(10,8):"...PATIENCE..." :: DISPLAY AT
     (12,2):"LOADING CHARACTER PATTERNS"
1640 DISPLAY AT(23,1):"(ALPHA-LOCK KEY MUST
     BE UP)" :: FOR I=127 TO 140 :: CALL CHA
     R(I,""):: NEXT I
1650 FOR I=L TO 140 :: CALL CHARPAT(I,A$(I-L
     )):: N$(I-L)=A$(I-L):: NEXT I :: RETURN
1660 DISPLAY AT(3,15):"PRINT MODE" :: OPEN #
     1:"RS232/2.BA=9600.DA=8.PA=N"
1670 TM=W :: IF H=1 THEN 1700
1680 FOR T=L TO 143 :: IF N$(T-L)<>A$(T-L)TH
     EN W=T ELSE 1750
1690 E=E+1 :: E=(E=17)*14+E :: CALL SCREEN(E
     )
1700 IF ((F=1)*(H=1))+(H=0)THEN GOSUB 200 ::
      GOSUB 1540
1710 FOR R=2 TO 9 :: IF R=5 THEN PRINT #1:TA
     B(5);"CHR$ # - "&"<"&STR$(W)&">";
1720 PRINT #1:TAB(30);:: FOR C=3 TO 10 :: CA
     LL GCHAR(R,C,X):: IF X=141 THEN X=45 EL
     SE X=88
```

```
1730 PRINT #1:CHR$(X);:: NEXT C :: IF R=5 TH
     EN PRINT #1:TAB(47);"HEX CODE - "&"<"&N
     $(W-L)&">"
1740 NEXT R :: PRINT #1 :: PRINT #1 :: IF H=
     1 THEN 1760
1750 NEXT T
1760 CLOSE #1 :: F=0 :: H=0 :: E=15 :: W=TM
     :: CALL SCREEN(E):: RETURN
1770 !TOGGLE COLORS
1780 FOR I=1 TO 14 :: IF W5=0 THEN FORE=2 ::
      BACK=1 ELSE BACK=B(I):: FORE=FR(I)
1790 IF (I<>14)THEN 1820 ELSE IF ((K=122)*(M
     <>111)*(W5=0))+((M=122)*(W5=0))THEN BAC
     K=16
1800 IF M=111 THEN IF (W5=0)THEN TT=FR(14)::
      FR(14)=FORE ELSE FR(14)=TT :: FORE=TT
1810 IF M=111 THEN CALL COLOR(#1,FORE)
1820 CALL COLOR(#2,FORE):: CALL COLOR(I,FORE
     ,BACK):: NEXT I :: W5=-(W5=0):: IF (M=1
     22)+((K=122)*(M<>111))THEN RETURN
1830 I=INT(W/8)-3 :: FORE=-(W5=1)*2-(W5=0)*F
     R(I):: RETURN
1840 !DOODLE
1850 FOR J=1 TO 15 :: CALL VCHAR(1,J,L,19)::
      CALL VCHAR(1,31-J,L,19):: NEXT J :: IF
      W5=1 THEN CALL COLOR(14,2,1)
1860 DISPLAY AT(1,1):"C  F  B  M=MENU  L  S=
     SAVE";
1870 W=L :: BR=20 :: BC=2 :: GOSUB 1830 :: G
     OSUB 1290
1880 GOSUB 1830 :: IF F=1 THEN 1960 ELSE BAC
     K=-(W5=1)-(W5=0)*B(I)
1890 CALL SPRITE(#2,W,FORE,BR*8+1,BC*8+1)
1900 CALL JOYST(1,X,Y):: BR=BR-Y/4 :: BC=BC+
     X/4
1910 BR=BR-(BR=0)+(BR>19):: BC=BC-(BC=0)+(BC
     =31)
1920 CALL LOCATE(#1,BR*8+1,BC*8+1,#2,BR*8+1,
     BC*8+1)
1930 CALL KEY(1,KK,S):: IF (KK<>18)+(BR>19)T
     HEN 1950
1940 CALL HCHAR(BR+1,BC+1,W):: CALL SOUND(10
     ,110,2):: GOTO 1900
1950 CALL KEY(0,K,S)
1960 IF K=109 THEN CALL DELSPRITE(#1,#2):: C
     ALL CLEAR :: CALL SCREEN(E):: F=0 :: GO
     TO 460
1970 IF K=13 THEN CALL DELSPRITE(#2):: GOTO
     1870
```

```
1980 IF K=122 THEN GOSUB 1780 :: GOTO 1900
1990 IF K=115 THEN 2140
2000 IF K=108 THEN 2240
2010 IF (K=98)+(K=99)+(K=102)THEN GOSUB 2060
2020 IF F=1 THEN F=0 :: GOSUB 1300 :: GOTO 1
     880 ELSE GOTO 1900
2030 GOTO 1880
2040 CALL COLOR(I,FR(I),B(I))
2050 RETURN
2060 IF W5=1 THEN 2110
2070 IF K<>98 THEN 2090 ELSE B(I)=B(I)+1 ::
     B(I)=B(I)+(B(I)>16)*16
2080 GOTO 2040
2090 IF K<>102 THEN 2110 ELSE FR(I)=FR(I)+1
     :: FR(I)=FR(I)+(FR(I)>16)*16
2100 GOSUB 1830 :: CALL COLOR(#2,FORE):: IF
     I=14 THEN CALL COLOR(#1,FORE):: GOTO 20
     40 ELSE 2040
2110 IF K<>99 THEN 2050 ELSE E=E+1 :: E=E+(E
     >16)*15
2120 CALL SCREEN(E):: GOTO 2050
2130 !SAVE SCREEN & COLORS
2140 CALL DELSPRITE(ALL):: GOSUB 2280 :: DIS
     PLAY AT(1,1):"PUTTING SCREEN IN ARRAY"
2150 FOR I=2 TO 20 :: FOR J=1 TO L :: CALL G
     CHAR(I,J,Z1(I,J)):: NEXT J :: NEXT I ::
      CALL HCHAR(1,1,L,25)
2160 OPEN #1:D$,INTERNAL,OUTPUT,FIXED
2170 FOR I=2 TO 20 :: P$="" :: FOR J=1 TO L
     :: P$=P$&CHR$(Z1(I,J)):: NEXT J :: PRIN
     T #1:P$ :: NEXT I
2180 P$=CHR$(E):: PRINT #1:P$ :: P$="" :: FO
     R I=1 TO 14 :: P$=P$&CHR$(FR(I))&CHR$(B
     (I)):: NEXT I :: PRINT #1:P$
2190 CLOSE #1 :: IF ASC(D$)<>67 THEN 1860
2200 CALL CLEAR :: FOR I=L TO 143 :: PRINT C
     HR$(I)::: NEXT I
2210 FOR I=2 TO 20 :: FOR J=1 TO L :: CALL H
     CHAR(I,J,Z1(I,J)):: NEXT J :: NEXT I
2220 GOTO 1860
2230 !LOAD SCREEN
2240 CALL DELSPRITE(ALL):: GOSUB 2280 :: OPE
     N #1:D$,INTERNAL,INPUT ,FIXED
2250 FOR I=2 TO 20 :: INPUT #1:P$ :: FOR J=1
      TO L :: Z1(I,J)=ASC(SEG$(P$,J,1)):: NE
     XT J :: NEXT I
2260 INPUT #1:P$ :: E=ASC(P$):: CALL SCREEN(
     E):: INPUT #1:P$ :: FOR I=1 TO 14 :: FR
     (I)=ASC(SEG$(P$,2*I-1,1)):: B(I)=ASC(SE
     G$(P$,2*I,1))
```

```
2270 CALL COLOR(I,FR(I),B(I)):: NEXT I :: CL
     OSE #1 :: IF ASC(D$)=67 THEN 2200 ELSE
     2210
2280 DISPLAY AT(1,1):Q$ :: FOR I=1 TO 400 ::
      NEXT I :: ACCEPT AT(1,1)BEEP:D$ :: IF
     D$="" THEN 1860 ELSE GOSUB 1260
2290 RETURN
```

Program 2. SuperFont LOAD Demo

```
100 !GAME
110 !GET REDEFINED CHARS
120 DIM Z1(20,32),FR(14),B(14):: CALL CLEAR
130 B$="DSK1.FONT" :: C$="DSK1.SCREEN" :: RE
    M B$,C$="CS1"::REM EQUIVALENT FOR CASSET
    TE
140 OPEN #1:B$,INTERNAL,INPUT ,FIXED
150 INPUT #1:F,NEWA$ :: IF F<>112 THEN CALL
    CHAR(F+32,NEWA$):: GOTO 150
160 CLOSE #1
170 !GET SCREEN & COLORS
180 OPEN #1:C$,INTERNAL,INPUT ,FIXED
190 FOR I=2 TO 20 :: INPUT #1:P$ :: FOR J=1
    TO 32 :: Z1(I,J)=ASC(SEG$(P$,J,1)):: NEX
    T J :: NEXT I
200 INPUT #1:P$ :: E=ASC(P$):: CALL SCREEN(E
    ):: INPUT #1:P$ :: FOR I=1 TO 14 :: FR(I
    )=ASC(SEG$(P$,2*I-1,1)):: B(I)=ASC(SEG$(
    P$,2*I,1))
210 CALL COLOR(I,FR(I),B(I)):: NEXT I :: CLO
    SE #1
220 CALL CLEAR :: FOR I=2 TO 20 :: FOR J=1 T
    O 32 :: CALL HCHAR(I,J,Z1(I,J)):: NEXT J
     :: NEXT I
230 FOR T=1 TO 1000 :: NEXT T
```

Sound Maker

Frank Elsesser

The TI-99/4A home computer can produce a great variety of sounds. "Sound Maker" will appeal to anyone who wants to add sound effects or music to a program. It's also an easy, but highly effective way to explore the audio capabilities of your computer.

The TI-99/4A, like most other computers, requires that you use numbers to program a sound's duration, pitch, and volume. Finding the right numbers to produce exactly the sound you want can be a fairly inefficient trial-and-error process. You must type a CALL SOUND statement with each attempt, trying out different values for the parameters, until you find the combination of numbers that matches the sound you're looking for. Wouldn't it be nice if this process were automated so you could spend more time being creative and less time typing and manipulating numbers?

"Sound Maker" does this and more. It allows you to experiment easily with different settings of amplitude (volume), frequency (pitch), and time (duration). You can work with simple and complex tones, noise, and modulation to create a variety of special effects. The computer will even print the program statements used to create the sounds so you can add them to your own programs.

When you run the program, it will take awhile for the computer to establish room for variables and arrays and do other housekeeping chores before you see the introduction. After a brief demonstration of tones and explosions, the main menu will be displayed.

You have a choice of three basic tones: simple, noise, and complex. Selecting simple tones allows you to experiment with the amplitude, frequency, and time of a simple tone. Choosing noise tones brings you a menu of four tonal types: periodic, periodic with tone, white, and white with tone. Complex tones consist of three simple tones and one noise tone played simultaneously. The frequency and amplitude of each tone can be changed individually.

After you have selected the basic tone and found the combination of parameters which suits your taste, you will be taken to the modulation menu. Here, you can make the amplitude, frequency, or time change—while the note is playing—to create special effects. The procedure for modulating frequency and time is fairly straightforward. However, choosing amplitude modulation displays another menu. Three types of amplitude modulation are available: on/off clicking, positive ramp, and negative steps. On/off clicking turns the sound on and off like the busy signal on a telephone. Positive ramp makes the tone louder with time. Negative steps make it quieter with time. Positive ramps and negative steps can be used in your programs to give the effect of an approaching and receding alien ship.

Experimenting with sounds using Sound Maker is so easy that you will have the freedom to create sounds you never thought possible on your TI.

Sound Maker

```
100 CALL CLEAR
110 DIM S1(60)
120 R$="SOUND MAKER"
130 CALL SCREEN(14)
140 FOR P=1 TO 11
150 CALL SOUND(150,-4,1)
160 CALL HCHAR(12,9+P,ASC(SEG$(R$,P,1)))
170 NEXT P
180 FOR DE=1 TO 500
190 NEXT DE
200 CALL CLEAR
210 FOR I=1 TO 8
220 CALL COLOR(I,16,1)
230 NEXT I
240 PRINT " YOUR TI COMPUTER IS CAPABLEOF MA
    KING AN ALMOST ENDLESS VARIETY OF SPECIA
    L EFFECT{3 SPACES}SOUNDS.": :
250 PRINT " THE PURPOSE OF THIS PROGRAMIS TO
     HELP YOU FIND JUST THERIGHT SOUND FOR Y
    OUR SPECIALEFFECT.": :
260 PRINT " IT ALLOWS YOU TO GENERATE  SIMPL
    E TO COMPLEX SOUNDS ANDTO THEN ADD SPECI
    AL EFFECT  MODULATIONS.": : :
270 PRINT "{4 SPACES}(ONE MOMENT PLEASE)": :
280 REM COMPUTING 5-OCTAVES
290 FOR N=0 TO 60
300 S1(N)=INT(110*(2^(1/12))^N+.5)
```

```
310 CALL SOUND(-500,S1(N),4)
320 NEXT N
330 FOR A=0 TO 20 STEP 5
340 CALL SOUND(700,-7,A)
350 NEXT A
360 REM START VALUES &MAIN MENU
370 CALL CLEAR
380 T1=1000
390 F2=30000
400 A3=30
410 F4=30000
420 A5=30
430 F6=30000
440 A7=30
450 L8=1
460 A9=30
470 PRINT TAB(12);"MENU": : : :
480 PRINT TAB(5);"1.SIMPLE TONES": : :
490 PRINT TAB(5);"2.NOISE TONES": : :
500 PRINT TAB(5);"3.COMPLEX TONES": : :
510 PRINT TAB(5);"4.EXIT": : :
520 INPUT "SELECT NO.(1,2,3,OR 4)":M
530 ON M GOTO 650,1650,2830,5110
540 REM MODULATION MENU
550 CALL CLEAR
560 PRINT TAB(10);"MODULATIONS": : : :
570 PRINT
580 PRINT TAB(9);"1.AMPLITUDE": :
590 PRINT TAB(9);"2.FREQUENCY": :
600 PRINT TAB(9);"3.TIME": :
610 PRINT TAB(9);"4.MAIN MENU": : : : : :
620 INPUT "ENTER NUMBER(1,2,3,4)":NS
630 ON NS GOTO 930,4570,4840,370
640 REM SINGLE TONE GEN
650 CALL CLEAR
660 PRINT TAB(9);"SIMPLE TONES": : :
670 PRINT
680 PRINT "(PRESS ENTER TO SELECT TONE)": :
    : : : : : :
690 A3=2
700 FOR N=0 TO 60
710 F2=S1(N)
720 CALL SOUND(500,F2,A3)
730 CALL KEY(0,K,Z)
740 IF K=13 THEN 770
750 NEXT N
760 GOTO 370
770 PRINT "FREQUENCY=";F2: :
780 PRINT "TIME=1000,AMPLITUDE=2": : : : :
790 PRINT "CHANGE PARAMETERS(Y OR N)?": : :
```

```
800 CALL KEY(0,K,Z)
810 IF Z+1=1 THEN 800
820 IF K=89 THEN 840
830 IF K=78 THEN 550 ELSE 800
840 INPUT "NEW TIME=":T1
850 INPUT "NEW AMPL=":A2
860 CALL CLEAR
870 CALL SOUND(T1,F2,A3)
880 PRINT TAB(6);"TRY AGAIN(Y OR N)": : : :
    : : : : : : :
890 CALL KEY(0,K,Z)
900 IF Z+1=1 THEN 890
910 IF K=89 THEN 650
920 IF K=78 THEN 550 ELSE 890
930 REM AMPL MODULATION MENU
940 CALL CLEAR
950 PRINT "{3 SPACES}AMPLITUDE MODULATION":
    : : :
960 PRINT TAB(5);"1.ON/OFF CLICKING": :
970 PRINT TAB(5);"2.POS RAMP": :
980 PRINT TAB(5);"3.NEG STEPS": :
990 PRINT TAB(5);"4.MODULATION MENU": : : :
1000 INPUT "SELECT(1,2,3,OR 4)":NM
1010 ON NM GO TO 1030,1230,1450,550
1020 REM ON/OFF AM GEN
1030 CALL CLEAR
1040 PRINT TAB(8);"ON/OFF CLICKING": : :
1050 PRINT
1060 PRINT "FOR Z=1 TO 10"
1070 PRINT "CALL SOUND(50,F2,A3,...)"
1080 PRINT "NEXT Z": : : :
1090 ZZ=10
1100 T1=50
1110 GOSUB 4640
1120 PRINT "CHANGE PARAMETERS(Y OR N)?": :
1130 FOR Z=1 TO ZZ
1140 CALL SOUND(T1,F2,A3,F4,A5,F6,A7,-L8,A9)
1150 NEXT Z
1160 CALL KEY(0,I,J)
1170 IF I=89 THEN 1190
1180 IF I=78 THEN 940 ELSE 1130
1190 INPUT "NEW Z MAX=":ZZ
1200 INPUT "TIME=":T1
1210 GOTO 1120
1220 REM +RAMP AM GEN
1230 CALL CLEAR
1240 PRINT TAB(8);"POSITIVE RAMP": : :
1250 PRINT "FOR A=30 TO 0 STEP -2"
1260 PRINT "CALL SOUND(-200,F2,A,F4,A,...)"
1270 PRINT "NEXT A": : : :
```

```
1280 T1=-200
1290 SS=2
1300 GOSUB 4640
1310 PRINT "CHANGE PARAMETERS(Y OR N)?": : :
1320 FOR A=30 TO 0 STEP -SS
1330 CALL SOUND(T1,F2,A,F4,A,F6,A,-L8,A)
1340 NEXT A
1350 FOR D=0 TO 500
1360 NEXT D
1370 CALL KEY(0,K,Z)
1380 IF Z+1=1 THEN 1320
1390 IF K=89 THEN 1410
1400 IF K=78 THEN 940 ELSE 470
1410 INPUT "AMPL STEP SIZE=-":SS
1420 INPUT "TIME=":T1
1430 GOTO 1310
1440 REM NEG STEPS AM GEN
1450 CALL CLEAR
1460 PRINT TAB(6);"NEGATIVE STEPS": :
1470 PRINT "FOR A=0 TO 30 STEP 5"
1480 PRINT "CALL SOUND(500,F2,A,F4,..)"
1490 PRINT "NEXT A": : :
1500 T1=500
1510 SS=5
1520 GOSUB 4640
1530 PRINT "CHANGE PARAMETERS(Y OR N)?": : :
1540 FOR A=0 TO 30 STEP SS
1550 CALL SOUND(T1,F2,A,F4,A,F6,A,-L8,A)
1560 NEXT A
1570 CALL KEY(0,K,Z)
1580 IF Z+1=1 THEN 1540
1590 IF K=89 THEN 1610
1600 IF K=78 THEN 940 ELSE 1530
1610 INPUT "AMPL STEP SIZE=":SS
1620 INPUT "TIME=":T1
1630 GOTO 1530
1640 REM NOISE MENU
1650 CALL CLEAR
1660 PRINT TAB(8);"NOISE TONES": : : :
1670 PRINT TAB(6);"1.PERIODIC NOISE": :
1680 PRINT TAB(6);"2.PERIODIC WITH TONE": :
1690 PRINT TAB(6);"3.WHITE NOISE": :
1700 PRINT TAB(6);"4.WHITE WITH TONE": :
1710 PRINT TAB(6);"5.MAIN MENU": : : :
1720 INPUT "NOISE TYPE(1.2,3,4,5)":NT
1730 ON NT GOTO 1750,1970,2290,2510,370
1740 REM PERIODIC N GEN
1750 CALL CLEAR
1760 PRINT TAB(8);"PERIODIC NOISE": :
1770 T1=1000
```

```
1780 A9=2
1790 FOR L8=1 TO 4
1800 CALL SOUND(T1,-L8,A9)
1810 PRINT TAB(12);"TYPE=";L8: :
1820 NEXT L8
1830 PRINT "SELECT TYPE&TIME(Y OR N)?": : :
1840 CALL KEY(0,K,Z)
1850 IF Z+1=1 THEN 1840
1860 IF K=78 THEN 1640
1870 IF K=89 THEN 1880 ELSE 1840
1880 INPUT "TYPE=":L8
1890 INPUT "TIME=":T1
1900 CALL SOUND(T1,-L8,A9)
1910 PRINT TAB(8);"TRY AGAIN(Y OR N)?": :
1920 CALL KEY(0,K,Z)
1930 IF Z+1=1 THEN 1920
1940 IF K=89 THEN 1880
1950 IF K=78 THEN 550 ELSE 1920
1960 REM TYPE 4 N WITH TONE
1970 CALL CLEAR
1980 PRINT "  PERIODIC NOISE WITH TONE": : :
1990 PRINT "(PRESS ENTER TO SELECT TONE)": :
     : : : : : :
2000 T1=2000
2010 A=30
2020 A9=2
2030 Z=0
2040 FOR N=0 TO 60
2050 F6=S1(N)
2060 CALL KEY(0,K,Z)
2070 IF K=13 THEN 2120
2080 CALL SOUND(T1,F6,A,F6,A,F6,A,-4,A9)
2090 L8=4
2100 NEXT N
2110 GOTO 1650
2120 CALL CLEAR
2130 PRINT "  TYPE -4 PARAMETERS": : :
2140 PRINT "CALL SOUND(T1,F,30,F...-4,2)": :
2150 PRINT "TIME=2000": :
2160 PRINT "FREQUENCY=";F6: : :
2170 PRINT "  (DEPRESS ""R""TO REPEAT)": : :
2180 PRINT "TRY NEW PARAMETERS(Y OR N)?": :
2190 CALL SOUND(T1,F6,30,F6,30,F6,30,-4,A9)
2200 CALL KEY(0,K,Z)
2210 IF Z+1=1 THEN 2200
2220 IF K=89 THEN 2250
2230 IF K=82 THEN 2190
2240 IF K=78 THEN 550 ELSE 2200
2250 INPUT "TIME=":T1
2260 INPUT "AMPL=":A9
```

```
2270 GOTO 2180
2280 REM WHITE N GEN
2290 CALL CLEAR
2300 PRINT TAB(9);"WHITE NOISE": : : :
2310 T1=2000
2320 A9=2
2330 FOR L8=5 TO 8
2340 CALL SOUND(T1,-L8,A9)
2350 PRINT TAB(9);"TYPE=";L8: : :
2360 NEXT L8
2370 PRINT "SELECT TYPE&TIME(Y OR N)?": : :
2380 CALL KEY(0,K,Z)
2390 IF Z+1=1 THEN 2380
2400 IF K=78 THEN 1650
2410 IF K=89 THEN 2420 ELSE 2380
2420 INPUT "TYPE=":L8
2430 INPUT "TIME=":T1
2440 CALL SOUND(T1,-L8,A9)
2450 PRINT TAB(8);"TRY AGAIN(Y OR N)?": :
2460 CALL KEY(0,K,Z)
2470 IF Z+1=1 THEN 2460
2480 IF K=89 THEN 2420
2490 IF K=78 THEN 550 ELSE 2460
2500 REM WHITE N WITH TONES
2510 CALL CLEAR
2520 PRINT "{3 SPACES}WHITE NOISE WITH TONES
     ": : :
2530 PRINT "(PRESS ENTER TO SELECT TONE)": :
      : : : : : :
2540 PRINT "NOTE:GOOD EFFECTS AT HIGH
     {3 SPACES}FREQUENCIES": :
2550 T1=1000
2560 A9=2
2570 L8=8
2580 Z=0
2590 FOR N=0 TO 60
2600 F6=S1(N)
2610 CALL SOUND(T1,F6,30,F6,30,F6,30,-L8,A9)
2620 CALL KEY(0,K,Z)
2630 IF K=13 THEN 2660
2640 NEXT N
2650 GOTO 1650
2660 CALL CLEAR
2670 PRINT TAB(7);"TYPE -8 PARAMETERS": : :
2680 PRINT "CALL SOUND(T1,F,30,F..-8,A9)": :
2690 PRINT "{3 SPACES}TIME=1000": :
2700 PRINT "{3 SPACES}FREQUENCY=";F6: : :
2710 PRINT "{3 SPACES}NOISE AMP=2": : : :
2720 PRINT "{4 SPACES}(PRESS""R""TO REPEAT)"
     : : : :
```

```
2730 PRINT "  NEW PARAMETERS(Y OR N)?": :
2740 CALL SOUND(T1,F6,30,F6,30,F6,30,-L8,A9)
2750 CALL KEY(0,K,Z)
2760 IF Z+1=1 THEN 2750
2770 IF K=89 THEN 2800
2780 IF K=82 THEN 2740
2790 IF K=78 THEN 550 ELSE 2750
2800 INPUT "TIME=":T1
2810 INPUT "AMPL=":A9
2820 GOTO 2730
2830 REM COMPLEX TONE
2840 CALL CLEAR
2850 PRINT "CALL SOUND(T1,F2,A3,F4,A5,F6"
2860 PRINT "{11 SPACES}A7,-L8,A8)": :
2870 PRINT "----------------------------":
2880 PRINT "-USE KEYS 1-9 TO INCREASE
     {4 SPACES}VALUES"
2890 PRINT "-DEPRESS SHIFT&1-9 KEYS TO
     {3 SPACES}DECREASE VALUES"
2900 PRINT "-DEPRESS""ENTER""FOR REPEAT"
2910 PRINT "-DEPRESS ""E"" TO EXIT"
2920 PRINT "____________________________": :
2930 PRINT "{12 SPACES}T1{3 SPACES}F2  A3": :
2940 PRINT "CALL SOUND({4 SPACES},
     {4 SPACES},  "
2950 T1=1000
2960 Z=0
2970 PRINT "{9 SPACES},  ,{4 SPACES},  ,- ,
      )": :
2980 PRINT "{6 SPACES}F4  A5  F6  A7 L8 A9"
2990 REM START VALUES
3000 F2=110
3010 A3=5
3020 F4=110
3030 A5=5
3040 F6=110
3050 A7=5
3060 L8=1
3070 A9=5
3080 REM STRING PRINT(T1,F2,..)
3090 D1$=STR$(T1)
3100 CALL HCHAR(20,17,32)
3110 FOR L=1 TO LEN(D1$)
3120 CALL HCHAR(20,L+13,ASC(SEG$(D1$,L,1)))
3130 NEXT L
3140 IF Z+1=1 THEN 3150 ELSE 3600
3150 D3$=STR$(F2)
3160 CALL HCHAR(20,22,32)
3170 FOR L=1 TO LEN(D3$)
3180 CALL HCHAR(20,L+18,ASC(SEG$(D3$,L,1)))
```

```
3190 NEXT L
3200 IF Z+1=1 THEN 3210 ELSE 3600
3210 D4$=STR$(A3)
3220 CALL HCHAR(20,25,32)
3230 FOR L=1 TO LEN(D4$)
3240 CALL HCHAR(20,L+23,ASC(SEG$(D4$,L,1)))
3250 NEXT L
3260 IF Z+1=1 THEN 3270 ELSE 3600
3270 D5$=STR$(F4)
3280 FOR L=1 TO LEN(D5$)
3290 CALL HCHAR(21,11,32)
3300 CALL HCHAR(21,L+7,ASC(SEG$(D5$,L,1)))
3310 NEXT L
3320 IF Z+1=1 THEN 3330 ELSE 3600
3330 D6$=STR$(A5)
3340 FOR L=1 TO LEN(D6$)
3350 CALL HCHAR(21,14,32)
3360 CALL HCHAR(21,L+12,ASC(SEG$(D6$,L,1)))
3370 NEXT L
3380 IF Z+1=1 THEN 3390 ELSE 3600
3390 D7$=STR$(F6)
3400 FOR L=1 TO LEN(D7$)
3410 CALL HCHAR(21,19,32)
3420 CALL HCHAR(21,L+15,ASC(SEG$(D7$,L,1)))
3430 NEXT L
3440 IF Z+1=1 THEN 3450 ELSE 3600
3450 D8$=STR$(A7)
3460 FOR L=1 TO LEN(D8$)
3470 CALL HCHAR(21,22,32)
3480 CALL HCHAR(21,L+20,ASC(SEG$(D8$,L,1)))
3490 NEXT L
3500 IF Z+1=1 THEN 3510 ELSE 3600
3510 D9$=STR$(L8)
3520 CALL HCHAR(21,25,ASC(D9$))
3530 IF Z+1=1 THEN 3540 ELSE 3600
3540 D0$=STR$(A9)
3550 FOR L=1 TO LEN(D0$)
3560 CALL HCHAR(21,28,32)
3570 CALL HCHAR(21,L+26,ASC(SEG$(D0$,L,1)))
3580 NEXT L
3590 REM SEPARATE UP-DN&EXIT
3600 CALL KEY(0,K,Z)
3610 IF K=69 THEN 550
3620 CALL KEY(0,I,J)
3630 IF I<>13 THEN 3660
3640 GOSUB 4140
3650 GOTO 3600
3660 IF Z+1=1 THEN 3600
3670 IF K>32 THEN 3680 ELSE 3600
3680 IF K=42 THEN 4530
```

```
3690 IF K<42 THEN 4220
3700 IF K=64 THEN 4170
3710 IF K=94 THEN 4400
3720 IF K>48 THEN 3740 ELSE 3600
3730 REM UP COMMANDS
3740 ON (K-48)GOTO 3750,3790,3840,3880,3930,
     3970,4020,4060,4100
3750 IF T1>3900 THEN 3090
3760 T1=T1+100
3770 GOSUB 4140
3780 GOTO 3090
3790 IF N2>59 THEN 3150
3800 N2=N2+1
3810 F2=S1(N2)
3820 GOSUB 4140
3830 GOTO 3150
3840 IF A3>29 THEN 3210
3850 A3=A3+1
3860 GOSUB 4140
3870 GOTO 3210
3880 IF N4>59 THEN 3270
3890 N4=N4+1
3900 F4=S1(N4)
3910 GOSUB 4140
3920 GOTO 3270
3930 IF A5>29 THEN 3330
3940 A5=A5+1
3950 GOSUB 4140
3960 GOTO 3330
3970 IF N6>59 THEN 3390
3980 N6=N6+1
3990 F6=S1(N6)
4000 GOSUB 4140
4010 GOTO 3390
4020 IF A7>29 THEN 3450
4030 A7=A7+1
4040 GOSUB 4140
4050 GOTO 3450
4060 IF L8>7 THEN 3510
4070 L8=L8+1
4080 GOSUB 4140
4090 GOTO 3510
4100 IF A9>29 THEN 3540
4110 A9=A9+1
4120 GOSUB 4140
4130 GOTO 3540
4140 CALL SOUND(-T1,F2,A3,F4,A5,F6,A7,-L8,A9
     )
4150 RETURN
4160 REM DOWN COMMANDS
```

```
4170 IF N2<1 THEN 3150
4180 N2=N2-1
4190 F2=S1(N2)
4200 GOSUB 4140
4210 GOTO 3150
4220 ON (K-32)GOTO 4230,4230,4270,4310,4360,
     4450,4400,4490
4230 IF T1<200 THEN 3090
4240 T1=T1-100
4250 GOSUB 4140
4260 GOTO 3090
4270 IF A3<1 THEN 3210
4280 A3=A3-1
4290 GOSUB 4140
4300 GOTO 3210
4310 IF N4<1 THEN 3270
4320 N4=N4-1
4330 F4=S1(N4)
4340 GOSUB 4140
4350 GOTO 3270
4360 IF A5<1 THEN 3330
4370 A5=A5-1
4380 GOSUB 4140
4390 GOTO 3330
4400 IF N6<1 THEN 3390
4410 N6=N6-1
4420 F6=S1(N6)
4430 GOSUB 4140
4440 GOTO 3390
4450 IF A7<1 THEN 3450
4460 A7=A7-1
4470 GOSUB 4140
4480 GOTO 3450
4490 IF A9<1 THEN 3540
4500 A9=A9-1
4510 GOSUB 4140
4520 GOTO 3540
4530 IF L8<2 THEN 3510
4540 L8=L8-1
4550 GOSUB 4140
4560 GOTO 3510
4570 REM FREQ MOD
4580 CALL CLEAR
4590 PRINT "{3 SPACES}FREQUENCY MODULATION":
     : :
4600 PRINT "FOR D=0 TO 100 STEP 2"
4610 PRINT "CALL SOUND(-50,F2+D,A3,F4+D, ...
     -L8,A9)"
4620 PRINT "NEXT D": : :
4630 GOTO 4680
```

```
4640 PRINT "T1=";T1;"F2=";F2;"A3=";A3
4650 PRINT "F4=";F4;"A5=";A5;"F6=";F6
4660 PRINT "A7=";A7;"L8=";L8;"A9=";A9: : :
4670 RETURN
4680 T1=-50
4690 DD=100
4700 FS=2
4710 GOSUB 4640
4720 PRINT "CHANGE PARAMETERS(Y OR N)?": :
4730 FOR D=O TO DD STEP FS
4740 CALL SOUND(T1,F2+D,A3,F4+D,A5,F6+D,A7,-
     L8,A9)
4750 NEXT D
4760 CALL KEY(0,K,Z)
4770 IF Z+1=1 THEN 4730
4780 IF K=89 THEN 4800
4790 IF K=78 THEN 550 ELSE 4730
4800 INPUT "FREQ RANGE=":DD
4810 INPUT "FREQ STEPS=":FS
4820 INPUT "TIME=":T1
4830 GOTO 4720
4840 REM TIME MOD
4850 CALL CLEAR
4860 PRINT TAB(5);"TIME MODULATION": : :
4870 PRINT "FOR T1=1 TO 300 STEP 10"
4880 PRINT "CALL SOUND(T1,F2,A3,...A9)"
4890 PRINT "FOR D=0 TO 5"
4900 PRINT "NEXT D"
4910 PRINT "NEXT T1": : :
4920 PRINT "LAST VALUES"
4930 GOSUB 4640
4940 TM=300
4950 D=5
4960 TS=10
4970 PRINT "CHANGE PARAMETERS(Y OR N)?": :
4980 FOR T1=TS TO TM STEP TS
4990 FOR T=0 TO D
5000 NEXT T
5010 CALL SOUND(T1,F2,A3,F4,A5,F6,A7,-L8,A9)
5020 NEXT T1
5030 CALL KEY(0,K,Z)
5040 IF Z+1=1 THEN 4980
5050 IF K=89 THEN 5070
5060 IF K=78 THEN 540 ELSE 4980
5070 INPUT "TOT TIME=":TM
5080 INPUT "TIME STEP=":TS
5090 INPUT "DELAY=":D
5100 GOTO 4970
5110 END
```

Sound Shaper

Steven Kaye
TI Translation by Patrick Parrish

"Sound Shaper" manipulates volume and frequency to give the TI with Extended BASIC a smoother, more musical sound. The program also runs on the TI with regular BASIC.

The TI produces waveforms which are square. One microsecond the sound is off, the next it's on. This abrupt onset of sound produces somewhat nonmusical sounds. The tones sound electronic and unlike any acoustic instrument.

As an alternative to turning the sound on and off abruptly, we can increase and decrease the amplitude (volume) more gradually under control of the program.

"Sound Shaper" has two sound producing routines that can be used in your programs. Echo effect produces a sound that its name implies. The actual routine producing the sound is in lines 550 to 670. The routine can be extracted as is and used.

The Shaped Musical Notes routine is a bit more flexible. The program will ask for a rise and fall value. Experiment with different values. Try low values like .5,2 and .1,1 and higher values like 10,10. For an eerie sound try 5,20. If the input values are much higher the program seems to continue endlessly, but will eventually return to the main menu.

Experiment with values and write down the ones you like. Once you have found the effect you want for a particular application, copy the routine from lines 400 to 490. Be sure to supply values for R and D.

Sound Shaper

```
100 CALL CLEAR
110 CALL SCREEN(15)
120 PRINT TAB(7):"SHAPING TI SOUNDS"
130 FOR T=1 TO 6
140 PRINT
150 NEXT T
160 PRINT "CHOOSE:"
170 PRINT
180 PRINT
190 PRINT TAB(4);"1) SHAPED MUSICAL NOTES"
```

```
200 PRINT
210 PRINT TAB(4);"2) ECHO"
220 PRINT
230 PRINT TAB(4);"3) QUIT"
240 PRINT
250 INPUT A$
260 IF (VAL(A$)<1)+(VAL(A$)>3)THEN 250
270 ON VAL(A$)GOTO 290,520,690
280 REM THIS PART PRODUCES "SHAPED" MUSICAL
    NOTES
290 CALL CLEAR
300 CALL SCREEN(13)
310 PRINT TAB(3);"* SHAPED MUSICAL NOTES *"
320 FOR T=1 TO 10
330 PRINT
340 NEXT T
350 PRINT "ENTER RISE AND FALL TIMES -"
360 PRINT "USE VALUES GREATER THAN ZERO";
370 PRINT
380 INPUT R,D
390 IF (R=0)+(D=0)THEN 380
400 FOR F=110 TO 880 STEP 30
410 FOR DB=30 TO 0 STEP -5/R
420 CALL SOUND(-10,F,DB)
430 NEXT DB
440 FOR DB=0 TO 30 STEP 5/D
450 CALL SOUND(-10,F,DB)
460 NEXT DB
470 FOR T=1 TO 50
480 NEXT T
490 NEXT F
500 GOTO 100
510 REM THIS PART CREATES AN ECHO EFFECT
520 CALL CLEAR
530 CALL SCREEN(14)
540 PRINT TAB(8);"* ECHO EFFECT *"
550 FOR T=1 TO 12
560 PRINT
570 NEXT T
580 FOR F=110 TO 880 STEP 30
590 FOR DB=1 TO 30
600 CALL SOUND(-10,F,DB)
610 FOR T=1 TO 10
620 NEXT T
630 CALL SOUND(-10,990-F,DB)
640 FOR J=1 TO 10
650 NEXT J
660 NEXT DB
670 NEXT F
680 GOTO 100
690 END
```

The Mozart Machine

Donald J. Eddington
TI Translation by Gregg Peele

Your computer can compose music with this special technique. The compositions are remarkably Mozartian in style.

If you've ever gone through the steps to make your computer play a particular piece of music, you realize that it can be a significant programming task. To have your computer actually *write* music is a real feat.

To accomplish this, we've first got to find a way to work with CALL SOUND values in DATA statements in order to make the measures of music. Also, we need to be able to READ the values in any order so that the songs will be different with each run of the program. The commonly used string manipulation methods won't work very well here. We need variety, and the traditional way of working with strings quickly results in a tangled mess.

Array Referencing

The shortest, best way to solve this problem is to use a technique called array referencing. First, to get the measures of music, you set up an array of all variables, then reference them by subscript in a loop. Specifically, 14 variations on nine variables are required to make the music for this program. The random number generator is used to make the music different every time the program is run.

A Mozartian flavor results from a deliberate shortening of the low notes and making the high notes of varying lengths. And to keep the music from becoming totally random, DATA statements select the measures by their underlying tonality—tonic, subdominant, dominant, or supertonic. In keeping with classical style, a cadence is provided every four measures with a final ending chord for each tune.

TI Mozart

```
100 DIM X(14,9)
110 REM THE TICLANG AMAZIUS MOZART
120 CALL CLEAR
130 CALL SCREEN(14)
140 PRINT "{3 SPACES}WELCOME! I AM TICLANG"
150 PRINT "{6 SPACES}AMAZIUS MOZART."
160 PRINT
170 PRINT "I PLAY SONGS LIKE THE CHILD"
180 PRINT "PRODIGY, WOLFGANG  AMADEUS "
190 PRINT "  MOZART MIGHT HAVE DONE."
200 PRINT
210 PRINT "{5 SPACES}MOZART LIVED FROM"
220 PRINT "{8 SPACES}1756 TO 1791"
230 PRINT "AND WROTE OVER 626 WORKS IN"
240 PRINT "{8 SPACES}31 YEARS."
250 PRINT
260 PRINT "THE 5 PIECES YOU HEAR ARE"
270 PRINT "{3 SPACES}BEING WRITTEN BY THE "
280 PRINT "  COMPUTER AS YOU LISTEN!"
290 PRINT
300 PRINT
310 PRINT
320 FOR T=1 TO 14
330 FOR TT=1 TO 9
340 READ X(T,TT)
350 NEXT TT
360 NEXT T
370 DATA 196,494,494,247,494,587,294,494,220
380 DATA 196,494,587,247,587,523,294,494,294
390 DATA 196,494,523,247,587,523,294,494,294
400 DATA 196,523,587,262,659,784,330,784,262
410 DATA 196,523,659,262,659,587,330,523,262
420 DATA 196,659,523,262,392,659,330,523,196
430 DATA 220,523,587,262,784,587,294,523,220
440 DATA 220,440,587,220,523,494,294,440,262
450 DATA 220,659,784,262,587,523,294,494,220
460 DATA 220,523,494,262,440,494,330,523,220
470 DATA 220,523,494,262,440,494,330,523,262
480 DATA 196,494,523,247,587,587,294,587,294
490 DATA 196,587,523,220,440,440,294,440,220
500 DATA 196,659,587,262,523,523,330,523,262
510 P=250
520 DATA 1,3,6,2,1,4,6,2,3,4,1,5,1,4,6,7,1,4
    ,6,2,1,3,6,9
530 DATA 1,1,4,5,1,4,6,2,3,4,1,5,1,4,1,5,1,4
    ,6,9
540 DATA 1,4,6,2,3,6,1,5,1,4,6,7,3,4,6,2,1,4
    ,3,7,1,4,6,9
```

```
550 DATA 1,4,3,7,1,6,4,5,6,3,6,2,4,6,1,5,1,4
    ,6,9
560 DATA 1,4,3,7,6,3,6,2,4,6,1,5,1,3,6,7,3,6
    ,1,5,1,4,6,9,8
570 READ RR
580 ON RR GOTO 650,590,730,780,610,860,640,9
    40,1040
590 Y=12
600 GOTO 990
610 Y=14
620 GOTO 990
630 Y=13
640 GOTO 990
650 Y=1
660 RANDOMIZE
670 IF RND>.35 THEN 700
680 Y=3
690 RANDOMIZE
700 IF RND<.75 THEN 720
710 Y=2
720 GOTO 990
730 Y=10
740 RANDOMIZE
750 IF RND>.4 THEN 780
760 Y=11
770 GOTO 990
780 Y=4
790 RANDOMIZE
800 IF RND>.35 THEN 820
810 Y=5
820 RANDOMIZE
830 IF RND<.75 THEN 850
840 Y=6
850 GOTO 990
860 Y=7
870 RANDOMIZE
880 IF RND>.35 THEN 900
890 Y=8
900 RANDOMIZE
910 IF RND<.75 THEN 930
920 Y=9
930 GOTO 990
940 PRINT "{5 SPACES}WELL, THAT'S ALL"
950 PRINT "{4 SPACES}HOPE YOU LIKED IT!!"
960 PRINT "RUN IT AGAIN AND HEAR FIVE "
970 PRINT "{8 SPACES}MORE SONGS."
980 END
990 FOR I=1 TO 9 STEP 3
1000 CALL SOUND(P,X(Y,I),2,X(Y,I+1),2)
```

```
1010 CALL SOUND(P,X(Y,I),30,X(Y,I+2),2)
1020 NEXT I
1030 GOTO 570
1040 CALL SOUND(1800,196,2,494,2,784,2)
1050 FOR T=1 TO 800
1060 NEXT T
1070 KOL=INT(RND*8)+8
1080 CALL SCREEN(KOL)
1090 GOTO 570
```

6
Sprites

6
A Beginner's Guide to Sprites

Gary K. Hamlin

Sprites are easy to create and use. They enhance the graphic displays and make smooth moving objects simple to control. This program requires Extended BASIC.

An exciting feature of most personal computers is their color graphics capability. Even if the computer was purchased for financial management or complex mathematical computations, it's hard to resist experimenting with graphics. Defining and manipulating your own characters—from oddly shaped "doodles" to those resembling actual objects—can be a lot of fun, and can have practical applications too.

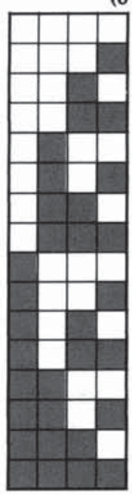
Graphics are quite easy to use on the TI-99/4A, with TI BASIC's series of built-in graphics subprograms. Once they are learned, subprograms used with sprites are also easy. Sprites require the addition of the TI Extended BASIC cartridge, and will greatly enhance the computer's possible graphics applications.

Sprites Vs. Characters

A sprite can be one of the characters from the TI character set or can be made from user-created dot patterns, just as is done in standard BASIC, using the CHAR subprogram. Sprites, however, are more versatile than standard BASIC characters. Sprites can be positioned at 49,152 different screen locations (192 rows by 256 columns); standard characters have only 768 possible screen positions (24 rows by 32 columns). This permits faster and smoother character movement, a significant advantage in game programming.

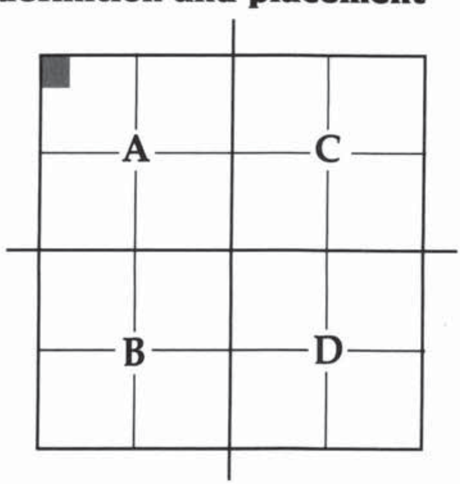
The CALL CHAR statement is used in defining Extended BASIC sprites much as it is in standard BASIC character definition. The same 8 × 8 dot grid and hexadecimal on/off codes are used (Figure 1), but sprites can occupy up to four 8 × 8 dot blocks. The resulting hexadecimal code pattern identifiers can contain up to 64 characters. The computer will automatically reserve four blocks for each sprite, whether or not all of them are actually used; therefore, it's advisable to think in sets of four blocks even if the sprite is to occupy only a small portion of the reserved area. Figure 2 illustrates the arrangement of the blocks.

Figure 1. Pattern Identifier Guide



Binary Code (0 = OFF 1 = ON)	Hexadecimal Code
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Figure 2. Order of 8 × 8 dot blocks for sprite definition and placement



Defining a Sprite

Sprite characters should be assigned character codes divisible by four if they are to occupy more than one of the four blocks. This is less critical for single-block sprites, but the character code assigned to a single-block sprite will affect which blocks—A, B, C, or D—the sprite will occupy. The computer will always assign a character code divisible by four to block A.

The order of the blocks, as shown in Figure 2, is also critical when writing out the pattern identifier of a multiple-block sprite. If the order is not observed—and if block A's character code is not evenly divisible by four—the four segments of the sprite will become jumbled when displayed on the screen. Always begin with the pattern identifier for block A, at the upper left, proceeding to the lower left (B), upper right (C), and concluding with block D at the lower right.

It should also be kept in mind that in program references to the screen location of a sprite, the specified location identifies the dot occupying the upper-left corner of the four reserved blocks (shown as the shaded dot in block A, Figure 2). This is true whether or not that dot constitutes a visible part of the sprite.

The sprite mapped out in Figure 3 is intended to occupy four 8 × 8 blocks. To illustrate the proper sequence of the hexadecimal code pattern identifiers, the pattern identifiers will be referred to as string variables with the letter of the variable corresponding to the letter designation of each of the four blocks. The program statements would be:

```
100 A$="0102040201235 68D"
110 B$="8D56230102040201"
120 C$="80402040 80C46AB1"
130 D$="B16AC48040204080"
140 CALL CHAR(96,A$&B$&C$&D$)
```

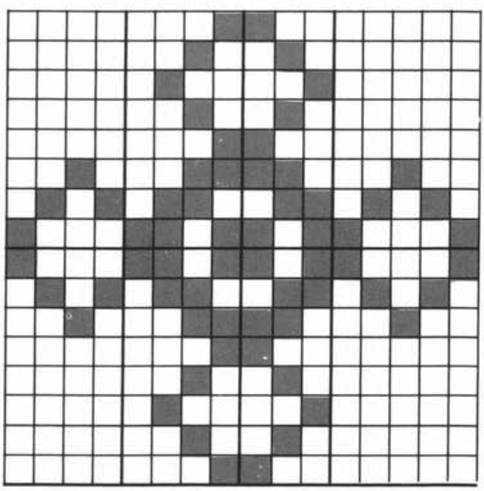
While it isn't necessary to use separate statements in this manner, it may be helpful to do so until the arrangement of pattern identifiers becomes familiar. The same thing could be accomplished by a single program statement:

**100 CALL CHAR(96,"0102040201
23568D8D562301020402018040204
080C46AB16AC48040204080")**

It's only necessary to specify one character code in the CALL CHAR statement; the computer automatically assigns the other three. Even in the case of single-block sprite characters, three character codes will be set aside for the sprite in addition to the specified character code.

Again, the computer will always assign a character code evenly divisible by four to block A, the upper-left portion of the sprite. For the snowflake sprite just identified (Figure 3), character 96 was specified. 96 is evenly divisible by four; therefore, character 96 is assigned to block A. Block B will automatically be assigned 97; block C will be assigned 98, and character 99 will be assigned to block D.

Figure 3. Snowflake Sprite



Hexadecimal Code

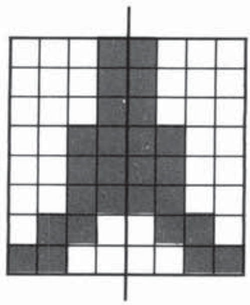
01	80
02	40
04	20
02	40
01	80
23	C4
56	6A
8D	B1
8D	B1
56	6A
23	C4
01	80
02	40
04	20
02	40
01	80

The spaceship sprite shown in Figure 4 will occupy only one of the four blocks. If the character code used for the spaceship were 90, the sprite itself would be placed in block C. Since the next lower number evenly divisible by four is 88, character code 88 would be assigned to block A.

The program line identifying the spaceship sprite would read:

```
150 CALL CHAR(90,"1818183C3C3C66C3")
```

Figure 4. Spaceship Sprite



Hex.
Code
18
18
18
3C
3C
3C
66
C3

Displaying the Sprite

Once a sprite has been defined, the CALL SPRITE statement is used to display it on the screen. The syntax for the CALL SPRITE statement is: CALL SPRITE (sprite number, character code, sprite color, row, column, row velocity, column velocity). Values for row and column velocities, which cause the sprites to move, are optional. The CALL MOTION statement is another method to move a sprite. Both methods will be demonstrated below.

The sprite number can be any number from 1 to 28. The sprite number is always preceded by a #. The character code must correspond to the one specified in the CALL CHAR statement. Since the CALL SPRITE statement requires naming a sprite color, no separate CALL COLOR is needed. The background color of sprites is always transparent, so only the foreground color is named in the statement.

The row and column determine the sprite's location on the screen. Each can be in the range of 1 to 256, but only rows above 193 will be visible on the screen: Rows 193 and 256 are offscreen. Position 1,1 is the upper-left corner of the screen. The values used for row and column will determine the screen placement of the dot in the upper-left corner of the space allotted to the sprite.

The following program lines, combined with lines 100 to 150 above, will clear the screen, color it gray, color the snowflake sprite red and the spaceship black, and place the sprites on the screen. Once the RUN command is given, line 200 will cause the program to continue running until CLEAR (FCTN 4) is pressed.

```
160 CALL CLEAR
170 CALL SCREEN(15)
180 CALL SPRITE(#1,96,7,95,75)
190 CALL SPRITE(#2,90,2,170,125)
200 GOTO 200
```

Magnify the Sprite

This places the two sprites on the screen. However, the snowflake sprite is displaying just the upper-left (block A) part of the sprite. Only that part of a sprite whose character code is named in the CALL SPRITE statement will appear.

A CALL MAGNIFY statement can be used to double the size of sprites or to display multiple-block sprites—our snowflake. It can perform either function separately, or both together, depending upon the *magnification factor* specified.

CALL MAGNIFY(1) would make no change in either the size or appearance of the sprites. CALL MAGNIFY(2) would double the size of all sprites displayed on screen. Rewriting line 200 and adding line 210 to the sample program will simply double the size of the spaceship and the upper-left portion of the snowflake.

```
200 CALL MAGNIFY(2)
210 GOTO 210
```

In order to correct the problem with sprite #1, line 200 must read:

```
200 CALL MAGNIFY (3)
```

This displays the snowflake correctly, but introduces a problem with the spaceship sprite. Extraneous characters have surrounded it. The same condition would exist if a magnification factor of four were used; only the sprites, and the extraneous characters, would be twice as large.

```
200 CALL MAGNIFY(4)
```

The extraneous characters correspond to the ASCII codes of the characters used for the sprite. Block A is character 88 (X); block B is character 89 (Y); and the open bracket ([) is 1 character 91. This problem can be avoided by using one of the "free" character codes (128 to 143) for sprites which will occupy fewer than four blocks.

Note that the CALL MAGNIFY statement affects all sprites in the program, and cannot be used to single out individual sprites.

The extraneous characters can be removed by changing statements 150 and 190 to:

```
150 CALL CHAR(130,"1818183C3C3C66C3")
190 CALL SPRITE(#2,130,2,170,125)
```

All other program lines will remain the same. After making the changes, try running the program with both magnification factors three and four used in line 200. The two sprites will now appear as intended. For the remaining program demonstrations, the magnification factor should be set at three.

Where's the Sprite?

Once sprites have been correctly displayed on screen, various subprograms can be CALLed to manipulate them. Motion can be added, the appearance of a sprite can be altered, and information can be obtained about character pattern identifiers, sprite location, and the distance between sprites.

The POSITION subprogram is used when the numeric values of the screen location of a sprite are desired. In the CALL POSITION statement, sprite number is first specified, followed by two numeric variables. When the statement is executed, the numeric variables are set equal to the values of the sprite's row and column, respectively. Values returned are for the location of the upper-left corner of the four block sprite allotment.

The following changes and additions to the demonstration program will illustrate the operation of the POSITION subprogram:

```
210 CALL POSITION(#1,DR1,DC1)
220 CALL POSITION(#2,DR2,DC2)
230 PRINT TAB(6);"ROW","COL"
240 PRINT "#1: ";DR1,DC1
250 PRINT "#2: ";DR2,DC2
260 STOP
```

When the program is run, the result will be:

	ROW	COL
#1:	**95**	**75**
#2:	**170**	**125**

What Does It Look Like?

Another built-in Extended BASIC subprogram allows the review of character pattern identifiers. The CALL CHARPAT statement is not exclusive to sprites and can also be used with standard user-defined characters as well as with predefined alphanumeric characters.

The CALL CHARPAT statement calls for the character code of the character whose pattern identifier is to be found, followed by a string variable. The result of the string variable will be the named character's 16-character pattern identifier, expressed in hexadecimal. For a multiple-block sprite like the snowflake, a FOR–NEXT loop should be used in order to obtain all four of its pattern identifiers. By making the following alterations to the sample program, the pattern identifiers of the snowflakes and spaceship sprites will be displayed, along with the asterisk (*) and the numeral four (4) and their character pattern identifiers.

```
210 CALL CHARPAT(130,CP2$)::PRINT CP2$
220 FOR CC=96 TO 99
230 CALL CHARPAT(CC,CP1$)
240 PRINT CP1$:: NEXT CC
250 FOR CC=42 TO 52 STEP 10
260 CALL CHARPAT(CC,CP$)
270 PRINT TAB(3);CHR$(CC);"";CP$
280 NEXT CC
290 STOP
```

This will display:

1818183C3C3C66C3
010204020123568D
8D56230102040201
8040204080C46AB1
B16AC48040204080
*** 000028107C102800**
4 00081828487C0808

How Far Is It?

The CALL DISTANCE statement is used to determine the distance between two sprites, expressed as the square of the number of dots separating them. It uses the dot occupying the upper-left corner of the four-block allotment as the point of measurement. CALL DISTANCE can also be used to measure the distance between a sprite and a given screen location.

The statement will include either two sprite numbers, or a sprite number and the row and column values of a screen location, followed by a numeric variable. The actual dot distance is found by taking the square root of the value found for the numeric variable. The results are calculated to eight decimal places, so if it is preferred that values be expressed as integers, the INT function can be added to the appropriate program lines. Changing the demonstration program with the lines below will make the program find the distance between the two sprites and the distance between the snowflake sprite and the upper-left corner of the screen (position 1,1):

```
210 CALL DISTANCE(#1,#2,X)
220 DST=SQR(X)
230 PRINT DST
240 CALL DISTANCE(#1,1,1,Y)
250 DIS=SQR(Y)
260 PRINT DIS
270 STOP
280 REM DELETE THIS LINE
```

The results of the sample program will be:

90.13878189
119.6327714

Moving Sprites

CALL LOCATE is used to change the screen location of a sprite. It does so immediately upon execution of the program line, producing an abrupt change rather than gradual motion. The syntax is CALL LOCATE(sprite number,row,column).

```
210 CALL LOCATE(#2,1,200)
220 FOR DELAY=1 TO 1000 :: NEXT DELAY
230 CALL LOCATE(#2,15,15)
240 FOR DELAY=1 TO 1000 :: NEXT DELAY
250 CALL LOCATE(#2,170,125)
260 GOTO 210
270 REM DELETE THIS LINE
```

These lines move the spaceship sprite from its original position first to a point near the upper right of the screen, then near the upper left, then back to its original location. The program will continue to run until CLEAR (FCTN 4) is pressed.

If a change in the character pattern is wanted without otherwise affecting the sprite, the CALL PATTERN statement is used. It can be used to completely reshape a sprite or to make more subtle changes in appearance. When combined with other statements, it can be used to simulate the visual effects of motion. By changing the sample program lines as follows, the spaceship sprite will change from vertical to horizontal orientation, then back again.

```
210 FOR D=1 TO 1000 :: NEXT D
220 CALL CHAR(140,"80C0783F3F78C080")
230 CALL PATTERN(#2,140)
240 FOR D=1 TO 1000 :: NEXT D
250 CALL PATTERN(#2,130)
260 GOTO 210
```

The CALL CHAR statement containing the pattern identifier for the modified sprite need not immediately precede the CALL PATTERN statement; it can be included anywhere in the program before the CALL PATTERN statement.

Motion of the sprites can be accomplished in either of two ways: by specifying row and column velocities in the CALL SPRITE statement, or by adding a CALL MOTION statement later in the program. In the CALL SPRITE statement, row and column velocities are specified following row and column values, which then become the sprite's starting point; in CALL MOTION statements, row and column velocities follow the sprite number.

Values for row and column velocity fall within the range of −128 to 127. The closer the value is to zero, the slower the motion will be. Negative values for row velocity move the sprite upward while negative column velocity moves the sprite to the left. Conversely, positive values move the sprite down or to the right. A value of zero for row velocity means that there is no vertical movement; likewise a column velocity of zero prevents horizontal movement. When unidirectional movement is desired, zero must be specified as a velocity for the direction in which motion is not wanted.

Examples of both methods of initiating sprite motion can be added to the demonstration program as follows:

Change line 180 and lines 210 to 240, and add line 250:

```
180 CALL SPRITE(#1,96,7,95,75,50,-50)
210 REM
220 REM
230 REM
240 CALL MOTION(#2,-5,0)
250 GOTO 250
```

These changes cause the snowflake sprite to move down and to the left at a diagonal, while the spaceship moves slowly upward. Notice that motion begins as soon as the program line is executed and is continuous until the program is stopped. The spaceship cycles from the bottom to the top of the screen over and over. A time-delay FOR–NEXT loop can be inserted to end the movement of the sprite by deleting the sprite. Differences in sprite velocity necessitate experimenting to find the upper limit needed in the FOR–NEXT loop. In this case, changing the program to:

```
250 FOR DELAY=1 TO 2200 :: NEXT DELAY
260 CALL DELSPRITE(#2)
270 GOTO 270
```

allows the spaceship to move as far as the top of the screen when it will disappear. The DELSPRITE subprogram deletes the specified sprite as soon as the program statement is executed. DELSPRITE can also be used to clear all sprites from the screen by writing CALL DELSPRITE(ALL).

To restore the sprite to the screen after motion is stopped, rewrite line 270 to send the computer back to line 240, or follow line 260 with a new CALL SPRITE statement duplicating line 190. As it was written, once the CALL DELSPRITE statement is executed, the spaceship will not return to the screen.

Motion begins as soon as the CALL SPRITE or CALL MOTION statement is reached. The start of motion can easily be controlled, however. Adding a KEY subprogram allows motion to begin only after a certain key has been pressed. Add these lines in place of the three REM statements:

```
210 CALL KEY(0,K,S)
220 IF S=0 THEN 210
230 IF K=32 THEN 240 ELSE 210
```

Now, when the program is run, the spaceship will not move until the space bar is pressed.

Different values for row and column velocity will, of course, change both the speed and direction of sprite movement. Changing the values may also affect the angle at which the sprites move. The closer the values, the greater the angle at which the sprites move. If values of 50 were specified for both row and column velocity, the angle would be 45 degrees. If row velocity were increased to 90, the angle of movement would be smaller, and motion would be more vertical. Accordingly, a greater value for column velocity would make movement more horizontal. This is true even if a negative value is specified for either row or column velocity while the other is positive, as demonstrated by the motion of the snowflake sprite. Try changing the values of the row and column velocities given for both sprites. Experimenting with different values demonstrates how they change the speed and angle of sprite motion.

Detecting Collision

The CALL COINC statement performs a function especially useful in game programming. It instructs the computer to monitor sprite movement to determine when two or more sprites occupy the same position, or are within a certain number of dots of each other. It can also be used to determine when a sprite reaches a specific screen location, or passes within a certain number of dots of the screen position.

If information is needed for all sprites, then CALL COINC(ALL) is used in the program; otherwise, CALL COINC is followed by two sprite numbers, or a sprite number and the row and column of a screen location, a tolerance value, and a numeric variable. Tolerance is simply the number of dots which may separate the two sprites or the sprite and screen position in order for coincidence to exist. Again, the dot in the upper-left corner of the sprite is used for measurement purposes.

When the CALL COINC statement is executed, the computer assigns a value to the numeric variable in the statement. If there is no coincidence, the numeric variable is set equal to zero; if there is coincidence within the allowance of the tolerance specified, the value is set equal to −1. Instructions can be given to the computer to act based on the value of the variable. IF-THEN statements employed in this way can be used to change a score or screen color, sound a tone, or even play a tune by using the proper sequence of CALL SOUND statements.

It should be remembered that the coincidence of two sprites, or of a sprite and screen location, does not have to be visible.

Program lines 250 to 390 below demonstrate the operation of the COINC subprogram.

```
250 CALL COINC(#1,#2,10,C1)
260 CALL COINC(#2,1,125,1,C2)
270 PRINT C1
280 IF C1=-1 THEN 300 ELSE 290
290 IF C2=-1 THEN 350 ELSE 250
300 CALL SCREEN(11)
310 CALL SOUND(100,262,2)
320 FOR D=1 TO 200 :: NEXT D
330 CALL SCREEN(15)
340 GOTO 250
350 CALL SCREEN(4)
360 CALL SOUND(100,523,2)
370 FOR D=1 TO 200 :: NEXT D
380 CALL SCREEN(15)
390 GOTO 250
```

Line 270 will continuously print the value of C1 as the program runs. If C1 = −1, coincidence of the two sprites exists, and control shifts to line 300, where the screen color is changed to yellow and middle-C is sounded. When the spaceship sprite comes within one dot of the top of the screen, C2 is set equal to −1, and control moves to line 350, where the screen becomes green, and C above middle-C is sounded. The program will continue to run until CLEAR (FCTN 4) is pressed. It will probably be necessary to allow the spaceship sprite to cycle the screen several times before coincidence with the snowflake sprite is detected.

You may notice that sometimes the sprites appear to collide but no coincidence is detected. Coincidence is only detected when the CALL COINC statement is being executed—in this program, line 250. One way to avoid this problem is to check coincidence often. This solution, though, tends to make the program longer than it needs to be, thus slowing it down. The best solution is to keep the loop (in the example, program lines 250–290) to as few lines as possible and adjust the tolerance. Since too large a tolerance will cause coincidence too often, it is best to experiment with different values.

Demonstration Program

These are the essentials of sprite programming, and the demonstrations used are only representative of what can be done with sprites. After experimenting with the different subprograms, you'll discover how to best use sprites in your own programs.

Below is a complete listing of the sprite demonstration program.

Sprite Demonstration

```
100 CALL CLEAR
110 PRINT TAB(5):"****SPRITE DEMO****"
120 PRINT :: PRINT "DESIGNED TO ACCOMPANY"
130 PRINT """ A BEGINNER'S GUIDE TO
    {6 SPACES}SPRITES IN TI EXTENDED
    {5 SPACES}BASIC """
170 FOR D=1 TO 1000 :: NEXT D
180 CALL CLEAR
190 PRINT "THIS DEMONSTRATION FOLLOWS  THE S
    EQUENCE OF THE ARTICLE."
200 PRINT :: PRINT "THE PROGRAM STEPS USED A
    RE  THE SAME AS THOSE USED IN{3 SPACES}T
    HE ARTICLE."
210 PRINT :: PRINT "AT THE END OF EACH DEMO.
     A  TONE WILL SOUND."
220 PRINT :: PRINT "THEN PRESS LETTER Q TO C
    ON- TINUE  WITH THE NEXT DEMO."
230 FOR D=1 TO 1000 :: NEXT D
240 CALL CLEAR
250 A$="0102040201235 68D"
260 B$="8D5623010204020 1"
270 C$="80402040 80C46AB1"
280 D$="B16AC48040204080"
290 CALL CHAR(96.A$&B$&C$&D$)
300 CALL CHAR(90."181818 3C3C3C66C3")
310 CALL SCREEN(15)
320 CALL SPRITE(#1.96,7,95.75)
330 CALL SPRITE(#2,90.2,170,125)
335 FOR D=1 TO 500 :: NEXT D
340 GOSUB 2500
350 CALL MAGNIFY(2)
360 DISPLAY AT(3,3):"MAG. FACTOR 2"
370 FOR D=1 TO 500 :: NEXT D
```

```
380 CALL MAGNIFY(3)
390 DISPLAY AT(3,3)ERASE ALL:"MAG. FACTOR 3"
400 FOR D=1 TO 500 :: NEXT D
410 CALL MAGNIFY(4)
420 DISPLAY AT(3,3)ERASE ALL:"MAG. FACTOR 4"
430 FOR D=1 TO 500 :: NEXT D
440 CALL CLEAR
450 CALL CHAR(130,"1818183C3C3C66C3")
460 GOSUB 3000
465 DISPLAY AT(2,3):"MAG. FACTOR 3"
467 DISPLAY AT(3,3):"UNWANTED CHARACTERS NOW
    RE-{3 SPACES}MOVED WHEN ""FREE"" CHAR-
    {3 SPACES}ACTER CODE USED"
468 FOR D=1 TO 500 :: NEXT D
470 FOR D=1 TO 500 :: NEXT D
480 CALL MAGNIFY(4)
490 DISPLAY AT(3,3)ERASE ALL:"MAG. FACTOR 4"
500 FOR D=1 TO 500 :: NEXT D
510 CALL MAGNIFY(3)
520 DISPLAY ERASE ALL
530 GOSUB 2500
535 DISPLAY AT(2,3):"POSITION DEMO"
536 FOR D=1 TO 500 :: NEXT D
540 CALL POSITION(#1,DR1,DC1)
550 CALL POSITION(#2,DR2,DC2)
560 PRINT TAB(5);"ROW"," COL"
570 PRINT "#1: ";DR1,DC1
580 PRINT "#2: ";DR2,DC2
590 FOR D=1 TO 500 :: NEXT D
600 GOSUB 2500
610 CALL CLEAR
620 GOSUB 3000
625 DISPLAY AT(2,3):"CHARPAT DEMO"
626 FOR D=1 TO 300 :: NEXT D
630 CALL CHARPAT(130,CP2$):: PRINT CP2$
640 FOR CC=96 TO 99
650 CALL CHARPAT(CC,CP1$)
660 PRINT CP1$ :: NEXT CC
670 FOR CC=42 TO 52 STEP 10
680 CALL CHARPAT(CC,CP$)
690 PRINT TAB(2);CHR$(CC);" ";CP$ :: NEXT CC
700 FOR D=1 TO 500 :: NEXT D
710 GOSUB 2500
720 CALL CLEAR
730 GOSUB 3000
735 DISPLAY AT(2,3):"DISTANCE DEMO"
736 FOR D=1 TO 300 :: NEXT D
740 CALL DISTANCE(#1,#2,X)
750 DST=SQR(X)
760 PRINT DST
```

```
770 CALL DISTANCE(#1,1,1,Y)
780 DIS=SQR(Y)
790 PRINT DIS
800 FOR D=1 TO 500 :: NEXT D
810 GOSUB 2500
820 CALL CLEAR
830 GOSUB 3000
835 DISPLAY AT(2,3):"LOCATE DEMO"
836 FOR D=1 TO 300 :: NEXT D
840 CALL LOCATE(#2,1,200)
850 FOR D=1 TO 500 :: NEXT D
860 CALL LOCATE(#2,16,16)
870 FOR D=1 TO 500 :: NEXT D
880 CALL LOCATE(#2,170,125)
890 FOR D=1 TO 500 :: NEXT D
900 GOSUB 2500
905 DISPLAY AT(2,3)ERASE ALL:"PATTERN DEMO"
906 FOR D=1 TO 300 :: NEXT D
910 CALL CHAR(140,"80C0783F3F78C080")
920 CALL PATTERN(#2,140)
930 FOR D=1 TO 500 :: NEXT D
940 CALL PATTERN(#2,130)
950 GOSUB 2500
960 GOSUB 3500
965 DISPLAY AT(2,3)ERASE ALL:"MOTION DEMO"
966 FOR D=1 TO 300 :: NEXT D
967 DISPLAY ERASE ALL
970 FOR D=1 TO 3000 :: NEXT D
980 GOSUB 2500
990 GOSUB 3500
995 DISPLAY AT(2,3)ERASE ALL:"DELSPRITE DEMO
    "
996 FOR D=1 TO 300 :: NEXT D
997 DISPLAY ERASE ALL
1000 FOR D=1 TO 2200 :: NEXT D
1010 CALL DELSPRITE(#2)
1020 FOR D=1 TO 1000 :: NEXT D
1030 GOSUB 2500
1040 CALL CLEAR
1045 CALL SPRITE(#2,130,2,170,125)
1050 GOSUB 4000
1055 DISPLAY AT(2,3):"USE OF CALL KEY TO INI
     TIATE MOTION"
1056 FOR D=1 TO 300 :: NEXT D
1057 DISPLAY ERASE ALL
1060 FOR D=1 TO 2000 :: NEXT D
1080 GOSUB 2500
1090 GOSUB 3500
1095 DISPLAY AT(2,3)ERASE ALL:"COINC DEMO"
1096 FOR D=1 TO 300 :: NEXT D
```

```
1100 CALL COINC(#1,#2,15,C1)
1110 CALL COINC(#2,1,125,1,C2)
1120 PRINT C1
1130 IF C1=-1 THEN 1150 ELSE 1140
1140 IF C2=-1 THEN 1200 ELSE 1100
1150 CALL SCREEN(11)
1160 CALL SOUND(100,262,2)
1170 FOR D=1 TO 200 :: NEXT D
1180 CALL SCREEN(15)
1190 GOTO 1100
1200 CALL SCREEN(4)
1210 CALL SOUND(100,523,2)
1220 FOR D=1 TO 200 :: NEXT D
1230 CALL SCREEN(15)
1240 GOTO 1100
1250 STOP
2500 DISPLAY BEEP
2510 CALL KEY(0,K,S)
2520 IF S=0 THEN 2510
2530 IF K=81 THEN 2540 ELSE 2510
2540 RETURN
3000 CALL SCREEN(15)
3010 CALL SPRITE(#1,96,7,95,75)
3020 CALL SPRITE(#2,130,2,170,125)
3030 CALL MAGNIFY(3)
3040 RETURN
3500 CALL SPRITE(#1,96,7,95,75,50,-50)
3510 CALL MOTION(#2,-5,0)
3520 RETURN
3530 STOP
4000 DISPLAY AT(3,3):"PRESS SPACE BAR TO STA
     RT MOTION OF SPACESHIP SPRITE"
4005 FOR D=1 TO 200 :: NEXT D
4006 DISPLAY ERASE ALL
4010 CALL KEY(0,K,S)
4020 IF S=0 THEN 4010
4030 IF K=32 THEN 4040 ELSE 4010
4040 CALL MOTION(#2,-5,0)
4050 RETURN
```

Sprite Editor

Larry Long

Here's a way to get maximum use of sprites on the TI-99/4A—and a program that generates listings for your sprite creations. Requires Extended BASIC.

A very powerful yet often unused feature of the TI-99/4A is its ability to display and control sprites. With the 99/4A and the Extended BASIC Module, it's possible to generate 28 sprites for display and independent simultaneous movement. Program 1 should convince any doubters that this can be done. Although a lot of colored letters floating around the screen are a bit pointless, if we can modify and control the sprites, we will have a most useful feature.

Sprites can be designed by drawing on a piece of graph paper and then converting the on/off pixels to a hexadecimal number. If the two largest sizes of sprites are used, the hexadecimal number describing the shape of the sprite would be 64 characters long (for a more extensive discussion on sprite creation see "A Beginner's Guide to Sprites" elsewhere in this book). A solution is a sprite editor that will allow us to draw the pattern we want on the screen and then have the computer create the program we need to make that sprite pattern. Program 2 will do exactly that, and more. It will allow us to edit the sprite pattern. Then, when we press the L key, it will display a complete listing that would, if copied on paper and then entered into the computer, provide a sprite and the necessary routine to control its movement.

Your Options

When you run the program, the first display screen will be a design grid with a box-shaped cursor. The area under the cursor will initially be white (signifying an off pixel). Press 1 to change the color beneath the cursor to black (representing an on pixel) or to move the cursor about the grid using the arrow keys. To turn off a particular pixel, press 0 and the background color will be returned to white. When you have completed your design, press the P key to see it displayed as a sprite.

At this point, you are given several options. You can magnify your newly constructed sprite (M key), change its color (C key), change its background color (B key), or set it in motion (E, S, D, X keys). If you are not pleased with the sprite's shape, you can modify it by striking the T key or (if the changes required are quite drastic) simply press the A key to start with a fresh grid. On the other hand, if you are satisfied with your sprite and its color and directional parameters, press the L key to create the BASIC statements needed to achieve these effects.

If using the sprite editor is your only concern, then skip the rest of this article and go straight to Program 2 and enjoy this easy access to sprites.

How the Editor Works

To understand what makes the editor work, let's take a general overview of the program:

Lines	
100–260	Set up screen display.
270–460	The main loop of the designing portion of the program.
470–680	Evaluate the design, put its values in an array, read the values in the array, convert them to hexadecimal numbers, and then build a 64-character string to describe the sprite pattern.
690–770	Put the sprite on the screen and display new program instructions.
780–930	Main loop of the implementation portion of the program.
940–980	Change size of sprite.
1000–1150	Display a listing of the sprite program.
1160–1220	Change the color of the sprite and screen.

A cursor is needed to indicate where you are located on the design grid. I chose to use a sprite (line 220) because I could move it around freely without disturbing the display under it. Repositioning the cursor is accomplished in line 380 with a CALL LOCATE. The arrow keys reposition the cursor, and the ENTER key changes the area under the cursor.

What makes "Sprite Editor" so valuable is its ability to generate the hexadecimal pattern for the sprite. The loop from line 500 through line 560 determines the character in each position of the design grid and stores that value in the array B (R,C). Line 570 provides a string with all of the possible hexadecimal digits placed in ascending order. Line 580 sets M$ to null. The loop from line 590 to line 630 evaluates the array elements and converts each row in the left half of the design grid to a pair of hexadecimal digits and concatenates them to M$. Line 620 is probably the most significant line in this loop, as it provides the hexadecimal numbers. It causes the computer to look at a particular digit (element) in HEX$ determined by the values calculated for HIGH and LOW. Lines 630–680 perform the same operation as 590–630, only for the right half of the design grid.

Line 690 assigns the hexadecimal numbers to ASCII characters 104, 105, 106, and 107. It is necessary to specify only the first character number in the CALL CHAR statement. When this feature is used, it is required that you start with a character that is evenly divisible by 4. Line 730 actually displays the sprite.

Lines 740–770 provide instructions for the implementation portion of the program. Lines 780–830 check for specific key presses and provide appropriate branching to list the program; end the program; start from the beginning; change the background color; modify the existing sprite; change sprite size; or change sprite color. Lines 840–920 check for arrow key presses and then increment or decrement sprite speed.

Lines 940–980 change sprite size. Lines 1000–1150 display a program listing that would generate a sprite like the one designed by the Sprite Editor. One problem with listing the program is displaying the quote character. The computer interprets it to mean that you want to end the PRINT statement. The solution is to redefine an unused character (I chose the lowercase *n*) to look like the quote character.

Finally, lines 1160–1220 allow you to change the color of the sprite and screen.

Program 1. Sprite Generation

```
100 CALL MAGNIFY(2):: FOR X=1 TO 28 :: CALL
    SPRITE(#X,64+X,X/2,96,128,INT(RND*100)-5
    0,INT(RND*100)-50):: NEXT X :: GOTO 100
```

Program 2. Sprite Editor

```
100 REM SPRITE EDITOR
110 DIM B(16,16):: SC=8
```

```
130 C1=7
140 CALL CHAR(100,"")
150 CALL CHAR(101,"FFFFFFFFFFFFFFFF")
160 CALL CHAR(102,"FFFFC3C3C3C3FFFF")
170 CALL COLOR(9,2,16)
180 CALL CLEAR
190 DISPLAY AT(1,10):"SPRITE EDITOR"
200 FOR R=1 TO 16 :: CALL HCHAR(4+R,2,100,16
    ):: NEXT R
210 CALL MAGNIFY(1)
212 IF K=84 THEN GOTO 217
215 CALL SCREEN(8)
217 CALL DELSPRITE(ALL)
220 CALL SPRITE(#28,102,14,32,8)
225 CALL HCHAR(21,1,32,31):: CALL HCHAR(22,1
    ,32,31)
230 DISPLAY AT(22,2):"E=UP X=DOWN S=LEFT D=R
    IGHT"
240 DISPLAY AT(23,2):"PRESS 1 - PIXEL ON ,0
    - OFF"
250 DISPLAY AT(24,2):"PRESS P TO DISPLAY SPR
    ITE"
260 R=1 :: C=1
265 KHAR=100
270 CALL KEY(0,K,S)
271 IF S=0 THEN 270
272 IF K=48 THEN KHAR=100
274 IF K=49 THEN KHAR=101
280 IF K=83 THEN C=C-1 :: GOTO 320
290 IF K=68 THEN C=C+1 :: GOTO 320
300 IF K=69 THEN R=R-1 :: GOTO 320
310 IF K=88 THEN R=R+1 :: GOTO 320
312 IF K=80 THEN 470
320 IF C<1 THEN C=16
330 IF C>16 THEN C=1
340 IF R<1 THEN R=16
350 IF R>16 THEN R=1
380 CALL LOCATE(#28,(8*R)+25,8*C+1)
420 CALL HCHAR(4+R,1+C,KHAR)
430 CALL SOUND(20,200,5)
460 GOTO 270
470 CALL DELSPRITE(ALL)
480 CALL HCHAR(21,1,32,128)
490 DISPLAY AT(22,2):"PLEASE WAIT WHILE I TH
    INK."
500 FOR R=1 TO 16
510 FOR C=1 TO 16
520 CALL GCHAR(4+R,1+C,GC)
530 GC=GC-100
540 B(R,C)=GC
```

```
550 NEXT C
560 NEXT R
570 HEX$="0123456789ABCDEF"
580 M$=""
590 FOR R=1 TO 16
600 LOW=B(R,5)*8+B(R,6)*4+B(R,7)*2+B(R,8)+1
610 HIGH=B(R,1)*8+B(R,2)*4+B(R,3)*2+B(R,4)+1
620 M$=M$&SEG$(HEX$,HIGH,1)&SEG$(HEX$,LOW,1)
630 NEXT R
640 FOR R=1 TO 16
650 LOW=B(R,13)*8+B(R,14)*4+B(R,15)*2+B(R,16
    )+1
660 HIGH=B(R,9)*8+B(R,10)*4+B(R,11)*2+B(R,12
    )+1
670 M$=M$&SEG$(HEX$,HIGH,1)&SEG$(HEX$,LOW,1)
680 NEXT R
690 CALL CHAR(104,M$)
700 CALL MAGNIFY(3)
710 MM=3
720 M=4
730 CALL SPRITE(#1,104,C1,50,170,0,0)
740 DISPLAY AT(21,2):"C COLOR  M MAGNIFY  T
    EDIT"
750 DISPLAY AT(22,2):"A ERASE  Q QUIT  B BAC
    KGRD"
760 DISPLAY AT(23,2):"E=UP X=DOWN S=LEFT D=R
    IGHT"
770 DISPLAY AT(24,8):"L LISTS PROGRAM"
780 CALL KEY(0,K,S)
790 IF K=76 THEN GOTO 1000
800 IF K=81 THEN GOTO 990
810 IF K=65 THEN GOTO 100
812 IF K=66 THEN GOSUB 1200
815 IF K=84 THEN GOTO 210
820 IF K=77 THEN GOTO 940
830 IF K=67 THEN GOTO 1160
840 IF K=83 THEN H=H-2
850 IF K=68 THEN H=H+2
860 IF K=69 THEN V=V-2
870 IF K=88 THEN V=V+2
880 IF V>120 THEN V=120
890 IF V<-120 THEN V=-120
900 IF H>120 THEN H=120
910 IF H<-120 THEN H=-120
920 CALL MOTION(#1.V,H)
930 GOTO 780
940 CALL MAGNIFY(M)
950 MM=M
960 IF M=3 THEN M=4 ELSE M=3
970 FOR D=1 TO 20 :: NEXT D
```

```
980 GOTO 780
990 STOP
1000 REM PROGRAM LISTER
1005 CALL SCREEN(8)
1010 CALL CHAR(110,"002424")
1020 CALL CLEAR
1030 PRINT "{6 SPACES}PROGRAM LISTING"
1035 CALL DELSPRITE(ALL)
1040 PRINT
1050 PRINT ">100 CALL CHAR(104,n";:: FOR W=1
     TO 64 :: PRINT SEG$(M$,W,1);:: NEXT W
     :: PRINT "n)"
1055 PRINT ">105 CALL SCREEN(";SC;")"
1060 PRINT ">110 CALL MAGNIFY(";MM;")"
1070 PRINT ">120 CALL SPRITE(#1,104,";C1;",1
     50,150,";V;",";H;")"
1080 PRINT ">130 CALL KEY(0,K,S)"
1090 PRINT ">140 IF K=68 THEN H=H+2"
1100 PRINT ">150 IF K=83 THEN H=H-2"
1110 PRINT ">160 IF K=88 THEN V=V+2"
1120 PRINT ">170 IF K=69 THEN V=V-2"
1130 PRINT ">180 CALL MOTION(#1,V,H)"
1140 PRINT ">190 GOTO 130"
1150 PRINT :: PRINT :: PRINT :: PRINT :: PRI
     NT
1155 DISPLAY AT(21,3):"A - ERASE{3 SPACES}Q
     - QUIT"
1156 CALL KEY(0,K,ST):: IF ST=0 THEN 1156
1157 IF K=81 THEN GOTO 990
1158 IF K=65 THEN GOTO 100
1159 GOTO 1156
1160 C1=C1+1 :: IF C1>16 THEN C1=2
1170 CALL COLOR(#1,C1)
1180 CALL KEY(0,K,S):: IF S THEN 1180 ELSE 7
     80
1200 REM SCREEN COLOR CHANGE
1210 SC=SC+1 :: IF SC=17 THEN SC=2
1220 CALL SCREEN(SC)
1230 CALL KEY(O,K,S):: IF S THEN 1230 ELSE
     RETURN
```

Runway 180
Using Sprites in Extended BASIC

James Dunn

The efficient, remarkable sprite-handling ability of Extended BASIC is clearly evident in this game. The author discusses creating sprites and explores sprite manipulation. There are several valuable pointers here for those interested in graphics, animation, or game programming.

One of the biggest problems in designing an arcade-type game in BASIC is that BASIC can move only one character at a time, usually slowly and not very smoothly. Ideally, we need the ability to move an object independently of the operation of the main program. Once set in motion, the object would continue in motion until acted upon by a new command from the main program. Sprites accomplish this.

Although a sprite is a type of subprogram that runs concurrently with a main program, the main program first must create the sprite, define its shape, and set it in motion. A sprite then continues its motion without requiring continuous control from the main program, except that the main program may at any time test the sprite for position, change the color or pattern, delete, or change its motion (see "A Beginner's Guide to Sprites" and "Sprite Editor" in this chapter).

Included in TI-99/4A Extended BASIC are 11 commands to control sprites: CALL COLOR, CALL CHAR, CALL SPRITE, CALL PATTERN, CALL MAGNIFY, CALL MOTION, CALL POSITION, CALL LOCATE, CALL DISTANCE, CALL COINC, and CALL DELSPRITE. To illustrate the use of these commands, we'll look at an airplane landing game, "Runway 180." Try some examples for yourself to get a feel for sprite programming.

Creating Sprites

Certain considerations must be taken into account before sprites are created. If a special graphics character is to be used for the sprite, the character must be created by use of CALL

CHAR. For example, in the game there are three special characters defined for the aircraft. One is with the wheels up (lines 430–460), one is with the wheels down (lines 510–540), and one is debris after a crash (lines 550–580).

To create a special character, it's necessary to redefine an existing standard character. The standard characters correspond to the numbers 32 through 127 (part of what's called the ASCII number code). The new pattern is created by using CALL CHAR and is referenced by its ASCII number.

Before we choose which ASCII number to use, we must examine some other factors. CALL MAGNIFY can enlarge a sprite to one of four magnification factors. Factor four is used in the game (line 630). This enlarges the sprites to double-size pixels and uses a block of four sequential characters. The ASCII number used to define the sprite must be evenly divisible by four and represents the upper-left character in the block of four. The next three ASCII numbers represent the lower-left, upper-right, and lower-right characters respectively in the block of four.

The sprite may be colored independently of the other characters in the same character set. In addition, the sprite with the lower sprite number (this is a different number from the ASCII number) will pass in front of (that is, *over*) the higher numbered sprite. Since the aircraft should pass in front of the tower, it should have a lower sprite number for each of its three configurations (line 610).

To set up a list of sprites, first number the lines on a sheet of paper from 32 to 143. Then beside each number, write what set it belongs to (set 1 to 14). Since you may want to use letters or numbers in a screen display at the same time, mark out ASCII numbers 48 through 57 and 65 through 90. The remaining ASCII numbers can be used to define special characters for graphics and sprites.

For sprites, using CALL MAGNIFY(4), select four sequential numbers starting at one of the numbers divisible by four. Now you are ready to use CALL SPRITE.

CALL CLEAR will not remove a sprite from the screen. To completely clear the screen, you must also use CALL DELSPRITE (line 1350).

Sprites in Motion

Now that the sprite has been created, there are two ways of moving it around the screen. Let's call these two methods *absolute* and *relative*. The absolute method uses exact row and column positions via the CALL LOCATE command. The relative method uses row and column motion values via the CALL MOTION command.

The absolute method uses a loop with CALL JOYST to increment row and column variables, and then a CALL LOCATE to move the sprite one step each time the loop is executed. This is analogous to nonsprite methods of animation. The drawback in using this method is that the sprite does not move independently; the main program causes the move. A modified form of this method is used for the stall subroutine (line 1470) and the new approach routine (line 1380).

The relative method is similar, using a loop with CALL JOYST to increment row and column *motion* variables which are used in a CALL MOTION command. This allows the sprites to continue moving independently of the main program. By this method, the runway stripe is moved horizontally only (line 680) and the aircraft vertically only (also line 680).

The sprite's shape may be changed anytime during the program by using CALL PATTERN to substitute a different ASCII character number and therefore a different pattern. When the fire button is depressed (line 1130), the aircraft landing gear comes down (line 1190). The pattern is changed again if the aircraft crashes (line 1720).

Testing for Game Conditions

During the operation of the program, it may become necessary to test for certain conditions. For example, we see if the aircraft has touched down on the runway (line 690), if the tower has reached the left side of the screen (line 700), or if the aircraft is going off the top of the screen (line 710). CALL COINC is used to test for these conditions.

However, there is a problem with this method. Since the main program tests for coincidence only when CALL COINC is executed and since the sprite moves independently of the main program, it is quite possible to miss an exact coincidence when it occurs. For this reason a tolerance factor is included in CALL COINC. So the test is really for a range of + or − tolerance. If the tolerance is too large, coincidence can be returned too early. If the tolerance is too small, coincidence can be missed altogether. How large the tolerance should be depends upon two things: the speed of the sprite and the speed of the loop which is testing for coincidence.

The test for the tower reaching the left side of the screen is in both the main loop (line 700) and the stall loop (line 1480). The tolerance in the stall loop is much smaller because the execution speed is so fast and the sprite moves so slowly that coincidence is actually read twice before the sprite leaves the tolerance range. Trial and error is the only way to find out how large the tolerance should be.

However, after programming this game, it's obvious that very fast-moving sprites will require tolerance ranges that will make arcade-style, fast-action games nearly impossible in Extended BASIC. The problem is that the coincidence test is executed from the main program. If it were part of the sprite subprogram instead, it would be possible to keep the tolerance very small.

CALL POSITION and CALL DISTANCE both suffer from the same problem as CALL COINC. By the time a position or distance can be computed and returned to the main program, the sprite has moved elsewhere. But it's possible to stop the sprite by using a CALL MOTION before using CALL POSITION or CALL DISTANCE (line 1330), then to restart whatever motion is required.

Despite a few shortcomings, the sprite capabilities in Extended BASIC are remarkable. For true arcade-type play, machine language is still necessary, but Extended BASIC sprites will carry the programmer a lot closer to this goal.

Runway 180

```
130 CALL CLEAR :: CALL SCREEN(5):: CALL COLO
    R(1,16,1,2,16,1,3,16,1,4,16,1,5,16,1,6,1
    6,1,7,16,1,8,16,1)
140 DISPLAY AT(10,9):USING "RUNWAY 180"
150 FOR B=0 TO 30 STEP 2 :: CALL SOUND(-10,1
    10,30,110,30,2500,30,-8,B):: CALL SOUND(
    -10,110,30,110,30,4000,30,-8,B):: NEXT B
160 CALL CLEAR :: DISPLAY AT(10,9):USING "PR
    ESS" :: DISPLAY AT(12,9):USING "I-FOR IN
    STRUCTIONS"
170 DISPLAY AT(14,14):USING "OR" :: DISPLAY
    AT(16,9):USING "G-FOR GAME"
```

```
180 CALL KEY(Ø,K,S):: IF S<>1 THEN 18Ø
190 IF K=1Ø3 THEN 33Ø
200 IF K=1Ø5 THEN 22Ø
210 PRINT "ALPHA LOCK MUST BE OFF" :: PRINT
    :: PRINT "TRY AGAIN" :: FOR DELAY=1 TO 6
    ØØ :: NEXT DELAY :: GOTO 16Ø
220 CALL CLEAR :: PRINT "YOU ARE PILOTING A
    JET" :: PRINT :: PRINT "AIRCRAFT WHICH H
    AS BEEN " :: PRINT :: PRINT "CLEARED TO
    LAND ON": :
230 PRINT "RUNWAY 18Ø." :: PRINT :: PRINT ::
     GOSUB 31Ø
240 CALL CLEAR :: PRINT "USE YOUR JOYSTICK T
    O CONTROL" :: PRINT :: PRINT "SINK RATE
    AND AIRSPEED. ": :
243 PRINT "JOYSTICK CONTROL-" :: PRINT
245 PRINT "LEFT: ACCELERATE" :: PRINT "RIGHT
    : BRAKE" :: PRINT "UP: DECREASE SINK RAT
    E"
247 PRINT "DOWN: INCREASE SINK RATE" :: PRIN
    T
250 PRINT "FIREBUTTON CONTROLS LANDING" :: P
    RINT :: PRINT "GEAR." :: PRINT :: PRINT
    :: GOSUB 31Ø :: CALL CLEAR
260 PRINT "TO RECOVER FROM A STALL" :: PRINT
     :: PRINT "INCREASE AIRSPEED ABOVE 6Ø."
    :: PRINT :: PRINT "IF YOU CANNOT STOP BE
    FORE": :
270 PRINT "TOWER REACHES LEFT SIDE OF" :: PR
    INT :: PRINT "SCREEN, INCREASE AIRSPEED"
     :: PRINT
280 PRINT "TO 6Ø AND LIFT OFF FOR " :: PRINT
     :: PRINT "ANOTHER PASS." :: PRINT :: PR
    INT :: GOSUB 31Ø :: CALL CLEAR
290 PRINT "YOU MAY HAVE FOUR PASSES" :: PRIN
    T :: PRINT "AT THE RUNWAY....." :: PRINT
     :: PRINT "BEWARE OF THE WIND SHIFTS!" :
    : PRINT :: PRINT
300 PRINT "GOOD LUCK!!!!" :: PRINT :: PRINT
    :: PRINT :: PRINT :: GOSUB 31Ø :: GO TO
    33Ø
310 PRINT :: DISPLAY AT(24,1):USING "HIT ANY
     KEY TO CONTINUE"
320 CALL KEY(Ø,R8,S8):: IF S8<>1 THEN 32Ø EL
    SE RETURN
330 A1=1
340 REM INITIALIZE
350 A=Ø :: B=-75 :: LG=Ø :: CALL SCREEN(2)
360 CALL CLEAR :: CALL CHAR(33,"FFFFFFFFFFFF
    FFFF"):: CALL COLOR(1,8,1)
```

```
370 LC=0 :: FOR Z=1 TO 16 :: CALL HCHAR(Z,1,
    33,32):: NEXT Z
380 CALL CHAR(42,"FFFFFFFFFFFFFFFF"):: CALL
    COLOR(2,13,1)
390 FOR Z=17 TO 20 :: CALL HCHAR(Z,1,42,32):
    : NEXT Z
400 RANDOMIZE
410 REM DEF CHAR
420 CALL CHAR(96,"00000000FFFFFFFFFFFFFF00
    00000000000000FFFFFFFFFFFFFF")
430 CALL CHAR(120,"0030181C3F1F0700")
440 CALL CHAR(121,"000000")
450 CALL CHAR(122,"00000000FCFF8000")
460 CALL CHAR(123,"00000000")
470 CALL CHAR(104,"0000000071F151F")
480 CALL CHAR(105,"0203020303020 3")
490 CALL CHAR(106,"00008080E0F8A8F8")
500 CALL CHAR(107,"C040C0C040C0C0C0")
510 CALL CHAR(124,"0030181C3F1F07050000")
520 CALL CHAR(126,"00000000FCFF88840000")
530 CALL CHAR(125,"00000000")
540 CALL CHAR(127,"00000000")
550 CALL CHAR(128,"000000000021F3B00")
560 CALL CHAR(129,"00000000E56E300")
570 CALL CHAR(130,"00000000")
580 CALL CHAR(131,"00000000")
590 REM DRAW DISPLAY
600 CALL SPRITE(#1,96,2,180,1,0,B):: CALL CO
    LOR(#1,16)
610 CALL SPRITE(#2,120,2,10,245,A,0):: CALL
    COLOR(#2,7)
620 CALL SPRITE(#3,104,2,110,250,0,-2)
630 CALL MAGNIFY(4)
640 FOR C5=1 TO 40 :: CALL LOCATE(#2,10,C5):
    : NEXT C5 :: GOSUB 870
650 REM MAIN LOOP
660 GOSUB 1120 :: GOSUB 890
670 IF J=0 THEN 690
680 CALL MOTION(#1,0,B,#2,A,0)
690 CALL COINC(#2,170,40,9,T)
700 CALL COINC(#3,110,1,4,DA)
710 CALL COINC(#2,240,40,9,E):: IF E=-1 THEN
     A=1 :: GOSUB 890 :: GOTO 680
720 IF DA=-1 THEN 1320
730 IF T<>-1 THEN 660
740 CALL MOTION(#2,0,0)
750 IF A>1 THEN GOSUB 920 :: GOSUB 960 :: GO
    TO 1660
760 IF LG=0 THEN 1660
770 GOTO 1760
```

```
780 REM UPDATE DISPLAY
790 IMAGE SINK RATE: ###
800 IMAGE RUNWAY ENDS ### YDS
810 IMAGE AIRSPEED: ###
820 IMAGE TOUCH DOWN
830 IMAGE SINK RATE TOO HIGH
840 IMAGE AIRSPEED TOO HIGH
850 IMAGE CRASH LANDING
860 IMAGE STALL WARNING!
870 DISPLAY AT(1,10)SIZE(20):USING "ATTEMPT
    NO. #":A1
880 RETURN
890 DISPLAY AT(3,10)SIZE(20):USING 790:A
900 DISPLAY AT(5,10)SIZE(20):USING 810:-B
910 RETURN
920 DISPLAY AT(7,5)SIZE(20):USING 830
930 RETURN
940 DISPLAY AT(7,5)SIZE(20)BEEP:USING 840
950 DISPLAY AT(9,5)SIZE(20):USING "BOUNCE" :
    : RETURN
960 DISPLAY AT(9,5)SIZE(20):USING 850
970 RETURN
980 CALL HCHAR(7,5,33,27):: DISPLAY AT(9,5)S
    IZE(20):USING 820
990 RETURN
1000 DISPLAY AT(9,5)SIZE(20):USING "WARNING
     "
1010 DISPLAY AT(11,5)SIZE(20):USING 800:RE
1020 RETURN
1030 CALL HCHAR(7,5,33,27):: RETURN
1040 CALL HCHAR(9,5,33,27):: RETURN
1050 CALL HCHAR(11,5,33,27):: RETURN
1060 DISPLAY AT(9,5)SIZE(20):USING "LIFT OFF
     " :: CALL HCHAR(11,5,33,27):: RETURN
1070 DISPLAY AT(3,10):USING "END OF RUNWAY "
      :: DISPLAY AT(5,10):USING "NEW APPROAC
     H" :: DISPLAY AT(7,10):USING "NECESSARY
     "
1080 RETURN
1090 PRINT "THAT'S 5 PASSES AT THE" :: PRINT
      :: PRINT "RUNWAY. TURN IN YOUR" :: PRI
     NT :: PRINT "PILOT LICENSE AND PUT": :
1100 PRINT "SOMEONE ELSE IN THE" :: PRINT ::
      PRINT "COCKPIT" :: PRINT :: RETURN
1110 DISPLAY AT(7,9)BEEP SIZE(20):USING 860
     :: RETURN
1120 REM JOYST/ LANDING GEAR
1130 CALL KEY(1,RV,ST):: IF RV=18 AND LG=0 T
     HEN 1190
```

```
1140 CALL JOYST(1,X,Y):: IF X=Ø AND Y=Ø THEN
      GOSUB 121Ø :: RETURN
1150 A=A-Y/4 :: B=B+X/4
1160 IF ABS(A)>127 THEN A=127*SGN(A)
1170 IF B>-5Ø THEN 143Ø
1180 J=1 :: RETURN
1190 CALL PATTERN(#2,124)
1200 A=A+3 :: B=B+2Ø :: LG=1 :: GOTO 116Ø
1210 REM COMPLICATIONS
1220 CP=INT(RND*16)
1230 IF CP=1 THEN B=B-1 :: GOTO 13ØØ
1240 IF CP=6 THEN B=B+1 :: GOTO 13ØØ
1250 IF CP=1Ø THEN A=A-1 :: GOTO 128Ø
1260 IF CP=15 THEN A=A+1 :: GOTO 128Ø
1270 J=Ø :: RETURN
1280 IF ABS(A)>127 THEN A=127*SGN(A)
1290 GOTO 131Ø
1300 IF B<-127 THEN B=-127
1310 J=1 :: RETURN
1320 REM NEW APPROACH
1330 CALL MOTION(#2,Ø,Ø):: CALL POSITION(#2,
     R4,C4)
1340 IF A1>4 THEN 14ØØ
1350 CALL DELSPRITE(#1,#3):: CALL CLEAR
1360 GOSUB 1Ø7Ø
1370 CALL PATTERN(#2,12Ø)
1380 FOR X=C4 TO 255 :: CALL LOCATE(#2,INT(R
     4),X):: R4=R4-(R4/(255-C4)):: NEXT X
1390 A1=A1+1 :: GOTO 34Ø
1400 CALL DELSPRITE(ALL):: CALL CLEAR
1410 GOSUB 1Ø9Ø
1420 FOR DELAY=1 TO 9ØØ :: NEXT DELAY :: GOT
     O 197Ø
1430 REM STALL
1440 GOSUB 111Ø
1450 CALL MOTION(#2,Ø,Ø)
1460 CALL POSITION(#2,SR,SC)
1470 CALL LOCATE(#2,SR,SC)
1480 CALL COINC(#2,17Ø,4Ø,2,T)
1490 CALL COINC(#3,11Ø,1,2,DE):: IF DE=-1 TH
     EN A1=A1+1 :: GOSUB 87Ø :: IF A1>4 THEN
      14ØØ
1500 IF T=-1 THEN 166Ø
1510 SR=SR+4
1520 CALL KEY(1,RV,ST)
1530 IF RV=18 AND LG=1 THEN 161Ø
1540 CALL JOYST(1,X,Y):: IF X=Ø AND Y=Ø THEN
      147Ø
1550 B=B+X/4
1560 REM
```

```
1570 IF B<-60 THEN 1640
1580 CALL MOTION(#1,0,B)
1590 GOSUB 890
1600 GOTO 1470
1610 CALL PATTERN(#2,120)
1620 A=A-3 :: B=B-22 :: LG=0
1630 GOTO 1560
1640 GOSUB 1030
1650 RETURN
1660 REM CRASH
1670 CALL MOTION(#1,0,0,#2,0,0,#3,0,0,#4,0,0
     )
1680 CALL SOUND(1000,-7,0)
1690 FOR P=1 TO 10
1700 CALL SCREEN(2)
1710 CALL SCREEN(16):: NEXT P :: CALL SCREEN
     (2)
1720 CALL PATTERN(#2,128)
1730 FOR DELAY=1 TO 400 :: NEXT DELAY
1740 CALL DELSPRITE(ALL)
1750 GOTO 1970
1760 REM TOUCHDOWN/BRAKE/T&G
1770 GOSUB 980 :: IF B<-53 THEN 1940
1780 CALL JOYST(1,X,Y):: B=B+X/2
1790 IF B>-1 THEN 1880
1800 CALL MOTION(#1,0,B)
1810 CALL COINC(#3,110,1,4,DA)
1820 IF DA=-1 THEN RE=0 :: GOSUB 1010 :: GOT
     O 1660
1830 CALL DISTANCE(#3,110,1,RQ)
1840 RE=INT(SQR(RQ)):: GOSUB 1000 :: GOSUB 9
     00
1850 CALL KEY(1,RV,ST):: IF RV=18 AND B<-60
     THEN GOSUB 1060 :: A=A-2 :: GOTO 1870
1860 GOTO 1780
1870 CALL MOTION(#2,A,0):: FOR DELAY=1 TO 20
     0 :: NEXT DELAY :: GOTO 650
1880 REM SCORING
1890 CALL MOTION(#1,0,0,#2,0,0,#3,0,0,#4,0,0
     )
1895 FOR DELAY=1 TO 800 :: NEXT DELAY
1900 CALL DELSPRITE(ALL):: CALL CLEAR
1910 PRINT "CONGRATULATIONS !": :
1920 PRINT "YOUR SCORE IS :";(RE/A1)*10: :
1930 GOTO 1990
1940 A=A-2 :: CALL MOTION(#2,A,0):: GOSUB 94
     0
1950 FOR DELAY=1 TO 20 :: NEXT DELAY
1960 A=A+2 :: GOSUB 1030 :: GOSUB 1040 :: GO
     TO 650
```

```
1970 REM PLAY AGAIN
1980 CALL CLEAR
1990 PRINT "PLAY AGAIN (Y/N)?"
2000 CALL KEY(2,RV,SV)
2010 IF SV=0 THEN 2000
2020 IF RV=15 THEN 2050
2030 IF RV=18 THEN 330
2040 GOTO 1990
2050 END
```

7 Utilities

7
TI Disk Deleter

Patrick Parrish

Now you can catalog and delete files on your TI-99/4A disks from BASIC. And you can print the catalog. Runs in Console or Extended BASIC.

Although the TI-99/4A has a DELETE command in its BASIC, its disk operating system (DOS) lacks a cataloging command. To overcome this limitation, TI provides the Disk Manager Command Module with its disk systems. Both delete and catalog options are available with this ROM cartridge.

Unfortunately, using this cartridge is not particularly convenient. First, you must shut down your system, insert the cartridge, and then power the system back up again. Even then, a number of keystrokes may still be required to execute the delete and catalog options. For instance, if you're unsure of the names of the files you wish to delete, you must sequence through all the files on your disk from within the delete option. Alternately, you can run the catalog option, carefully record the names of files for deletion, and then return to the delete option. Either way, this is a slow and laborious process.

If you happen to be programming in Extended BASIC, this is an additional annoyance. Replacement of the Extended BASIC Module with the Disk Manager Command Module is not only a time-consuming interruption, but it also puts a lot of wear and tear on the motherboard cartridge connection. Eventually, you may even begin to experience shorting problems at this interface.

It's possible to delete a file and catalog the disk entirely from BASIC. First, you can delete a file with DELETE "DSK1.FILENAME". Then, you can catalog the disk with a BASIC program provided in the TI *Disk Memory System* manual.

But this approach is also somewhat tedious. Again, if you are unsure of the names of the files you wish to delete, you must first run the cataloging program and carefully record each filename.

Of course, you can combine these two methods. For instance, you can DELETE from BASIC and then catalog the disk with the Disk Manager cartridge or vice versa. But again, there is little, if any, advantage in this.

An Easier Way

Structured much like TI's BASIC catalog program, "TI Disk Deleter" combines the delete and catalog functions in a single program that runs in Console or Extended BASIC.

When run, the program immediately prompts you for the number of the drive you wish to access. If you have only a single drive, this drive is usually referenced as Drive 1. Enter the appropriate number and the drive will begin to whir as the directory is read.

Once the directory has been read, the disk name, the amount of disk space used (in sectors where 1 sector = 256 bytes), the amount of free disk space (also in sectors), and the page number (filenames may occupy as many as four screens) are printed at the top of the screen. Then, a series of filenames are printed in a two-column format. Protected files, or files which cannot be erased or written over, will appear with an asterisk before their names.

Next, a menu with several handy options is given at the bottom of the screen. The six options in this menu—Advance, Back, Kill, Print, Catalog, and Quit—are called by typing their first letters.

The Options

At this time, a pointer (an arrow-shaped character) will be positioned next to the first filename on your screen. This pointer is used to indicate which file will be purged when you execute the Kill function. Move this pointer to any other filename with the arrow keys (E, S, D, and X). When the pointer is next to the file you wish to delete, press K.

After you press K, you'll be asked "Are you sure ?" Press Y (for yes) to delete the file. The filename will disappear from the screen once the file has been deleted. Press N (for no) to abort the deletion and return to the menu. If the file is protected, you cannot delete it without first changing its status to unprotected with the Disk Manager Command Module.

The Advance and Back options are used to move forward and backward through pages of filenames. If you happen to be on the last page of filenames and press A for Advance, nothing will happen. Likewise, if you are on the first page of filenames and press B for Back, nothing happens.

The last three options are very straightforward. Print sends a list of the remaining files on the disk (original catalog minus deleted files) to the printer—you'll have to adjust line 390 to suit your printer. The filename, the size of each file (in sectors), the file type (see below), and its status (protected files are indicated with a P) are given. The Catalog option catalogs any disk in the drive. So, you can clean up all your disks at one time without rerunning the program. The last option, Quit, simply ends the program.

File Types and Program Description

Up to 127 filenames and information on each file are read in from disk in line 700. Filenames are read in as A$(I). Each file type is represented as E(I). The five file types are defined in lines 100–150 as X$(I).

The first four file types are used to store data in records. Data in these files is stored either in binary (INTernal) or ASCII format (DISplay). Also, each record in these files is either FIXed or VARiable in length.

If the value of E(I) is negative, the file is protected. (Only with the Disk Manager Command Module can the protect status be removed.) Next, the length of each file (in sectors) is read as F(I). And finally, G(I) is the record length of files used for data storage.

TI Disk Deleter Program Structure

Lines	
100–190	DIMension and initialize variables
210–240	Subroutine to PRINT at any screen position
250–380	Subroutine to INPUT and PRINT general disk information
390–470	OPEN printer file, define character and set color codes
490–550	Clear out prior filenames
680–730	Routine to INPUT directory information
840–1180	PRINT each page of filenames
1190–1710	Main loop
1210–1450	Pointer movement
1460–1590	Scroll screen
1630–1650	Routine to catalog
1670–1690	Quit program

1720–2040 Routine to DELETE file
2050–2240 Printer routine

Disk Deleter

```
100 DIM A$(127),E(127),F(127),G(127),H$(127)
    ,PAGE(4)
110 X$(1)="DIS/FIX"
120 X$(2)="DIS/VAR"
130 X$(3)="INT/FIX"
140 X$(4)="INT/VAR"
150 X$(5)="PROGRAM"
160 PAGE(1)=1
170 PAGE(2)=37
180 PAGE(3)=73
190 PAGE(4)=109
200 GOTO 390
210 FOR T=1 TO LEN(R$)
220 CALL HCHAR(PROW,PCOL+T,ASC(SEG$(R$,T,1))
    )
230 NEXT T
240 RETURN
250 OPEN #1:"DSK"&STR$(M)&".",INPUT ,RELATIV
    E,INTERNAL
260 INPUT #1:B$,C,C,A
270 IF D=0 THEN 380
280 PROW=1
290 R$=STR$(C-A)
300 CALL HCHAR(PROW,21,32,3)
310 PCOL=23-LEN(R$)
320 GOSUB 210
330 R$=STR$(A)
340 CALL HCHAR(PROW,28,32,3)
350 PCOL=30-LEN(R$)
360 GOSUB 210
370 D=0
380 RETURN
385 REM CHANGE THE PARAMETERS IN LINE 390 TO
     SUIT YOUR PRINTER (SEE YOUR MANUAL)
390 OPEN #2:"RS232.BA=9600.PA=N.DA=8"
400 CALL CHAR(128,"080C0EFFFF0E0C08")
410 CALL CHAR(136,"")
420 CALL COLOR(14,1,1)
430 CALL CLEAR
440 CALL SCREEN(9)
450 FOR I=9 TO 12
460 CALL COLOR(I,2,1)
470 NEXT I
480 IF FL=0 THEN 560
```

```
490 PRINT "...CLEARING OLD FILENAMES"
500 FOR I=1 TO 127
510 A$(I)=""
520 NEXT I
530 FL=0
540 CALL CLEAR
550 GOTO 570
560 PRINT TAB(6);"TI DISK DELETER": : : : :
    : : : : : : :
570 HI=2
580 PRINT "  DRIVE NUMBER (1-3(,) ? ";
590 CALL KEY(0,K,S)
600 IF (S=0)+((K<49)+(K>51))THEN 590
610 M=K-48
620 CALL CLEAR
630 CALL SCREEN(15)
640 FOR I=9 TO 12
650 CALL COLOR(I,16,1)
660 NEXT I
670 GOSUB 250
680 PRINT TAB(3);"...READING DIRECTORY"
690 FOR I=1 TO 127
700 INPUT #1:A$(I),E(I),F(I),G(I)
710 IF LEN(A$(I))<>0 THEN 730
720 I=127
730 NEXT I
740 SC=1
750 LAST=20
760 I=1
770 ROW=3
780 COL=3
790 CALL CLEAR
800 CALL SCREEN((SC+1)*2+1)
810 PRINT "DSK:";B$;TAB(16);"U:";TAB(21-LEN(
    STR$(C-A));C-A;TAB(23);"F:";TAB(28-LEN(
    STR$(A)));A
820 PRINT TAB(21);"PAGE #";SC;
830 IF A5=1 THEN 850
840 IF (LEN(A$(I))=0)+((I=37)+(I=73)+(I=109)
    )THEN 950
850 A5=0
860 PRINT TAB(1);CHR$(136);
870 IF E(I)>=0 THEN 890
880 PRINT "*";
890 PRINT TAB(3);A$(I);TAB(15);CHR$(136);
900 IF E(I+1)>=0 THEN 920
910 PRINT "*";
920 PRINT TAB(17);A$(I+1)
930 I=I+2
940 GOTO 840
```

```
950 HI=INT((I-2)/36+1)
960 ON HI GOTO 970,990,1010,1030
970 DIFF=37-I
980 GOTO 1040
990 DIFF=73-I
1000 GOTO 1040
1010 DIFF=109-I
1020 GOTO 1040
1030 DIFF=145-I
1040 HI=HI+1
1050 U=INT(DIFF/2)
1060 LAST=20-U
1070 FOR Q=1 TO U
1080 PRINT
1090 NEXT Q
1100 PRINT
1110 PRINT "Advance{5 SPACES}Back{7 SPACES}K
     ill"
1120 PRINT "Print{6 SPACES}Catalog
     {5 SPACES}Quit"
1130 PRINT " (USE ARROW KEYS TO MOVE)";
1140 ODD=0
1150 IF LEN(A$(I-1))<>0 THEN 1180
1160 CALL HCHAR(2+(I-((SC-1)*36)-1)/2,17,32)
1170 ODD=1
1180 CALL HCHAR(ROW,COL,128)
1190 CALL KEY(0,K,S)
1200 IF S=0 THEN 1190
1210 IF K<>69 THEN 1290
1220 OLDROW=ROW
1230 ROW=ROW-1
1240 CALL GCHAR(ROW,COL,Q)
1250 IF Q=136 THEN 1270
1260 ROW=LAST-(ODD=1)*(COL=17)
1270 CALL HCHAR(OLDROW,COL,136)
1280 GOTO 1700
1290 IF (K<>68)*(K<>83)THEN 1380
1300 OLDCOL=COL
1310 COL=20-COL
1320 CALL GCHAR(ROW,COL,Q)
1330 IF Q=136 THEN 1360
1340 COL=20-COL
1350 GOTO 1190
1360 CALL HCHAR(ROW,OLDCOL,136)
1370 GOTO 1700
1380 IF K<>88 THEN 1460
1390 OLDROW=ROW
1400 ROW=ROW+1
1410 CALL GCHAR(ROW,COL,Q)
1420 IF Q=136 THEN 1440
```

```
1430 ROW=3
1440 CALL HCHAR(OLDROW,COL,136)
1450 GOTO 1700
1460 IF K<>65 THEN 1520
1470 A5=1
1480 SC=SC+1
1490 IF (SC<=HI)*(LEN(A$(I))<>0)THEN 770
1500 SC=SC-1
1510 GOTO 1190
1520 IF K<>66 THEN 1600
1530 A5=1
1540 SC=SC-1
1550 IF SC=0 THEN 1580
1560 I=PAGE(SC)
1570 GOTO 770
1580 SC=1
1590 GOTO 1190
1600 IF K=75 THEN 1720
1610 IF K=80 THEN 2050
1620 IF K<>67 THEN 1660
1630 CLOSE #1
1640 FL=1
1650 GOTO 430
1660 IF K<>81 THEN 1190
1670 CLOSE #1
1680 CLOSE #2
1690 STOP
1700 CALL HCHAR(ROW,COL,128)
1710 GOTO 1190
1720 J=(SC-1)*36+((ROW-2)*2-1)-(COL=17)
1730 IF E(J)<=0 THEN 1190
1740 C$=""
1750 FOR T=2 TO 11
1760 CALL GCHAR(ROW,COL+T,Z)
1770 IF (Z<>32)+(T<>2)THEN 1810
1780 T=11
1790 FL=1
1800 GOTO 1850
1810 IF Z<>32 THEN 1840
1820 T=11
1830 GOTO 1850
1840 C$=C$&CHR$(Z)
1850 NEXT T
1860 IF FL=0 THEN 1890
1870 FL=0
1880 GOTO 1190
1890 PROW=21
1900 PCOL=5
1910 R$="ARE YOU SURE (Y/N)?"
1920 GOSUB 210
```

```
1930 CALL KEY(0,K,S)
1940 IF S=0 THEN 1930
1950 IF (K<>78)*(K<>89)THEN 1930
1960 CALL HCHAR(21,5,32,20)
1970 IF K<>89 THEN 1190
1980 D=1
1990 DELETE "DSK"&STR$(M)&"."&C$
2000 CLOSE #1
2010 GOSUB 250
2020 CALL HCHAR(ROW,COL+2,32,10)
2030 A$(J)=" "
2040 GOTO 1190
2050 PRINT #2:"DSK";STR$(M);TAB(8);"DISKNAME
      : ";B$:"FREE= ";A;"{8 SPACES}USED= ";C
     -A
2060 PRINT #2:" FILENAME  SIZE{4 SPACES}TYPE
     {4 SPACES}ST":"---------- ---- --------
     -- --"
2070 FOR J=1 TO 127
2080 IF A$(J)=" " THEN 2200
2090 IF LEN(A$(J))<>0 THEN 2120
2100 J=127
2110 GOTO 2200
2120 PRINT #2:A$(J);TAB(12);F(J);TAB(19);X$(
     ABS(E(J)));
2130 IF ABS(E(J))=5 THEN 2160
2140 W$="  "&STR$(G(J))
2150 PRINT #2:SEG$(W$,LEN(W$)-2,3);
2160 IF E(J)<0 THEN 2190
2170 PRINT #2
2180 GOTO 2200
2190 PRINT #2:TAB(28);"P"
2200 NEXT J
2210 FOR J=1 TO 5
2220 PRINT #2
2230 NEXT J
2240 GOTO 1190
```

Master Disk Directory

Raymond J. Herold

This menu-driven utility lets you update, list, search, delete, sort, and print a directory of the files on your disks. It also displays the number and length of files, and the remaining free sectors per disk. Extended BASIC, disk drive, and 32K memory expansion are required.

"Master Disk Directory," for the TI-99/4A, requires the following system configuration: Peripheral Expansion box, Extended BASIC command module, disk controller card, at least one disk drive, and 32K memory board. For those who have this system, this program provides an easy way to keep track of the various disks and the programs and files you have stored on them. Anyone who has a library of 20 or more disks and a hundred or more programs knows the headache involved in trying to keep track of where a particular program is.

Master Disk Directory maintains a catalog of all your disks and the programs and files stored on them. This program lets you display a list of all your disks, showing how many files and how much free space is available on each. You can also display a list of all your programs and files, indicating how large they are and identifying the disk on which they reside. You can search the directory by disk number or program name. You can also sort the directory in program name sequence, list the directory on a printer, delete the entries for a particular disk, and update and save your directory, which will hold data for up to 50 disks and 450 programs and files.

Main Menu Options

When the program is first run, the main menu listing all available functions is displayed. Figure 1 shows the format of this menu. Simply type in the appropriate number for the option you choose. The program provides prompts for easier use.

Figure 1. Disk Directory Menu

```
 1—LOAD CURRENT DIRECTORY
 2—ADD NEW DISKS TO DIR.
 3—LIST ALL DISKS IN DIR.
 4—LIST ALL FILES IN DIR.
 5—SEARCH DIR. BY DISK #
 6—SEARCH DIR. BY FILENAME
 7—DELETE DISK # FROM DIR.
 8—SORT DIR. BY FILENAME
 9—SAVE NEW/UPDATED DIR.
10—PRINT DIRECTORY SELECTION—>
```

1—Load current directory. This first option allows you to load an existing disk directory into the computer's memory. It assumes that a directory exists with the default filename DSK1.DISKDATA which is created by the SAVE option (9). This operation will overlay any directory currently residing in memory. To insure that a directory in memory is not inadvertently destroyed, you will be asked to verify loading of an exiting directory.

2—Add new disks to directory. This allows you to place information for new disks into the directory file. This option is used when a directory file is initially created, or when new disks are to be added to the directory. The program will prompt you to indicate which drive is to be used to load the disks. Once this is established, you will be instructed to insert a disk into the assigned drive. The program will then display the name of that disk and ask you to enter its number (1–50). If you enter 00 the program will return to the menu.

You must number your disks consecutively. You must have cataloged disks 1 to 5 before you number a disk 6. If you are adding to an existing directory, it is best to follow this procedure: From the menu, load the current directory (option 1), list all disks in directory (option 3) to find out how many disks you have already cataloged, use this information to determine the next available disk number, then go to the menu and select option 2.

3—List all disks in directory. This option displays a list of all disks currently in the directory. The display includes the disk number and name, number of files on the disk, and the number of available sectors on the disk. The format of this display is shown in Figure 2.

Figure 2: Disks on File

# NAME	FILES	SECT FREE
1 WORKDISK	7	256
2 RHSOFTWARE	11	102
3 EDITASSMWK	5	252
4 ASSMDEBUG	7	94
5 ASSMGAMES2	3	301
6 CJFMASTER1	8	158
7 E/A	11	1
8 E/A*PARTB	9	5

PRESS ENTER TO CONTINUE

4—List all files in directory. This displays a list of all files (data files and programs) in the directory. For each file, the display provides the filename, file type, file size in sectors, and the disk number on which the file resides. If a particular filename exists on more than one disk, it will be listed the appropriate number of times. This can be helpful in reducing redundancy and, consequently, increasing available storage space. It is helpful if the filenames have first been sorted alphabetically. Figure 3 shows a typical screen display for this option.

Figure 3: Files in Directory

NAME	TYPE	SIZE	DISK
ARTICLES	PROGRAM	30	11
ASSM1	PROGRAM	33	3
ASSM1	PROGRAM	33	7
ASSM2	PROGRAM	20	3
ASSM2	PROGRAM	20	7
BACHMUSIC	PROGRAM	31	2
BARCHARTS	PROGRAM	31	6
BARRICADE	PROGRAM	30	10

PRESS ENTER TO CONTINUE

5—Search directory by disk #. This allows you to search the directory file by disk number. It will generate a display similar to Figure 3. However, the list will contain only those files on the indicated disk number. This is useful for determining which files are on a particular disk.

6—Search directory by filename. This allows you to search the directory file for a particular filename. It will display the disk number and name on which the requested file resides. This is useful when you want to locate a particular file or program, but don't remember which disk it is on. The search routine will handle a generic argument. For example, a search argument of ASSM will display the location of ASSMA, ASSMB, ASSMSORT, and so on. This way you can find the location of a program even if you don't remember its exact name.

7—Delete disk # from directory. This option allows you to delete all data for a particular disk from the directory. The program will display the disk name and ask if you are sure you want to delete it. If you respond with Y, all information for that disk is erased. The filenames deleted will be displayed on the screen.

This option has two main purposes. First, it can be used to delete information for a disk which has been erased or destroyed. Second, this option in conjunction with the add option (2) can be used to easily update the directory periodically. For example, if disk 5 has had files added, changed, or deleted since the last directory update, you would do the following: delete disk 5; invoke the add option (2) and put disk 5 into the appropriate drive; invoke the sort routine (8). The directory would then be updated to reflect any changes to disk 5. Of course, you can update more than one disk at a time this way. You would only need to invoke the sort routine once at the end. Since deletion creates "holes" in the directory array, an array compression routine is automatically invoked after the delete function is complete.

8—Sort directory by filename. This option alphabetically sorts the directory file by filename. The routine involves a BASIC sort and is therefore the slowest function in the program. Just so you don't think the machine has bogged down, the routine will continuously display the number of sort passes remaining. In my own tests, the program took 13 minutes to sort 220 records.

9—Save new/updated directory. This will save the newly created or updated directory file on a disk. The directory will be saved with the default filename: DSK1.DISKDATA. The file may then be referenced or updated at a later time.

10—Print directory. This provides a hard copy list of the directory. The print routine is set up to use a parallel printer.

If you are using a serial printer or have different parameters than mine, you will have to change the OPEN statement in line 10015.

Master Disk Directory

```
10 DIM D$(50),F$(400),TF$(5)
20 TF$(1)="DIS/FIX" :: TF$(2)="DIS/VAR" :: T
   F$(3)="INT/FIX" :: TF$(4)="INT/VAR" :: TF
   $(5)="PROGRAM"
30 D=0 :: F=0
100 CALL CLEAR :: CALL SCREEN(6)
110 CALL CHAR(96,"FFFFFFFFFFFFFFFF")
112 CALL CHAR(112,"E0E0FFFFFFFFFFFF")
113 CALL CHAR(120,"00FFFFFFFFFFFF00")
115 CALL CHAR(104,"01071F3F3F7F7FFFFF7F7F3F3
    F1F070180E0F8FCFCFEFEFFFFFEFEFCFCF8E080"
    )
120 CALL COLOR(9,2,2)
130 FOR L=10 TO 18
135 CALL HCHAR(L,12,96,9)
140 NEXT L
150 CALL SPRITE(#1,104,6,100,115):: CALL MAG
    NIFY(3)
160 CALL HCHAR(10.13.112,1)
170 DISPLAY AT(3,11):"D I S K" :: DISPLAY AT
    (5,6):"D I R E C T O R Y"
190 DISPLAY AT(22,3):"PRESS ANY KEY TO BEGIN
    "
200 CALL KEY(3,K,ST):: C=C+1 :: IF C=20 THEN
     DISPLAY AT(24,1):RPT$(" ",28)
202 IF C=40 THEN C=0 :: GOTO 190
205 IF ST=0 THEN 200
210 CALL DELSPRITE(#1)
500 CALL CLEAR :: CALL SCREEN(14)
510 DISPLAY AT(1,2):"** DISK DIRECTORY MENU
    **"
520 DISPLAY AT(4,1):"1 - LOAD CURRENT DIRECT
    ORY"
530 DISPLAY AT(6,1):"2 - ADD NEW DISKS TO DI
    R."
540 DISPLAY AT(8,1):"3 - LIST ALL DISKS IN D
    IR."
550 DISPLAY AT(10,1):"4 - LIST ALL FILES IN
    DIR."
560 DISPLAY AT(12,1):"5 - SEARCH DIR. BY DIS
    K #"
570 DISPLAY AT(14,1):"6 - SEARCH DIR. BY FIL
    E NAME"
```

```
575 DISPLAY AT(16,1):"7 - DELETE DISK # FROM
     DIR."
580 DISPLAY AT(18,1):"8 - SORT DIR. BY FILE
    NAME"
590 DISPLAY AT(20,1):"9 - SAVE NEW/UPDATED D
    IR."
600 DISPLAY AT(22,1):"10- PRINT DIRECTORY"
610 DISPLAY AT(24,4):"SELECTION---->" :: ACC
    EPT AT(24,19)SIZE(2)VALIDATE(NUMERIC)BEE
    P:S
615 IF S>10 OR S<1 THEN 610
620 ON S GOTO 1000,2000,3000,4000,5000,6000,
    7000,8000,9000,10000
999 GOTO 500
1000 CALL CLEAR :: CALL SCREEN(8)
1010 DISPLAY AT(4,1):"THIS OPTION WILL LOAD
     THE"
1020 DISPLAY AT(6,1):"DIRECTORY FILE FROM DI
     SK."
1030 DISPLAY AT(8,1):"IT WILL OVERLAY ANY FI
     LE"
1040 DISPLAY AT(10,1):"CURRENTLY IN MEMORY."
1050 DISPLAY AT(14,1):"LOAD DIRECTORY FILE (
     Y/N)? ." :: ACCEPT AT(14,28)VALIDATE("Y
     N")SIZE(-1)BEEP:O$
1060 IF O$="N" THEN 500
1100 OPEN #2:"DSK1.DISKDATA",INPUT ,INTERNAL
     ,FIXED 20
1105 INPUT #2:ND,NF
1110 FOR L=1 TO ND
1120 INPUT #2:D$(L)
1130 NEXT L
1140 FOR L=1 TO NF
1150 INPUT #2:F$(L)
1160 NEXT L
1180 DISPLAY AT(20,1):"DIRECTORY FILE LOADED
     " :: DISPLAY AT(22,1):"PRESS ANY KEY FO
     R MENU"
1185 CLOSE #2
1190 CALL KEY(3,K,S):: IF S=0 THEN 1190
1199 GOTO 500
2000 CALL CLEAR :: CALL SCREEN(8)
2005 DISPLAY AT(4,1):"DISKS WILL BE PLACED I
     N" :: DISPLAY AT(6,1):"DRIVE 1, 2 OR 3?
     " :: ACCEPT AT(6,18)VALIDATE("123")SIZE
     (1)BEEP:N :: CALL CLEAR
2010 DISPLAY AT(4,1):"INSERT DISK AND PRESS
     ENTER"
2015 FOR L=1 TO 100 :: NEXT L
```

```
2020 CALL KEY(3,K,S):: IF K<>13 THEN 2020
2030 OPEN #1:"DSK"&STR$(N)&".",INPUT ,RELATI
     VE,INTERNAL
2040 INPUT #1:A$,I,J,A
2050 DISPLAY AT(10,1):"DISKNAME: ";A$
2060 DISPLAY AT(12,1):"ENTER DISK NUMBER: __
     " :: ACCEPT AT(12,20)VALIDATE(NUMERIC)S
     IZE(-2)BEEP:DN
2062 IF DN=00 THEN CLOSE #1 :: GOTO 500
2065 IF DN>50 THEN 2060
2070 IF D$(DN)="" THEN 2100
2080 DISPLAY AT(16,1):"DISK NUMBER ALREADY U
     SED" :: DISPLAY AT(18,1):"PRESS R TO RE
     TRY" :: DISPLAY AT(19,1):"PRESS M FOR M
     ENU"
2085 CALL KEY(3,K,S):: IF S=0 THEN 2085
2090 IF K=82 THEN CALL CLEAR :: GOTO 2050
2095 IF K=77 THEN CLOSE #1 :: GOTO 500
2097 GOTO 2085
2100 DN$=A$&(RPT$(" ",10-LEN(A$))):: AV$=RPT
     $(" ",3-LEN(STR$(A)))&STR$(A)
2105 C=0
2110 FOR L=1 TO 127
2120 INPUT #1:A$,I,J,K
2130 IF LEN(A$)=0 THEN 2200
2140 C=C+1 :: NF=NF+1
2150 N$=A$&RPT$(" ",10-LEN(A$)):: S$=RPT$("
     ",3-LEN(STR$(J)))&STR$(J):: T$=STR$(ABS
     (I)):: DD$=RPT$(" ",2-LEN(STR$(DN)))&ST
     R$(DN)
2160 F$(NF)=N$&S$&T$&DD$
2190 NEXT L
2200 NF$=RPT$(" ",3-LEN(STR$(C)))&STR$(C)
2202 D$(DN)=DN$&AV$&NF$
2205 CLOSE #1 :: ND=ND+1
2210 DISPLAY AT(16,1):"DISK CATALOGED" :: DI
     SPLAY AT(18,1):"PRESS A TO ADD ANOTHER
     DISK" :: DISPLAY AT(19,1):"PRESS M FOR
     MENU"
2220 CALL KEY(3,K,S):: IF S=0 THEN 2220
2230 IF K=65 THEN CALL CLEAR :: GOTO 2010
2240 IF K=77 THEN 500
2250 GOTO 2220
3000 CALL CLEAR :: CALL SCREEN(8)
3005 X=0
3010 GOSUB 3100
3020 FOR L=1 TO 50
3022 IF D$(L)="" THEN 3040
3025 GOSUB 3300
3030 DISPLAY AT(X*2+7,1):USING "## #########
     # ###{7 SPACES}###":L,DN$,NF$,AV$
```

```
3035 X=X+1 :: IF X=8 THEN GOSUB 3200
3040 NEXT L
3050 DISPLAY AT(24,1):"PRESS ENTER FOR MENU"
3060 CALL KEY(3,K,S):: IF K<>13 THEN 3060
3070 GOTO 500
3100 DISPLAY AT(2,4):"** DISKS ON FILE **"
3110 DISPLAY AT(4,24):"SECT"
3120 DISPLAY AT(5,1):" # NAME{7 SPACES}FILES
     {4 SPACES}FREE"
3125 CALL HCHAR(6,3,45,28)
3130 RETURN
3200 X=0
3210 DISPLAY AT(24,1):"PRESS ENTER TO CONTIN
     UE"
3220 CALL KEY(3,K,S):: IF K<>13 THEN 3220
3230 CALL CLEAR :: GOSUB 3100
3240 RETURN
3300 DN$=SEG$(D$(L),1,10)
3310 AV$=SEG$(D$(L),11,3)
3320 NF$=SEG$(D$(L),14,3)
3330 RETURN
4000 CALL CLEAR :: CALL SCREEN(15)
4005 X=0
4010 GOSUB 4100
4020 FOR L=1 TO NF
4022 IF F$(L)="" THEN 4040
4025 GOSUB 4300
4030 DISPLAY AT(X*2+7,1):USING "########## #
     ###### ###{3 SPACES}##":N$,TF$(ABS(VAL(
     T$))),S$,DD$
4035 X=X+1 :: IF X=8 THEN GOSUB 4200
4040 NEXT L
4050 DISPLAY AT(24,1):"PRESS ENTER FOR MENU"
4060 CALL KEY(3,K,S):: IF K<>13 THEN 4060
4070 GOTO 500
4100 DISPLAY AT(2,3):"** FILES IN DIRECTORY
     **"
4110 DISPLAY AT(4,1):"NAME{9 SPACES}TYPE  SI
     ZE DISK"
4115 CALL HCHAR(5,3,45,28)
4120 RETURN
4200 X=0
4210 DISPLAY AT(24,1):"PRESS ENTER TO CONTIN
     UE"
4220 CALL KEY(3,K,S):: IF K<>13 THEN 4220
4230 CALL CLEAR :: GOSUB 4100
4240 RETURN
4300 N$=SEG$(F$(L),1,10)
4310 S$=SEG$(F$(L),11,3)
4320 T$=SEG$(F$(L),14,1)
```

```
4330 DD$=SEG$(F$(L),15,2)
4340 RETURN
5000 GOSUB 5500
5003 CALL CLEAR :: CALL SCREEN(8)
5005 X=0
5010 GOSUB 5100
5015 IF D$(DN)="" THEN DISPLAY AT(12,1):"**
     NO SUCH DISK # IN DIR. **" :: GOTO 5050
5020 FOR L=1 TO NF
5022 IF F$(L)="" THEN 5040
5023 DW$=RPT$(" ",2-LEN(STR$(DN)))&STR$(DN)
5024 IF DW$<>SEG$(F$(L),15,2)THEN 5040
5025 GOSUB 4300
5030 DISPLAY AT(X*2+7,1):USING "########## #
     ###### ###{3 SPACES}##":N$,TF$(ABS(VAL(
     T$))),S$,DD$
5035 X=X+1 :: IF X=8 THEN GOSUB 5200
5040 NEXT L
5050 DISPLAY AT(24,1):"PRESS ENTER FOR MENU"
5060 CALL KEY(3,K,S):: IF K<>13 THEN 5060
5070 GOTO 500
5100 DISPLAY AT(2,3):"** FILES ON DISK #":DN
     ;"**"
5110 DISPLAY AT(4,1):"NAME{9 SPACES}TYPE  SI
     ZE DISK"
5115 CALL HCHAR(6,3,45,28)
5120 RETURN
5200 X=0
5210 DISPLAY AT(24,1):"PRESS ENTER TO CONTIN
     UE"
5220 CALL KEY(3,K,S):: IF K<>13 THEN 5220
5230 CALL CLEAR :: GOSUB 5100
5240 RETURN
5500 CALL CLEAR :: CALL SCREEN(6)
5510 DISPLAY AT(4,1):"SEARCH DIRECTORY BY DI
     SK #"
5520 DISPLAY AT(8,1):"ENTER DISK # __" :: AC
     CEPT AT(8,14)VALIDATE(NUMERIC)SIZE(-
2)BEEP:DN
5530 IF DN<01 OR DN>50 THEN 5520
5550 RETURN
6000 GOSUB 6500
6003 CALL CLEAR :: CALL SCREEN(8)
6005 X=0 :: SW=0
6010 GOSUB 6100
6020 FOR L=1 TO NF
6022 IF F$(L)="" THEN 6040
6024 IF SEG$(PW$,1,LEN(PW$))<>SEG$(F$(L),1,L
     EN(PW$))THEN 6040
```

```
6030 DISPLAY AT(X*2+7,1):USING "##
     {3 SPACES}##########  ##########":SEG$(
     F$(L),15,2),SEG$(D$(VAL(SEG$(F$(L),15,2
     ))),1,10),SEG$(F$(L),1,10)
6035 X=X+1 :: SW=1 :: IF X=8 THEN GOSUB 6200
6040 NEXT L
6045 IF SW=0 THEN DISPLAY AT(12,4):"** NO MA
     TCH FOUND **"
6050 DISPLAY AT(24,1):"PRESS ENTER FOR MENU"
6060 CALL KEY(3,K,S):: IF K<>13 THEN 6060
6070 GOTO 500
6100 DISPLAY AT(2,6):"SEARCH FOR ":PW$
6110 DISPLAY AT(5,1):"DISK  DISKNAME
     {3 SPACES}FILENAME"
6115 CALL HCHAR(6,3,45,28)
6120 RETURN
6200 X=0
6210 DISPLAY AT(24,1):"PRESS ENTER TO CONTIN
     UE"
6220 CALL KEY(3,K,S):: IF K<>13 THEN 6220
6230 CALL CLEAR :: GOSUB 6100
6240 RETURN
6500 CALL CLEAR :: CALL SCREEN(6)
6510 DISPLAY AT(4,1):"SEARCH DIRECTORY BY FI
     LENAME"
6520 DISPLAY AT(8,1):"ENTER FILE NAME " :: A
     CCEPT AT(8,17)SIZE(10)BEEP:PW$
6550 RETURN
7000 CALL CLEAR :: CALL SCREEN(10)
7010 DISPLAY AT(4,1):"THIS OPTION WILL DELET
     E ALL" :: DISPLAY AT(6,1):"FILES FOR A
     SPECIFIED DISK"
7020 DISPLAY AT(8,1):"NUMBER." :: DISPLAY AT
     (10,1):"DELETE DISK? Y/N _" :: O$="" ::
      ACCEPT AT(10,18)VALIDATE("YN")SIZE(-1)
     BEEP:O$
7030 IF O$="N" THEN 500
7050 DISPLAY AT(12,1):"DISK # TO BE DELETED?
      __" :: DEL=0 :: ACCEPT AT(12,23)VALIDA
     TE(NUMERIC)SIZE(-2)BEEP:DEL
7055 IF DEL=00 THEN 500
7057 IF DEL>50 THEN 7050
7059 IF D$(DEL)<>"" THEN 7100
7060 DISPLAY AT(16,1):"** NO SUCH DISK # IN
     DIR." :: DISPLAY AT(18,1):"R TO RETRY -
      M FOR MENU"
7070 CALL KEY(3,K,S):: IF S=0 THEN 7070
7080 IF K=82 THEN CALL HCHAR(16,1,32,32):: C
     ALL HCHAR(18,1,32,32):: GOTO 7050
```

```
7085 IF K=77 THEN 7200
7090 GOTO 7070
7100 CALL CLEAR :: DISPLAY AT(2,1):"DISK TO
     DELETE = "&SEG$(D$(DEL),1,10):: DISPLAY
      AT(4,1):"DELETE? Y/N _"
7110 O$="" :: ACCEPT AT(4,13)VALIDATE("YN")S
     IZE(-2)BEEP:O$ :: IF O$="N" THEN 7200
7115 IF O$<>"Y" THEN 7100
7120 CALL CLEAR :: DISPLAY AT(1,4):"** FILES
      DELETED **"
7130 L2=0 :: D$(DEL)="" :: DELSTR$=RPT$(" ",
     2-LEN(STR$(DEL)))&STR$(DEL):: XX=NF ::
     DC=0
7140 FOR L=1 TO XX
7150 IF DELSTR$=SEG$(F$(L),15,2)THEN GOSUB 7
     300 :: DISPLAY AT(3+INT(DC/2),PC):SEG$(
     F$(L),1,10):: F$(L)="" :: DC=DC+1 :: D
     S=1
7160 NEXT L
7170 DISPLAY AT(24,1):"PRESS ANY KEY TO CONT
     INUE"
7175 CALL KEY(3,K,S):: IF S=0 THEN 7175
7180 CALL CLEAR :: DISPLAY AT(4,1):"DELETE A
     NOTHER DISK? Y/N _" :: O$="" :: ACCEPT
     AT(4,26)VALIDATE("YN")SIZE(-1)BEEP:O$
7190 IF O$="Y" THEN 7050
7200 IF DS=0 THEN 500
7210 CALL CLEAR :: CALL SCREEN(4):: DISPLAY
     AT(8,4):"AUTOMATIC COMPRESSION" :: DISP
     LAY AT(10,6):"ROUTINE ACTIVATED"
7215 DISPLAY AT(14,1):"----> PLEASE STAND BY
      <----"
7220 L2=0 :: XX=0
7230 FOR L=1 TO NF
7240 IF F$(L)="" THEN 7260
7250 L2=L2+1 :: F$(L2)=F$(L):: XX=XX+1
7260 NEXT L
7262 FOR L=NF+1 TO 400
7264 F$(L)=""
7266 NEXT L
7270 NF=XX
7290 GOTO 500
7300 IF DC/2=INT(DC/2)THEN PC=1 ELSE PC=15
7310 RETURN
8000 CALL CLEAR :: CALL SCREEN(6)
8010 DISPLAY AT(10,5):"SORTING...."
8020 F$(0)="___" :: Y=1 :: HX=0
8025 SS=0 :: DISPLAY AT(10,17):NF-Y
8030 FOR L=Y TO NF
```

```
8040 IF F$(L)<F$(Ø)THEN F$(Ø)=F$(L):: HX=L :
     : SS=1
8050 NEXT L
8060 IF SS=1 THEN HF$=F$(Y):: F$(Y)=F$(HX)::
      F$(HX)=HF$
8070 Y=Y+1 :: F$(Ø)=F$(Y)
8080 IF Y<NF THEN 8025
8090 GOTO 5ØØ
9000 CALL CLEAR :: CALL SCREEN(8)
9010 DISPLAY AT(4,1):"THIS OPTION WILL WRITE
      THE"
9020 DISPLAY AT(6,1):"DIRECTORY FILE TO DISK
     .  IT"
9030 DISPLAY AT(8,1):"WILL OVERLAY ANY PREVI
     OUS"
9040 DISPLAY AT(1Ø,1):"DIRECTORY FILE."
9050 DISPLAY AT(14,1):"WRITE FILE (Y/N)?  ."
      :: ACCEPT AT(14,2Ø)VALIDATE("YN")SIZE(
     -1)BEEP:O$
9060 IF O$="N" THEN 5ØØ
9100 OPEN #2:"DSK1.DISKDATA",OUTPUT,INTERNAL
     ,FIXED 2Ø
9105 PRINT #2:ND,NF
9110 FOR L=1 TO ND
9120 PRINT #2:D$(L)
9130 NEXT L
9140 FOR L=1 TO NF
9150 PRINT #2:F$(L)
9160 NEXT L
9180 DISPLAY AT(2Ø,1):"UPDATE COMPLETE" :: D
     ISPLAY AT(22,1):"PRESS ANY KEY FOR MENU
     "
9185 CLOSE #2
9190 CALL KEY(3,K,S):: IF S=Ø THEN 919Ø
9199 GOTO 5ØØ
10000 CALL CLEAR :: CALL SCREEN(6)
10010 DISPLAY AT(8,1):"PRINTING....."
10015 OPEN #3:"PIO.LF",OUTPUT
10020 GOSUB 12ØØØ
10030 FOR L=1 TO NF
10040 GOSUB 43ØØ
10050 PRINT #3,USING "{6 SPACES}#########
      {5 SPACES}###{5 SPACES}#######
      {4 SPACES}##":N$,S$,TF$(VAL(T$)),DD$
10060 LC=LC+1 :: IF LC=58 THEN GOSUB 11ØØØ
10070 NEXT L
10075 CLOSE #3
10090 GOTO 5ØØ
11000 FOR X=LC TO 65 :: PRINT #3:" " :: NEXT
       X
```

```
11010 GOSUB 12000
11020 RETURN
12000 PRINT #3:"{16 SPACES}DIRECTORY INDEX"
12010 PRINT #3:" " :: PRINT #3:" " :: PRINT
      #3:" "
12020 PRINT #3:"{6 SPACES}FILENAME
      {7 SPACES}SIZE{5 SPACES}TYPE
      {5 SPACES}DISK"
12030 PRINT #3:" "
12040 LC=6
12050 RETURN
```

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.

Braces

The exception to this typing rule is when you see the braces, such as {10 SPACES}. This special situation occurs in PRINT statements. For example,

ENERGY{10 SPACES}MANAGEMENT

means that ten spaces should be left between the words ENERGY and MANAGEMENT. *Do not* type in the braces or the words 10 SPACES.

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

A TI Anthology

COMPUTE!'s TI Collection, Volume 1 contains a wide variety of articles, all selected because of their excellence and high quality. But unlike most anthologies, this book also includes dozens of programs that you can type in and run on your TI-99/4A computer. Games, applications, utilities, and tutorials show you how to make your TI a powerful game machine, as well as a versatile tool for the home, for education, and even for business.

COMPUTE!'s TI Collection, Volume 1 continues the tradition of presenting the best programs and articles from COMPUTE! Publications in a clear and easy-to-understand style. Just as with the best-selling *Programmer's Reference Guide to the TI-99/4A* and *COMPUTE!'s First Book of TI Games,* this anthology gives you a wealth of information you can immediately use. And like all COMPUTE! books, the programs have been thoroughly tested and are ready to run.

Most of the 30 programs in this book have never before been published. Included are:

- "SuperFont," an easy-to-use character editor.
- Articles on how to add sprites to your own programs, plus a sprite editor that makes sprite creation fun and easy.
- Seven games—some strategic, some arcade action—including the popular "Worm of Bemer."
- "The Mozart Machine," a program that produces computer-generated music.
- A complete data base management system.
- An electronic spreadsheet for your TI-99/4A.
- A word processor you can use to write reports, memos, letters, or even stories.
- A way to transfer variables from one program to another.
- Useful applications, such as a program to keep track of your disk files and another to record your mailing lists.

Whether you're looking for programming applications, useful home applications, or just some entertaining games, you'll find *COMPUTE!'s TI Collection, Volume 1* full of high quality programs that are easy to use and enjoy.

ISBN 0-942386-71-X