For the Commodore 128 in 64 mode

COMPUTE!'s

THIRD BOOK OF COMMODORE 64

The best games, applications, utilities, and BASIC tutorials from COMPUTE! Publications. Solve a murder mystery, create an 80-column display, perform disk surgery, paint in hi-res, and improve programming style.

A **COMPUTE! Books** Publication

$12.95

COMPUTE!'s
THIRD BOOK OF COMMODORE 64

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

The following article was originally published in *COMPUTE!* magazine, copyright 1983, COMPUTE! Publications, Inc.:
"Machine Language Saver" (June).

The following articles were originally published in *COMPUTE!* magazine, copyright 1984, COMPUTE! Publications, Inc.:
"64 Hi-Res Graphics Editor" (May); "Programming 64 Sound" (June and July—originally titled "Programming 64 Sound, Part 1" and "Programming 64 Sound, Part 2"); "64 Paintbox" (December).

The following articles were originally published in *COMPUTE!'s Gazette* magazine, copyright 1983, COMPUTE! Publications, Inc.:
"Word Match" (October); "Connect The Dots" (November).

The following articles were originally published in *COMPUTE!'s Gazette* magazine, copyright 1984, COMPUTE! Publications, Inc.:
"Making More Readable Listings" (March); "Variable Storage: A Beginner's Tour of BASIC RAM" (April—originally titled "Variable Storage: A Beginner's Tour of BASIC RAM for VIC and 64"); "Sound Sculptor" (May—originally titled "Sound Sculptor for the 64"); "One-Touch Keywords" (June—originally titled "Power BASIC: One-Touch Keywords"); "Word Scramble" (June); "Mystery at Marple Manor" (September); "Disk Surgeon" (September—originally titled "Disk Tricks"); "Screen-80: 80 Columns for the 64" (September—originally titled "80 Columns for the 64"); "Screen Headliner" (September); "Autoload" (November—originally titled "Disk Auto Load"); "Supertank" (November); "Turtle Graphics Interpreter" (October)

The following article was originally published in *COMPUTE!* magazine, copyright 1983, Jim Butterfield:
"BASIC Style: Program Evolution" (May 1984)

Printed in the United States of America

ISBN 0-942386-72-8

10 9 8 7 6 5

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. Commodore 64 is a trademark of Commodore Electronics Limited.

Contents

Foreword

In the two and a half years since the Commodore 64 was first introduced, it's become the home computer of millions of people. And its popularity shows no sign of decreasing. COMPUTE! Publications has supported the 64 from the time of its introduction at the Summer 1982 Consumer Electronics Show, extending that tradition with a wide variety of books dedicated to the 64.

COMPUTE!'s Third Book of Commodore 64 is now part of that tradition. With the same high-quality programs and concise writing that people have come to expect from COMPUTE!, this book follows in the path of the best-selling *First* and *Second Book of Commodore 64*. Filled with articles and programs from *COMPUTE!* magazine and *COMPUTE!'s Gazette*, many enhanced or extensively revised, as well as several never before published, this book presents the best programs from a strong group. It wasn't always an easy choice; there are always more to choose from than can fit in one book.

You'll find a variety of programs and articles here. Some, like "Screen-80" and "HiSprite," are sophisticated programs that allow you to display 80 columns on your monitor or control all aspects of sprites. Others, such as "BASIC Style: Program Evolution" and "Programming 64 Sound" are tutorials that show you how to write cleaner programs or how to get the most out of your 64's SID chip. Graphics and sound applications let you paint on a high-resolution screen, turn your 64 into an Atari-like graphics computer, and even manipulate sound parameters. Utilities enlarge your programmer's toolkit with routines like "Crunch," "Machine Language Saver," "Autoload," and "One-Touch Keywords." And "Programming Without the Keyboard," COMPUTE!'s first 64 programming utility designed for the physically handicapped, allows joystick-controlled BASIC programming.

Games, always a strength of the Commodore 64, are not forgotten. "Mystery at Marple Manor" puts you in a house filled with potential suspects, possible murder weapons, and a trail that can lead to only one conclusion. "Supertank" sets you in a modern armored battle, and "Reversi" lets you demonstrate your strategic planning and execution skills.

There are even programs that insure error-free typing of both BASIC and machine language programs.

From new owners to experienced programmers, everyone who has a Commodore 64 will find that *COMPUTE!'s Third Book of Commodore 64* contains valuable information, tested programs, and clear explanations.

1
BASIC Programming

BASIC Style
Program Evolution

Jim Butterfield

Sometimes you see programs that are so crisp and neat that you wonder how the programmer's mind can be so orderly. The statements come out in an elegant, incisive style. Every line does exactly the right thing. But can you learn to program like that?

How does a programmer develop an elegant style? Why can't *you* write like that? Sometimes you can feel inferior after looking at such immaculate programming style. Yet the program you see is often a matter of evolution—rewriting and tidying up. Just as a story or a novel isn't usually published after only one draft, a computer program may go through an entire series of revisions.

I've been accused of writing "squeaky clean" programs. That's not the way they start. Like most other programmers' work, my murky first attempts get reworked and tightened up into their final version. In fact, programming style often isn't what you write (at least at first)—it's knowing what to look for when you clean up. Since showing is better than just describing, how about taking a look at the evolution of one of my programs?

A Simple Lister

I needed to do an almost trivial job: list a sequential file from disk to the printer. I had a minor extra feature to add: I wanted individual pages, so that the lines needed to be counted; I needed a title on each page; and at the end of the run, for the sake of neatness, I wanted the printer to eject the page.

It wasn't a demanding task, but I'd like to show you how I went about it. Even a simple job like that can be revised and tightened up extensively.

(Note: If you want to use this program yourself, remember that it's only for listing *sequential* files, not program files.)

Here's my first program: I'll talk my way through the listing.

```
100 OPEN 4,3
```

Open file number four to the screen. Why? So I can send the program's output to the screen and see that it's working right. After the program looks good, I'll change the above line to OPEN 4,4

```
105 OPEN 1,8,3,"CONTROL"
```

CONTROL is my input file to be listed.

```
110 REM START OF PAGE
120 FOR J=1 TO 2:PRINT#4:L=L+1:NEXT J
130 PRINT#4,"{5 SPACES}TITLE{3 SPACES}":L=L+1
140 PRINT#4:L=L+1
```

This prints the page title. I know the program will come back here for each new page, so I'm placing a REM statement to mark the place. I make sure that the program adds 1 to the line count, L, each time a line is printed.

```
150 INPUT#1,A$:SW=ST
170 PRINT#4,A$:L=L+1
```

Here's where I input from disk and output (to the screen first, later to the printer). I have the program save the value of ST (the status variable) so that later it can check to see if this is the last line from the file. ST will be changed by the PRINT# command, so its input value is saved in variable SW.

```
180 IF L<62 GOTO 250
190 IF L=66 THEN L=0:GOTO 250
200 PRINT#4:L=L+1:GOTO 190
```

If the program has printed the maximum number of lines desired, I want it to eject the paper by printing until the line count, L, equals 66. Since each page has 66 lines, if L is greater than that, the next page has started and L can be set back to zero.

```
250 IF SW<>0 GOTO 300
260 IF L=0 GOTO 110
270 GOTO 150
```

If the program finds the end of the input file (SW<>0), it will go to line 300 and wind things up. Otherwise, I want it to go back.

Lines 260 and 270 contain a cute touch—perhaps too cute for some tastes. Variable L can be equal to zero only if a page has just been ejected. If so, I want the routine to go back to 110 and print a new title. If not, I want it to get another line from the input file starting at line 150.

```
300 IF L<>0 GOTO 190
```

Here's a *supercute* trick. I pondered using this for a while, since it's almost too clever; that sort of thing can trip up your logic. Here's what I was thinking of: If the program's finished, but the paper *hasn't* been ejected, go back to line 190 and eject the paper. The program will branch back here again, but then L will be zero and everything can be wrapped up by closing the files with the next two lines.

```
310 CLOSE 1
320 CLOSE 4
```

That's it. It's really rather messy. It works, and for a temporary job that's all we would need.

But it doesn't feel right. The code feels sloppy; it seems to jump around, and I don't get a feeling of smoothness in the program. If that feeling comes to you, you're telling yourself it's time to pick at the program. I listened to that instinct and began revising.

First Revision

The first awkward spot is around lines 190 and 200. The routine to eject the paper works but looks clumsy. Besides, it's called twice (once when the paper's at 62 lines, and again at the end of the file).

I have other ideas about this part of the program, too. It's a unit to do a particular job. I think it would be better to move it to a separate subroutine where it can stand out as an identifiable action. Sometimes I even create a subroutine out of some lines in the middle of the program and then move it back later; it helps me identify the modules that make up the program. Let's move the paper ejecting routine to a subroutine at line 500, clean up the program a bit, and see what we get. It might look something like this:

```
100 OPEN 4,3
105 OPEN 1,8,3,"CONTROL"
110 REM START OF PAGE
120 FOR J=1 TO 2:PRINT#4:L=L+1:NEXT J
130 PRINT#4,"{5 SPACES}TITLE{3 SPACES}":L=L+1
```

```
140 PRINT#4:L=L+1
150 INPUT#1,A$:SW=ST
170 PRINT#4,A$:L=L+1
180 IF L<62 GOTO 250
190 GOSUB 500:GOTO 250
250 IF SW<>0 GOTO 300
260 IF L=0 GOTO 110
270 GOTO 150
300 IF L<>0 GOTO 190
310 CLOSE 1
320 CLOSE 4
330 END
500 FOR J=L TO 66:PRINT#4:NEXT J
510 L=0:RETURN
```

You can see that the GOTO 250 in line 190 is redundant since the program will go there anyway. But we have other things to do. We're still trimming the program and have a ways to go yet.

Digging Deeper

Around lines 250 to 270, the program jumps around a lot. It has one jump forward to 300 and two jumps back to 110 and 150. The logic seems scattered.

I have a thing about loops: I like to see them neatly nested, with short jumps entirely within longer jumps. It might even be summarized as a rule of thumb: Where possible, make short jumps as short as possible.

Using this rule, I want to get the loop which returns to 150 into logical order. That's first. Then I'll work on the longer loop to 110. Finally, I'll fix the forward branch to 300. We'll need to expand the logic using an AND operator, but that's not too hard.

As the routine is written, certain logical things start to fall together. For example, we don't have to GOTO forward to line 300. When we're finished writing the two loops, the program will fall into 300 naturally. (*Naturally* seems to be a key word in how programs seem to come together as you tighten them up.)

We can also tighten up the page-eject conditions. If we write line 180 correctly, there'll be no need to go back to get a page ejection. One option would be to call the subroutine at 500 twice. But if we think of what our objective really is at line 180, we can do it all correctly the first time through. Inverting the logic and adding an OR connective does the trick nicely.

Look at how far the original program has come:

```
100 OPEN 4,4
105 OPEN 1,8,3,"CONTROL"
110 REM START OF PAGE
120 FOR J=1 TO 2:PRINT#4:L=L+1:NEXT J
130 PRINT#4,"{5 SPACES}TITLE{3 SPACES}":L=L+1
140 PRINT#4:L=L+1
150 INPUT#1,A$:SW=ST
170 PRINT#4,A$:L=L+1
180 IF L>61 OR SW<>0 THEN GOSUB 500
250 IF SW=0 AND L>0 GOTO 150
260 IF SW=0 GOTO 110
310 CLOSE 1
320 CLOSE 4
330 END
500 FOR J=L TO 66:PRINT#4:NEXT J
510 L=0:RETURN
```

This is pleasing, but we can do even more. The repeated SW=0 test in lines 250 and 260 still irks me a little: It seems clumsy. The whole business is tied up in whether a title should be printed. Is there a better way? Could the test of L>0 be somehow shuttled to the top of the loop instead of sitting at the bottom?

The Header Module

While we're thinking about it, that whole business of printing a header is really a module—we must do the whole thing, title and all, or nothing. If we move it out to a subroutine, we might see the logic flow more clearly. Let's do it and work on the logic flow. We'd end up with this:

```
100 OPEN 4,3
105 OPEN 1,8,3,"CONTROL"
110 IF L=0 THEN GOSUB 600
150 INPUT#1,A$:SW=ST
170 PRINT#4,A$:L=L+1
180 IF L>61 OR SW<>0 THEN GOSUB 500
260 IF SW=0 GOTO 110
310 CLOSE 1
320 CLOSE 4
330 END
500 FOR J=L TO 66:PRINT#4:NEXT J
510 L=0:RETURN
600 FOR J=1 TO 2:PRINT#4:L=L+1:NEXT J
610 PRINT#4,"{5 SPACES}TITLE{3 SPACES}":L=L+1
620 PRINT#4:L=L+1
630 RETURN
```

Look at that main section from lines 100 to 330. It now seems tight and concise, like a finely constructed poem. That's not a bad simile, for just as every word should count for something in a poem, so should every line in a program work towards the final result.

Both subroutines—at lines 500 and 600—are called only once. If it seemed important, we could put them back into the main program stream. But I'm happy to see them as clearly isolated modules. At this stage I would add comments (for instance, REM PAGE EJECT at line 499 and REM PAGE TITLE at line 599) to make the program even neater.

Moral

First, what you see published is not always the first idea that popped into the author's head. The programmer is not always smarter than you. Time and thought have been taken to groom the program into its final shape. When many people are going to read your code, you like to take a few extra pains with its appearance.

Second, don't be afraid to revise your programs, even if they work correctly. Sure, a one-shot program might not warrant picking over; use it and forget it. But sometimes, the exercise can reveal, almost accidentally, powerful and effective programming methods.

Third, *style* isn't an inborn talent that some people have and others don't. You learn it as you go. Some things you'll discover for yourself, and others you'll pick up by looking at other people's programs.

The odd thing is that we instinctively recognize better writing when we have written it. It's the same with programming. You may not know exactly why, but you often feel good about a certain program. Usually, it's because it has style.

Variable Storage
A Beginner's Tour of BASIC RAM

Pete Marikle

You can simplify the search for program bugs if you take a short tour through BASIC RAM and use this subroutine that displays variable values.

Normally, you don't need to know what happens to your program when you type RUN. The BASIC interpreter takes over, leaving you free to use the computer to figure your income tax, write a letter, or save the galaxy.

When your program crashes, though, or gives you an incorrect result, you have to switch hats. You're not just a computer user then; you have to be a programmer who can locate the bug and fix it. Debugging is easier if you can look at the values of your variables and arrays while the program is running, to insure that loops are being completed and data is put in the right place at the right time.

Programs 1 and 2, listed at the end of this article, are expanded and condensed versions of a subroutine that displays the current values of all program variables. By inserting STOP statements in any line where you suspect a problem, you can freeze the action and GOTO the subroutine to check your logic, statement by statement.

A Quick Tour of RAM

Before we examine the subroutine, let's take a short sightseeing tour through BASIC RAM to see where your Commodore 64 stores programs and variables, how it tells a string from an integer variable, and how you might use less memory. You don't have to take this descriptive tour to use the subroutine, but it *will* give you a better idea of how the subroutine works.

First, type in this short BASIC program. It lets you peek into the computer's memory.

```
10 S=256:PRINT"{CLR}START ADDRESS":INPUTZ
20 S$="**********************":T$="---------------
   -------"
30 FORX=ZTO(PEEK(55)+S*PEEK(56)):PRINTCHR$(144)X,P
   EEK(X)SPC(2)CHR$(PEEK(X))
31 Y=X+1
35 U=PEEK(45)+S*PEEK(46):V=PEEK(47)+S*PEEK(48):W=P
   EEK(49)+S*PEEK(50)
40 IFY=UORY=VORY=WTHEN PRINTS$
45 IFX>=UANDY<VTHENT=T+1:GOTO47
46 T=0
47 IFT THEN IFT/7-INT(T/7)<.01THENPRINTT$
50 WAIT 197,32:NEXT
60 REM END OF PROGRAM APPROACHING
```

(If you want to use this program again, you should save it to tape or disk.)

Now enter these two lines in direct mode (without using line numbers):

AB=12.34:CD=−12.34:AB$="HELLO":AB%=1983:AB(1)=1 11:CD(1)=−111:AB%(1)=1024

AB$(1)="BYE"

Hit RETURN after each line, and enter some more:

DIMCD$(3,5,5):CD$(1,0,0)="SEE":CD$(2,0,0)="YOU":CD$(1, 1,1)="LATER"

Hit RETURN again, and your computer will have at least one example of every type of variable stored in RAM. Now type GOTO 10 and press RETURN. Do *not* type RUN (it resets all variables). You'll see a prompt at the top of the screen; respond with 2250.

The Program Looks at Itself

The space bar is your one-touch control. Every time you press it, a line of infomation appears in black on the screen. Hold it down until the screen is nearly full, then sit back and take a look. You're looking at the middle of the tour program; the memory addresses are on the left, memory contents in the middle, and some interesting characters on the right.

Some of those characters are meaningless, because a CHR$ interpretation of the contents of a memory location is invalid and out of context if the location contains a keyword, line link, line number, and so on. But many of the characters are valid, recognizable translations of what you put into the program. These are the ones we'll look at.

Use the space bar to move through another hundred or so bytes, to address 2377. You're looking for the end of the BASIC program, represented by three consecutive zeros in the center column beside addresses 2377–2379. It's not hard to find with the REM billboard (created by line 60) and neat borders in place. Now look at the first address after the three zeros. It should be 2380, the address produced by PEEK(45)+256*PEEK(46). Line 35 in the above routine sets U equal to that address. Hold the space bar down until 2380 is near the top of the screen.

Scanning Variable Storage

You're now in the area where strings and variables are stored. Everything in this area is in seven-byte clusters, which have been neatly separated with dashes for easy viewing. Find the characters A and B, followed by five more bytes (the cluster is in addresses 2380–2386). This first seven-byte cluster is the variable AB. The first two bytes are the variable name. The next five bytes contain the value you gave AB, but in floating-point arithmetic notation. Don't worry about how the math works. The decimal value *is* neatly tucked away in those five bytes.

Note that the next variable, CD (addresses 2387–2393), has a similar structure. Remember that you put the same numbers in CD as you did in AB, but you included a minus sign to make it negative. Take a close look at the five bytes following CD, and you'll see that the values are almost identical to those in the bytes following AB. The only difference is that the fourth byte's value is 128 greater than the corresponding byte in AB. You can check this for yourself by subtracting 128 from the byte in CD; you should get the value in the corresponding byte in AB. The high-order bit (bit 7) in that particular byte is used as a sign indicator: 0 for positive numbers and 1 for negative. Since that bit is on (set at 1) for variable CD, the byte's decimal value is 128 (2^7) higher. Your computer ignores that bit in reconstructing the value of CD, but uses the bit when the time comes to determine the sign of the number.

String Variables

Press the space bar and look at the next cluster, representing the string variable AB$. The A is clear enough, but where did the B go? Here's the secret: The second character of a string variable name is stored after adding 128 to the normal CHR$ value for that character. It's the high-order bit trick again.

By checking to see if this high-order bit is 1 or 0, your computer can tell whether this is a string or floating-point variable. Memory address 2395 has a value of 194—subtract 128 from it and you have 66, which just happens to be the CHR$ value for the letter B in AB$. Your computer now knows that the next byte (which has a value of 5 in the example) is the length of AB$ and the next two bytes give it the address where it can find the actual characters you designated for the string. The address is in standard low byte/high byte order (LB+256*HB=decimal address). The computer will start at that address, select a number of characters equal to the value (5) in the length byte, and then go on to do whatever you asked it to do with the string.

The final two bytes of the cluster both hold 0; they're put in to fill up the seven bytes.

That address for the string character can point to one of two very different areas of memory. If the string is assigned in the direct mode, the string characters themselves are stored at the top of free BASIC RAM. If the string is assigned by the program, the address points to the place in the program where the string values are assigned to the variable name. Since the characters must be stored as part of the program anyway, your computer doesn't waste RAM by repeating the characters in the variable storage area.

An Unreadable Name

Let's continue the tour. In the next cluster, notice that the variable name is unreadable. The symbols are a spade and a vertical bar, displayed in addresses 2401 and 2402 respectively. The values in those two bytes are 193 and 194. Subtract 128 from each and you'll find the CHR$ values for the letters A and B of the integer variable AB%.

When both characters in the variable name are greater than 127, your computer knows this is an integer variable, that only the next two bytes need to be looked at to obtain its value, and that the last three bytes of the cluster will be filled with zeros. As you can see, this cluster is in that format.

Those value bytes contain a signed binary number, a different form than you saw with the floating-point variables. Again, don't worry about the details of the math. The more compact method of storing integer variables doesn't do much for you until you start using them in arrays. Integer arrays can cut your memory consumption considerably (two bytes versus five per entry).

As long as we're talking about arrays, let's look at them in more detail. Hold down the space bar to pass by several clusters where the variables in this tour program are stored. You're approaching the address found by PEEK(47)+256*PEEK(48). That's the beginning of array storage. You'll know you're there when you see the borderline and the A and B characters in the right column. The memory address right beneath the border should be 2478.

How Arrays Are Stored

There are three kinds of arrays, paralleling the three normal variable types: floating-point arrays, integer arrays, and string arrays. Each can be multidimensional, but we'll cover that last. Your 64 allows you to use arrays with up to 11 elements (numbers 0–10) without a DIMension statement, but it does not reserve space for the array until you assign a value to one of the array elements. As soon as you do, it sets up an 11-element array, even if you used only one or two elements. Of course, you can dimension (with a DIM statement) for more or fewer elements if you wish. (For more information on arrays, take a look at "How to Use Arrays" in *COMPUTE!'s Second Book of Commodore 64.*)

Each one-dimensional array begins with a seven-byte definition cluster followed by the 11-element clusters (or more or less according to the DIM statement).

The seven-byte cluster holds the array name in the first two bytes, following the same general rules you saw for simple variables, depending on the type of array. The next two bytes contain a link address to the next array set. The fifth byte tells you (and your computer) the number of dimensions in this array. The sixth and seventh bytes will show the total number of elements in the array set (11 for our unDIMed examples). These two bytes store the total in reverse high byte/low byte order.

The element clusters that follow the definition cluster will be five bytes long for floating-point arrays, two bytes long for integer arrays, or three bytes long for string arrays. These clusters contain the same kind of information held in the corresponding simple variables, but without the trailing zeros or repeated label bytes needed in variable storage.

Unused Elements Contain Zeros

Hold down the space bar until the first array, AB, nearly fills the screen. See the seven-byte cluster in memory addresses 2478–2484? It's followed by five zeros only because AB(0), the first element of this array, has a zero value. The next five bytes represent the value you gave to AB(1). The following sets of zeros represent the remaining unused elements through AB(10). Use the space bar to look at the CD array, then continue to the AB% integer array.

Both begin with a seven-byte definition cluster, followed this time by 11-element clusters of two bytes each. The lesson in saving memory with integer arrays is dramatic.

Next, note the seven-byte cluster for the AB$ array and its 11 three-byte clusters, each containing the string length byte and the address of the string characters.

The Three-Dimensional Array

If you move even further into the tour, you'll reach the sample multidimensional array. Things get a bit tricky here. The definition cluster will now be *more* than seven bytes long. Add two bytes for each extra dimension. Remember, you can set up two, three, four, or more dimensions of any size if you have the memory capacity to handle them. The number of dimensions for each array set is held in the fifth byte (address 2675) of the definition cluster. The very next two bytes hold the number of elements in the nth dimension (n=number of dimensions); the next two contain the number of elements in the (n−1)th dimension, and so on until finally the first dimension is structured. You should see a 0 and a 6 in 2676 and 2677, another pair in 2678 and 2679, and a 0 and a 4 in addresses 2680 and 2681. The 6, 6, and 4 represent, in reverse order, the fact that you dimensioned CD$ as 3, 5, 5 (remember that arrays always start with 0).

Immediately following the definition cluster, the array elements will troop by in orderly formation. In this example, where you DIMed CD$(3,5,5), the order of the three-byte clusters will be: CD$(0,0,0), CD$(1,0,0)...CD$(3,0,0), CD$(0,1,0), CD$(1,1,0)...CD$(3,1,0), and so on until CD$(3,5,5) is reached.

As you pass through this area, you'll see that the clusters for CD$(1,0,0) and CD$(2,0,0) are occupied. If you count, you'll find that the position for CD$(1,1,1) is also occupied, as you directed. As with any string, the characters themselves are stored elsewhere.

If you race through the rest of this array, you'll cross the PEEK(49)+256*PEEK(50) border into the area of unused RAM. Don't be surprised if you recognize some of it. You may find remnants from other programs which have been NEWed, or even CLRed variables.

To end the tour, just hold down the RUN/STOP key and hit the space bar.

The Variable Dump Utility

Now let's try out the promised subroutine. Because it takes all the values stored in a section of memory and sends them to an output device, our subroutine is called a *dump utility.* Type NEW to get rid of the tour program, type in Program 1, and save it to tape or disk. The dump utility has high line numbers because it's designed as an easy add-on to existing programs.

Type in a few sample variables in direct mode. You can enter the samples you used for the tour if you like. Again, do not type RUN; enter GOTO 44444 and press RETURN. Your variables should be displayed; the program won't show the arrays until you press the space bar. Note that the dump utility doesn't list the contents of multidimensional arrays. It's not hard to do, just time-consuming. The routine will simply tell you which multidimensional arrays have been implemented and what their dimensions and element sizes are.

Pointer Settings Affect the Utility

Now CLR your variables, enter this new temporary program step, and run the program again:

```
10 A$="HELLO":A=1983:AB$(2)="HELLO AGAIN"
```

Not much happens, because it ends at line 44443, the subroutine protector. Type GOTO 44444 and hit RETURN to view your variables as before. Now for a surprise—when you type GOTO 44444 and hit RETURN one *more* time, you'll see a display of the variables used in the dump utility.

This happens because, on the first pass through the routine, line 44444 reads the pointers *before* they are changed to make room for the routine's own internal variables. On the second pass, the new pointer values include the storage areas for the new variables. If you don't ever want to see the internal variables, just modify line 44543 to read:

IF PEEK(ZZ)=90 THEN RETURN

Tailor the Utility for Your Needs

You can customize the routine to fit your needs. For example, if you don't need the array and integer variable features, just delete lines 44465, 44525, and everything from 44700 on. That'll leave you with a much trimmer 800-byte package that will still dump all normal string and floating-point variables. If you delete one of the simple variable subroutines, though, you should also delete the corresponding array variable type. Eliminate REMs and spaces and you'll end up with a tidy utility well under 600 bytes that'll still fill most needs. Program 2 is this condensed version.

To use your dump utility as a debugging tool, simply insert STOP statements at desired points in your program, type GOTO 44444, analyze variable values, and then type CONT to continue to the next break. Add the appropriate printer commands, and the program will even dump to the printer.

Program 1. Variable Utility, Expanded Version

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C, with the next two programs.

```
44440 REM{3 SPACES}DUMP...                           :rem 164
44441 REM***START WITH GOTO 44444                    :rem 106
44443 END:REM PROTECT SUBROUTINE                      :rem 41
44444 ZB=PEEK(47)+256*PEEK(48)-7:ZA=PEEK(45)+256*P
      EEK(46)                                        :rem 185
44450 PRINT"STRINGS &{2 SPACES}VARIABLES:":PRINT"*
      *********************"                         :rem 114
44460 FOR ZZ=ZA TO ZB STEP 7                          :rem 39
44465 IF PEEK(ZZ)>127 THEN GOSUB 44710:GOTO44520:R
      EM INT VAR                                     :rem 171
44470 IF PEEK(ZZ+1)<128 THEN GOSUB 44543:GOTO 4452
      0:REM FP VAR                                   :rem 177
```

```
44475 REM*****STRING**************VARIABLE :rem 39
44480 GOSUB44485:GOTO44520                  :rem 255
44485 PRINTCHR$(144)CHR$(PEEK(ZZ))CHR$(PEEK(ZZ+1)-
      128)CHR$(36)CHR$(61);                  :rem 76
44490 ZY=PEEK(ZZ+3)+256*PEEK(ZZ+4):ZX=PEEK(ZZ+2):R
      EM STRINGADDRESS AND LENGTH            :rem 56
44495 IF ZY=0 THEN 44510                    :rem 230
44500 FOR Z0=1TOZX:PRINTCHR$(PEEK(ZY));:ZY=ZY+1:NE
      XTZ0                                  :rem 234
44510 PRINT:RETURN                          :rem 165
44520 NEXTZZ                                :rem 242
44525 GOSUB 44805:REM DO ARRAYS NOW           :rem 0
44530 PRINT:PRINTCHR$(144)"....ALL DONE":END
                                             :rem 75
44540 REM***FLOAT PT************VARIABLE    :rem 187
44543 IFPEEK(ZZ)=90 AND(PEEK(ZZ+1)=65 OR PEEK(ZZ+1
      )=66)THEN RETURN                      :rem 148
44545 PRINTCHR$(144)CHR$(PEEK(ZZ))CHR$(PEEK(ZZ+1))
      CHR$(61);                             :rem 198
44550 Z1=2↑(PEEK(ZZ+2)-129)                 :rem 251
44560 Z2=128:Z3=256:Z4=1                     :rem 62
44570 Z5=PEEK(ZZ+3): IF Z5>=128 THEN Z5=Z5-128:Z4=
      -1                                    :rem 123
44575 J=PEEK(ZZ+4):K=PEEK(ZZ+5):L=PEEK(ZZ+6)
                                            :rem 179
44580 Z9=Z1+Z5*Z1/Z2+J*Z1/Z2/Z3+K*Z1/Z2/Z3↑2+L*Z1/
      Z2/Z3↑3                               :rem 145
44590 PRINTZ9*Z4                            :rem 222
44600 RETURN                                :rem 222
44700 REM***INTEGER***************VARIABLE :rem 43
44710 PRINTCHR$(144)CHR$(PEEK(ZZ)-128)CHR$(PEEK(ZZ
      +1)-128)CHR$(37)CHR$(61);              :rem 12
44720 Z4=1:Z7=PEEK(ZZ+2):Z8=PEEK(ZZ+3)       :rem 29
44730 IF Z7 >127THENZ7=255-Z7:Z8=256-Z8:Z4= -1
                                             :rem 25
44740 Z9=Z7*256+Z8:REMNOTE REVERSE HIBYTE-LOBYTE S
      EQUENCE                               :rem 114
44750 PRINTZ9*Z4                            :rem 220
44760 RETURN                                :rem 229
44800 REM*** ARRAY***************VARIABLES:rem 240
44805 IFZQ=0THENZA=0:GOSUB44550:ZA%=0:GOSUB44720:Z
      R=2:ZQ=2:ZX=2:ZY=2:Z0=2                :rem 84
44806 REM ABOVE DUMMIES NEEDED TO STABILIZE
      {3 SPACES}POINTER TO ARRAYS           :rem 240
44810 ZZ=PEEK(47)+256*PEEK(48):IFZZ=PEEK(49)+256*P
      EEK(50)THEN RETURN                     :rem 32
44815 PRINT"SPACEBAR WHEN READY{3 SPACES}FOR ARRAY
      S":WAIT197,32                          :rem 25
44820 IF PEEK(ZZ+4)<>1THENGOSUB45110:GOTO44820:REM
       MULTI-D ARRAY                        :rem 125
```

```
44825 IF PEEK(ZZ)>127 THEN GOSUB 44900:GOTO44820:R
      EM INT ARRAY                              :rem 69
44828 IF PEEK(ZZ+1)>127 THEN GOSUB 45010:GOTO44820
      :REM STRING ARRAY                         :rem 137
44829 REM*****FLOAT PT ************ARRAY        :rem 82
44830 ZQ=ZZ:ZZ=ZZ+7                             :rem 224
44840 FOR ZR=0 TO PEEK(ZQ+6)+256*PEEK(ZQ+5)-1:REM*
      *DIM                                      :rem 70
44850 PRINTCHR$(144)CHR$(PEEK(ZQ))CHR$(PEEK(ZQ+1))
      CHR$(40)ZRCHR$(41)CHR$(61);               :rem 204
44860 ZZ=ZZ-2:GOSUB44550:ZZ=ZZ+2                :rem 2
44870 ZZ=ZZ+5                                   :rem 12
44880 NEXTZR:IFZZ=PEEK(49)+256*PEEK(50)THEN RETURN
                                                :rem 108
44890 GOTO44820                                 :rem 68
44900 REM****INTEGER************ARRAYS          :rem 101
44910 ZQ=ZZ:ZZ=ZZ+7                             :rem 223
44920 FOR ZR=0 TO PEEK(ZQ+6)+256*PEEK(ZQ+5)-1:REM*
      *DIM                                      :rem 69
44930 PRINTCHR$(144)CHR$(PEEK(ZQ)-128)CHR$(PEEK(ZQ
      +1)-128)CHR$(37)CHR$(40);                 :rem 251
44940 PRINTZRCHR$(41)CHR$(61);                  :rem 233
44950 ZZ=ZZ-2:GOSUB44720:ZZ=ZZ+2                :rem 1
44960 ZZ=ZZ+2                                   :rem 9
44970 NEXTZR:IFZZ=PEEK(49)+256*PEEK(50)THEN GOTO 4
      4530                                      :rem 197
44980 RETURN                                    :rem 233
45000 REM****STRING*************ARRAYS          :rem 80
45010 ZQ=ZZ:ZZ=ZZ+7                             :rem 215
45020 FOR ZR=0 TO PEEK(ZQ+6)+256*PEEK(ZQ+5)-1:REM*
      *DIM                                      :rem 61
45030 PRINTCHR$(144)CHR$(PEEK(ZQ))CHR$(PEEK(ZQ+1)-
      128)CHR$(36)CHR$(40);                     :rem 42
45040 PRINTZRCHR$(41)CHR$(61);                  :rem 225
45050 ZZ=ZZ-2:GOSUB44490:ZZ=ZZ+2                :rem 253
45060 ZZ=ZZ+3                                   :rem 2
45070 NEXTZR:IFZZ=PEEK(49)+256*PEEK(50)THEN GOTO 4
      4530                                      :rem 189
45080 RETURN                                    :rem 225
45100 REM**MULTI-D***************ARRAYS         :rem 160
45110 ZX=2:ZY=2:PRINTCHR$(43)PEEK(ZZ+4)"DIMENSIONA
      LARRAY:":PRINTTAB(5);                     :rem 16
45120 IF PEEK(ZZ)<127THENPRINTCHR$(PEEK(ZZ));:GOTO
      45140                                     :rem 111
45130 PRINTCHR$(PEEK(ZZ)-128);:ZX=1             :rem 99
45140 IFPEEK(ZZ+1)=0THEN45170                   :rem 176
45145 IFPEEK(ZZ+1)=128THEN ZY=1:GOTO45170 :rem 180
45150 IF PEEK(ZZ+1)<127THENPRINTCHR$(PEEK(ZZ+1));:
      GOTO45170                                 :rem 45
45160 PRINTCHR$(PEEK(ZZ+1)-128);:ZY=1           :rem 195
```

```
45170 IF ZX=1 AND ZY=1THENPRINT"%";:GOTO45190
                                             :rem 122
45180 IF ZY=1 THENPRINT"$";                   :rem 17
45190 PRINTCHR$(40);                         :rem 129
45200 Z9=PEEK(ZZ+4)                           :rem 84
45210 FORZ8=Z9TO1STEP-1:Z7=PEEK(ZZ+4+2*Z8)+(PEEK(Z
      Z+4+2*Z8-1))*256-1                     :rem 254
45220 PRINTZ7;                                :rem 86
45230 IFZ8=1THENPRINTCHR$(41):GOTO45250      :rem 115
45240 PRINTCHR$(44);:NEXTZ8                  :rem 140
45250 PRINT                                  :rem 141
45260 ZZ=ZZ+PEEK(ZZ+2)+PEEK(ZZ+3)*256:IF ZZ=PEEK(4
      9)+256*PEEK(50)THEN44530               :rem 107
45270 RETURN                                 :rem 226
```

Program 2. Variable Utility, Condensed Version

```
44443 END:REM MINIDUMP FPVAR & $              :rem 36
44444 ZB=PEEK(47)+256*PEEK(48)-7:ZA=PEEK(45)+256*P
      EEK(46)                                :rem 185
44460 FORZZ=ZATOZBSTEP7                       :rem 39
44470 IFPEEK(ZZ+1)<128THENGOSUB44543:GOTO44520
                                              :rem 20
44480 GOSUB44485:GOTO44520                   :rem 255
44485 PRINTCHR$(144)CHR$(PEEK(ZZ))CHR$(PEEK(ZZ+1)-
      128)CHR$(36)CHR$(61);                   :rem 76
44490 ZY=PEEK(ZZ+3)+256*PEEK(ZZ+4):ZX=PEEK(ZZ+2)
                                             :rem 168
44495 IFZY=0THEN44510                        :rem 230
44500 FOR Z0=1TOZX:PRINTCHR$(PEEK(ZY));:ZY=ZY+1:NE
      XTZ0                                   :rem 234
44510 PRINT:RETURN                           :rem 165
44520 NEXTZZ                                 :rem 242
44530 END                                    :rem 215
44543 IFPEEK(ZZ)=90THENRETURN                :rem 114
44545 PRINTCHR$(144)CHR$(PEEK(ZZ))CHR$(PEEK(ZZ+1))
      CHR$(61);                              :rem 198
44550 Z1=2↑(PEEK(ZZ+2)-129)                  :rem 251
44560 Z2=128:Z3=256:Z4=1                      :rem 62
44570 Z5=PEEK(ZZ+3):IFZ5>=128THENZ5=Z5-128:Z4=-1
                                             :rem 123
44575 J=PEEK(ZZ+4):K=PEEK(ZZ+5):L=PEEK(ZZ+6)
                                             :rem 179
44580 Z9=Z1+Z5*Z1/Z2+J*Z1/Z2/Z3+K*Z1/Z2/Z3↑2+L*Z1/
      Z2/Z3↑3                                :rem 145
44590 PRINTZ9*Z4                             :rem 222
44600 RETURN                                 :rem 222
```

Making More Readable Listings

Brent Dubach

Have you ever tried to find a key subroutine or loop in a long BASIC listing? If you have, you know how tedious it can be. This tutorial demonstrates some very sneaky BASIC editing techniques that you can use for more readable listings.

A few carefully chosen variable names can help make the difference between a readable program and an unintelligible mess. But BASIC does not make these choices easy. Did you ever want to use a BASIC keyword like TO or FN within a variable name, such as LET TOP=10 or PRINT FN$?

Commodore BASIC won't allow it. But by fooling a couple of BASIC routines, you can use these illegal variable names and do even more to improve the appearance of your listings. Let's see how to use this technique and then consider what makes it work.

Illegal Variable Names

The key is to use graphics characters where they normally don't belong. You're probably used to seeing a graphics character as the last character in the abbreviation of a BASIC keyword. For example, if you type a P followed by a SHIFTed O, you'll see the letter P, followed by a graphics character. BASIC, however, understands that you mean POKE. But how will BASIC handle a graphics character in the middle of a variable name?

```
1Ø LET NJUMBER = 5Ø
2Ø PRINT NJUMBER
```

To get the graphics character between N and U, type a SHIFTed J. You can use any graphics character that will not result in an abbreviation of a BASIC keyword. (For example, an N and a SHIFTed E combine to form the keyword NEXT.)

Now list the two-line program, and you should see the following on the screen:

10 LET NUMBER = 50
20 PRINT NUMBER

Now run it, and this appears:

50
READY.

Nothing too impressive. All you have is a program that lists and runs exactly as it would if you had left out the graphics characters. Now let's do something that's downright illegal.

```
1Ø LET TOP = 65
2Ø LET BOTTOM = 9Ø
3Ø PRINT BOTTOM - TOP + 1
```

If you enter and run this program, you'll get a syntax error. The sequence *TO* may not appear anywhere within a variable name as it does here in *TO*P and in BOT*TO*M. It's reserved as a BASIC keyword (as in FOR J=1 TO 5).

Let's try to fool BASIC. You can place a graphics character (the SHIFTed J) just before the character that completes the BASIC keyword—that is, before the *O* in each *TO*.

```
1Ø LET TJOP = 65
2Ø LET BOTTJOM = 9Ø
3Ø PRINT BOTTJOM - TJOP + 1
```

Here's what you see when you list it:

10 LET TOP = 65
20 LET BOTTOM = 90
30 PRINT BOTTOM − TOP + 1

These lines appear identical to the illegal program you entered just a moment ago. Now run the program with the embedded graphics characters. You should see:

26
READY.

It works, with an illegal variable name in every line. Try it with variable names such as LETTER, FN$, EFFORT, SEND, or your own favorite forbidden name.

A word of caution, though. ST, TI, and TI$ are *reserved variable names*, not *keywords* like LET, PRINT, and other BASIC commands or functions. You'll not be able to use variable names whose first two letters match these (like START or

TIME) even with the technique described in this article. Since they are just variable names, however, you may embed them elsewhere within longer names of your own (FIRST and AT-TIC, for example, will work) without any special editing tricks.

Indented Listings and Blank Lines

Besides preventing the selection of certain variable names, BASIC also seems to prevent the entry of blank lines and spaces at the beginning of a line. Thus, it's not possible to neatly frame the blocks of code—loops or IF-THEN options or subroutines—that occur in a program. If you've programmed only in BASIC, you may not be concerned about this. But any-one who has used a computer language like Pascal appreciates being able to see a listing like this:

```
1Ø FOR I = 1 TO 1Ø
2Ø   PRINT "WE INDENT EVERY STATEMENT"
3Ø   PRINT "THAT LIES WITHIN"
4Ø   PRINT "THE FOR-NEXT 'BLOCK'"
5Ø NEXT I
6Ø
7Ø PRINT "AND LEAVE A BLANK LINE BETWEEN BLOCKS"
```

Try entering and listing the program above on your 64. Here's what you should see on the screen:

10 FOR I = 1 TO 10
20 PRINT "WE INDENT EVERY STATEMENT"
30 PRINT "THAT LIES WITHIN"
40 PRINT "THE FOR-NEXT 'BLOCK'"
50 NEXT I
70 PRINT "AND LEAVE A BLANK LINE BETWEEN BLOCKS"

The blank line and all the indentations have disappeared. Of course, Commodore BASIC lets you place a single colon at the start of each line and then indent as much as you wish. But that's not quite the same as a nice, clean blank line.

Once again, you can type an extra graphics character and fool BASIC. When entering a program, many people type a space after the line number for readability. But instead of the space, you can type the SHIFTed J. Reenter the preceding pro-gram this way:

```
1ØJFOR I = 1 TO 1Ø
2ØJ  PRINT "WE INDENT EVERY STATEMENT"
3ØJ  PRINT "THAT LIES WITHIN"
4ØJ  PRINT "THE FOR-NEXT 'BLOCK'"
```

```
50JNEXT I
70JPRINT "AND LEAVE A BLANK LINE BETWEEN BLOCKS"
```

Now when you type LIST, you see an indented format identical to the one you first tried to enter.

Fooling BASIC into giving you a blank line is a little trickier. A single SHIFTed J will not do the job. If you add a line 99, say, to your program and put only the graphics character on that line, line 99 will still not show up in the a listing. But try entering this (note the space between the two SHIFTed Js):

```
99 J J
```

Now type LIST and you'll see a blank line 99.

Paying the Price

There *is* a price to pay for all this. The most obvious is memory consumption. Long variable names and indentation gobble up a lot of bytes. A final version of a routine, though, can be condensed by a good list-crunching program (such as "Crunch," found elsewhere in this book), while the original remains a very readable version for later examination or revision. And with the Commodore 64, most times you don't have to worry about memory limitations.

Another penalty is simply the bother of remembering to type extra characters. Be careful whenever you try to edit a line. To preserve any indentation, you must enter a SHIFTed J in place of the space following the line number *each time you change the line.* And it's easy to forget to convert a variable name this way by inserting a graphics character within an embedded BASIC keyword. If you do forget, you'll be reminded when you get a syntax error in the program. So watch your editing steps carefully.

If you're a hunt-and-peck typist, you might find entering all these extra characters a nuisance. But a little irritation can lead to a lot of satisfaction when you get a more readable program listing.

How Does It Work?

There are BASIC routines that run and list a program. If you've experimented with the short listings here, or with your own, you've already proved that the RUN command apparently doesn't mind using keywords in variable names, and that the LIST command seems to accept leading spaces in indented lines. If these key routines are so tolerant, what is it that requires us to be so sneaky in achieving these results? The answers lie in the behavior of several other parts of BASIC.

Are They Really Illegal?

First, let's consider illegal variables and a BASIC routine we'll call TOKENIZE.

We usually think of BASIC commands as words like INPUT or LET or GOTO. But the RUN routine does not see it that way. By the time RUN sees a program, BASIC keywords have been replaced by single-byte numeric codes, or *tokens.* TOKENIZE is the part of BASIC that translates the keywords you enter into these codes. For example, when you type the word INPUT, TOKENIZE collects the characters in that word from the five bytes of memory they occupy, matches them with a word in the computer's list of BASIC keywords, and then replaces them with the token for INPUT (the number 133), which takes up only one byte. This saves space in BASIC memory.

But TOKENIZE also discards any out-of-place graphics characters as it crunches a BASIC command into the computer's memory. This is what allows us to enter forbidden variable names. When you insert a graphics character (like the SHIFTed J) in the middle of what would otherwise be a keyword, imagine how TOKENIZE must react. Does it ever find the word INPUT? Not quite. As it is collecting characters, it's interrupted before finding a perfect match with the BASIC word INPUT. The match is a failure, but the character which foiled it is eventually discarded. When RUN gets at the program, it now finds a plain INPUT (five bytes worth) instead of the single-byte token that represents the INPUT command. Any such character string is treated as a variable name.

Our illegal variable names, then, are not illegal at all. You just have to be sneaky enough in entering and editing them to prevent TOKENIZE from doing its job.

Finding the Right Routine

And what of the graphics character used at the beginning of an indented line?

TOKENIZE is involved again, this time because it does just what you want done: It keeps spaces right where you put them. Some other parts of BASIC use a routine that discards spaces. One of these is the part that translates the characters in a line number you type to the numeric form in which it is stored. Try leaving a space between two digits in a line number. No problem—the spaces are discarded and the line number appears in a listing just as if you had not inserted them.

BASIC continues to throw away spaces until a nondigit character which eliminates all indented lines is found. The rest of the line is turned over to the TOKENIZE routine. But by then it is too late: All indentations have already been stripped.

Our strategy must be to place a character immediately after the line number so that the following spaces will be handled by the right routine for our purposes—by TOKENIZE. A graphics character, first recognized as a nondigit character in the collection of a line number and then neatly discarded by TOKENIZE, is the perfect choice.

Guarding the Blanks

Finally, you may recall that in order to create a blank line (but which still has a line number), you needed first a graphics character, then a space, then a second graphics character. The reason for the first was just discussed. A space is needed so there will be something on the line for TOKENIZE to accept. Remember that entering a completely blank line just results in its elimination from the program. But what of the second graphics character? If TOKENIZE doesn't mind spaces, why shouldn't it accept a whole line full of them following the initial graphics character?

In the first place, you probably want only one space—just enough to create a blank line. And second, TOKENIZE never gets to look at those trailing spaces anyway. The very first part of BASIC involved in handling a new line, the part that collects characters off the screen, discards these spaces. Both graphics characters are needed to protect lone blanks from the space-killing habits in a couple of parts of BASIC. If you want blank lines with a lot of spaces, though, there is no reason why you couldn't enter one with, say, 70 of them. Just be sure they have graphics "bodyguards" on either end.

2
Recreations and Applications

Mystery at Marple Manor

John R. Prager

You've been summoned to Marple Manor on a dark and stormy night to investigate the unexpected demise of one of the dinner guests. Clues are everywhere, but can you discover who did it, to whom, how, and where? A mystery text-adventure for one to six players.

Searching through the study, you find a duelling pistol hidden under a cushion. Later, you discover the cook cowering in a closet. The greenhouse door is locked, but you have the key. And there, concealed in the potted ferns, is the body of the Duchess.

Your job is to find out "whodunit," and how, before the other detectives crack the case. They're a shifty lot, who might hide vital clues or steal the evidence you've accumulated, just to throw you off the track. There are over 15,000 possible solutions, but only one correct answer. A different mystery is chosen each time the program runs. It's a race against your fellow sleuths to find that unique answer.

"Mystery at Marple Manor" may be a departure from the computer games you're used to playing. Patient strategy is more important than quick reflexes for the successful detective. In many ways, the game resembles computer text-adventures, as well as familiar board games of logic and deduction.

For Sleuths Only

Type in and save Mystery at Marple Manor. Use "The Automatic Proofreader" (Appendix C) to insure an error-free copy of the program the first time. Although the program is a bit long, much of it is in the form of PRINT statements, which should be easier to enter than other BASIC program statements.

In order to solve the case, you must correctly identify the murderer, the victim, the weapon used, and the room where the heinous deed was done. Before you arrived, the manor held ten people and twelve possible weapons; however, the murderer has fled to parts unknown with the weapon he or she used, leaving behind the body of the victim, eight potential witnesses, and only eleven weapons.

As you travel through the mansion, use paper and pencil to keep a careful record of all suspects and weapons you see. When you've located all the objects that remain in the house, use the process of elimination to identify the murderer and weapon used. The victim's body is also in one of the rooms; once you find it, you can record the victim's identity and the scene of the crime.

It sounds simple, but there are complications. At the outset, many of the suspects and weapons will be hidden in the various nooks and crannies of the manor. You and your fellow detectives may have to search each room thoroughly, possibly several times, before all the concealed items are discovered. The detectives can even pick up and move items from room to room in the course of play. Suspects and the body of the victim cannot be moved, but they can be hidden by detectives in the same room.

Marple Manor is a house of 14 rooms. Part of the fun of a game like this is to discover the floor plan. (If you really want help in the form of a map, you can refer to the September 1984 issue of *COMPUTE!'s Gazette,* where this game originally appeared.) Up to six people can play, and all players begin the game in the foyer at the southern end of the house. Detectives alternate turns until one correctly solves the mystery, or until all have made incorrect guesses and, consequently, have been eliminated from the game. Although each player takes a separate turn, the game works just as well if the players form teams of equal size. This allows two or three teammates to travel through the house independently, yet share their discoveries and arrive at a solution together.

Passwords and Locked Doors

The game begins with a title screen and a thunderclap. This gives detectives time to assign player numbers, organize teams (if desired), and ready their notepads. Type a number from 1 to 6 to enter the number of players, and the game begins.

At this point, all players except the first should position themselves so they can't see the screen. After all, each player will be acquiring information in the course of the game that he or she wishes to keep secret from the others for as long as possible. To help preserve secrecy, you'll be asked to enter a password code on your first turn. This password can be any two characters from the keyboard—numerals, letters, spaces, special symbols, or even function keys. Be sure to choose a code that you can recall easily, and bear in mind that the computer will recognize shifted keys and unshifted keys as different entries. On later turns, you must enter your secret code before going on. This prevents other players from illegally using your turn to gather information for themselves.

After you type in your password, the computer reminds you of your current location and asks if you wish to move. If you answer yes, the computer lists all available exits. Type in the appropriate compass direction (N, S, E, or W) to move to a new room. If you try to move in a direction that doesn't have a matching door—for example, if you try to move south from the foyer—your move will be blocked.

Your move may also be blocked if you attempt to move through a locked door. Eleven doors in Marple Manor can be fastened shut, and at the start of the game, most of these doors are locked. To move through a locked door, you must possess a key which matches the lock; for example, the bedroom key will open any door that adjoins the bedroom. All of these keys are initially placed in the pantry. One special key, the skeleton key, can open any locked door but is powerless to lock doors; its starting location will vary from game to game.

Whether or not you move to a new room, the computer describes your surroundings. It tells you the room you're in; notes what item you carry, if any; lists all suspects, weapons, and keys in view; and names all the other players in the room.

Searching for Clues

Following the description, you'll see a list of choices. Select from these options by pressing the appropriate key. One option is to take no action; this allows you to end your turn and readies the computer for the next player.

Searching is the most popular option. At the start of play, many suspects and items are hidden in various rooms. Additionally, players may use the *Hide* option to stash away even more clues. Searching is the only way to find these hidden objects. Each time a player searches in a given room, there's a 50 percent chance of finding each item hidden in that area. For this reason, a room may be searched several times before all the objects it contains are revealed. A searching player does not automatically take any item he finds.

The Hide option is the logical counterpart to the Search. You may choose to hide any one object in the room you occupy. This object may be a weapon, a suspect, a key, or the corpse. You may even hide the object you're carrying. But you can't hide yourself or another player. Hiding items makes it more difficult for your opponents to locate the clues they need in order to win. Don't forget, of course, to record each clue in your notes before you hide it. Hidden objects may be subsequently discovered by any player searching in the room.

The *Take* option allows you to pick up a weapon or key in the room you occupy. You may carry only one item at any time. If you choose the Take option while holding an object, you automatically drop the object you're holding. Alternatively, the *Drop* option allows you to discard an item without taking another. The usefulness of the Take option cannot be overstated: Carrying keys allows you to pass through locked doors, while weapons in your possession cannot be discovered by players who search. However, the *Pilfer* option allows a player to steal from another player in the same room. The pilfering player drops any item carried, and takes the object the other player had held.

When you're certain you have the solution to the case, select the *Accuse* option. You'll be asked to identify the murderer, the victim, the weapon, and the scene of the crime from lists of the possibilities. *An incorrect guess eliminates you from further play.* Give the correct solution, though, and you win the game.

Mystery at Marple Manor

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C.

```
9 POKE53280,1:POKE53281,Ø:S=54272:FORJ=ØTO24:POKES
  +J,Ø:NEXT:POKES+24,15                        :rem 35
12 PRINT"{CLR}{6 DOWN}"TAB(7)"{8}{RVS}{*}{4 RIGHT}
   £"                                          :rem 121
13 PRINTTAB(7)"{RVS} {*}{2 RIGHT}£ ":PRINTTAB(7)"
   {RVS}{2 SPACES}{*}£{2 SPACES}{OFF} YSTERY"
                                               :rem 238
```

```
15 PRINTTAB(7)"{RVS} B{2 SPACES}B ":PRINTTAB(7)"
   {RVS} B{2 SPACES}B ":PRINT"{3 UP}"TAB(21)CHR$(1
   42);                                      :rem 153
24 GOSUB1713:PRINT"AT"                       :rem 82
27 PRINT"{DOWN}"TAB(12)"E5{RVS}E*{4 RIGHT}£":PR
   INTTAB(12)"{RVS} E*{2 RIGHT}£ "           :rem 42
28 PRINTTAB(12)"{RVS}{2 SPACES}E*£{2 SPACES}
   {OFF} ARPLE"                              :rem 102
30 PRINTTAB(12)"{RVS} B{2 SPACES}B ":PRINTTAB(12)"
   {RVS} B{2 SPACES}B "                      :rem 129
33 PRINT"{UP}"TAB(17)"E4{RVS}E*{4 RIGHT}£":PRIN
   TTAB(17)"{RVS} E*{2 RIGHT}£ ":PRINTTAB(17)"
   {RVS}{2 SPACES}E*£{2 SPACES}{OFF} ANOR"
                                             :rem 167
36 PRINTTAB(17)"{RVS} B{2 SPACES}B ":PRINTTAB(17)"
   {RVS} B{2 SPACES}B "                      :rem 145
39 GOSUB1713                                 :rem 184
42 FORJ=1TO1000:NEXT                         :rem 226
45 POKES+5,15:POKES+6,0:POKES+4,129          :rem 57
50 J=1:FORI=1TO15:POKE53281,J:POKE53280,1-J:rem 31
51 POKES+1,INT(RND(1)*20)+5                  :rem 254
53 J=1-J:FORP=1TO30:NEXT:NEXT                :rem 110
56 POKES+4,0                                 :rem 168
100 DEFFNR(X)=INT(RND(1)*X)+1:J=RND(-TI)     :rem 108
103 DIMP%(50),S$(22),R$(14),C$(6),V$(3),V(3),D%(10
    ,2)                                      :rem 56
112 FORJ=1TO10:P%(J)=FNR(11)+3:NEXT          :rem 46
115 FORJ=11TO22:P%(J)=FNR(13)+1:NEXT         :rem 101
118 FORJ=24TO31:P%(J)=4:NEXT                 :rem 169
121 P%(23)=FNR(8)+6                          :rem 204
124 J=FNR(10):P%(35)=J:P%(34)=P%(J):P%(J)=0:rem 16
127 J=FNR(10):IFP%(J)=0THEN127               :rem 200
130 P%(32)=J:P%(J)=0:J=FNR(12):P%(33)=J:P%(J+10)=0
                                             :rem 136
133 FORJ=1TO22:IFRND(1)<=.75THENP%(J)=-P%(J)
                                             :rem 56
136 READS$(J):NEXT                           :rem 65
139 FORJ=1TO14:READR$(J):NEXT                :rem 36
142 FORJ=0TO10:READD%(J,1),D%(J,2):IFRND(1)<.9THEN
    D%(J,0)=-1                               :rem 122
143 NEXT                                     :rem 215
145 FORJ=0TO3:READV$(J):NEXT                 :rem 242
148 P=2049:I=0:FORJ=4000TO7000STEP1000       :rem 188
151 IFJ=PEEK(P+2)+PEEK(P+3)*256THENDA(I)=P:I=I+1:G
    OTO157                                   :rem 54
154 P=PEEK(P)+PEEK(P+1)*256:GOTO151          :rem 11
157 NEXT                                     :rem 220
172 PRINT"{HOME}{21 DOWN}{BLK}{6 SPACES}HOW MANY P
    LAYERS (1-6) ?"                          :rem 218
175 GETA$:IFA$<"1"ORA$>"6"THEN175            :rem 75
```

```
178 I=VAL(A$):P%(49)=I                          :rem 178
181 FORJ=1TOI:P%(35+J)=1:NEXT                   :rem 233
190 PRINT"{CLR}{2 DOWN}E4}ALL PLAYERS EXCEPT PLAYE
    R #1 MUST LEAVE"CHR$(14)                     :rem 7
192 PRINT"THE ROOM AT THIS POINT.":PRINT"{DOWN}
    {3 SPACES}PLAYER # 1: PRESS {RVS} RETURN {OFF}
    "                                           :rem 152
193 PRINT"{7 SPACES}TO BEGIN THE GAME!"         :rem 146
194 GETA$:IFA$<>CHR$(13)THEN194                 :rem 14
196 POKE53280,12:POKE53281,15:Q=1               :rem 87
200 PRINT"{CLR}{2 DOWN}{BLK}PLAYER #"Q"-----------
    ----E4}{DOWN}"                              :rem 120
203 IFC$(Q)<>""THEN212                          :rem 175
206 PRINT"PRESS ANY TWO KEYS TO ESTABLISH YOUR"
                                                :rem 37
207 PRINT"SECRET CODE. WITH THIS CODE, NO OTHER"
                                                :rem 211
209 PRINT"PLAYER CAN STEAL YOUR TURN!":PRINT"
    {DOWN}ENTER YOUR CODE NOW!"                 :rem 214
210 GOSUB1700:C$(Q)=A$:GOTO218                  :rem 206
212 PRINT"{DOWN}ENTER YOUR SECRET CODE!":GOSUB1700
                                                :rem 72
215 IFC$(Q)<>A$THENI=0:GOSUB1710:GOTO200        :rem 124
218 PRINT"{CLR}{2 DOWN}{BLK}PLAYER #"Q"-----------
    ----E4}{DOWN}"                              :rem 129
221 R=P%(35+Q):PRINT"YOU ARE IN THE "R$(R)"."
                                                :rem 49
224 PRINT"DO YOU WISH TO LEAVE THIS ROOM [Y/N] ?"
                                                :rem 6
227 GETA$:IFA$="N"THENPRINT"NO":GOTO330         :rem 3
230 IFA$<>"Y"THEN227                            :rem 106
233 I=1:J=R:GOSUB1730:FORJ=0TO3:READV(J):NEXT
                                                :rem 85
236 PRINT"YES":PRINT"{DOWN}DOORS FROM THIS ROOM AR
    E FOUND TO THE:"                            :rem 187
239 FORJ=0TO3:IFV(J)<>0THENPRINTTAB(4);V$(J)
                                                :rem 222
242 NEXT:PRINT"{DOWN}TYPE {RVS}{BLK} N {OFF} ,
    {RVS} S {OFF} , {RVS} E {OFF} ,E4}OR {RVS}
    {BLK} W {OFF}E4} TO MOVE!":I=3              :rem 227
245 GETA$:IFA$=""THEN245                        :rem 89
248 A=ASC(A$)OR128:I=0:IFA<197ORA>215THEN245
                                                :rem 62
251 IFA=ASC(V$(I))THEN260                       :rem 168
254 I=I+1:IFI<4THEN251                          :rem 15
257 GOTO245                                     :rem 114
260 PRINT"GO "V$(I)                             :rem 147
261 IF V(I)<1THENPRINT"NO DOOR THIS WAY. YOU CAN'T
     MOVE.":GOTO1910                            :rem 154
```

```
263 IFV(I)<100THENR=V(I):PRINT"MOVING TO NEW ROOM.
    ":FORI=1TO1000:NEXT:GOTO330                :rem 166
266 Z=V(I)-100:IFD%(Z,0)=0THEN300               :rem 75
269 PRINT"THAT DOOR IS LOCKED":GOSUB1760       :rem 45
270 IFA=0THENPRINT"YOU DON'T HAVE A MATCHING KEY."
    :PRINT"NO MOVE.":GOTO1910                   :rem 65
272 PRINT"YOUR KEY OPENS THE DOOR.":GOSUB1770:PRIN
    T"MOVING TO NEW ROOM."                     :rem 200
300 I=D%(Z,1):IFI=RTHENI=D%(Z,2)                :rem 82
303 R=I:GOSUB1760:IFA<>1THEN330                :rem 112
306 PRINT"DO YOU WANT TO LOCK THIS DOOR BEHIND
    {4 SPACES}YOU{2 SPACES}[Y / N] ?"           :rem 96
309 GETA$:IFA$="N"THENPRINT"NO":GOTO330          :rem 4
312 IFA$<>"Y"THEN309                           :rem 108
315 PRINT"YES":GOSUB1770:PRINT"DOOR LOCKED.":rem 3
330 P%(Q+35)=R:PRINT"{DOWN}{CLR}{5 DOWN}YOU ARE IN
     THE "R$(R)"."                              :rem 43
333 PRINT"YOU CARRY ";:I=P%(Q+41):GOSUB1780:PRINT"
    ."                                         :rem 205
336 J=0:PRINT"YOU SEE THE FOLLOWING HERE:":rem 168
339 FORI=1TO31:IFP%(I)=RTHENJ=J+1:PRINT"{3 SPACES}
    ";:GOSUB1780:PRINT"."                       :rem 16
342 NEXT:FORI=1TO6:IFI<>QANDP%(35+I)=RTHENPRINT"
    {3 SPACES}PLAYER #"I".":J=J+1              :rem 252
345 NEXT:IFP%(34)=RTHENPRINT"{3 SPACES}THE BODY OF
     THE "S$(P%(35))".":J=J+1                  :rem 180
348 IFJ=0THENPRINT"NOTHING OF INTEREST."  :rem 173
351 PRINT"{DOWN}PRESS {RVS}{BLK} RETURN
    {SHIFT-SPACE}{OFF}E4} FOR OPTIONS....":rem 158
354 GETA$:IFA$<>CHR$(13)THEN354                 :rem 10
375 PRINT"{CLR}"                                 :rem 3
376 PRINT"{4 DOWN}{BLK}{3 SPACES}>>>>> TURN
    {SHIFT-SPACE}OPTIONS <<<<<{2 DOWN}":PRINT" E4}
    {RVS}A{OFF} ACCUSE THE MURDERER!"          :rem 129
377 PRINT"{SHIFT-SPACE}{RVS}D{OFF} DROP AN ITEM.":
    PRINT" {RVS}H{OFF} HIDE AN ITEM OR SUSPECT."
                                               :rem 224
379 PRINT" {RVS}N{OFF} NO ACTION.":PRINT" {RVS}P
    {OFF} PILFER FROM ANOTHER PLAYER."         :rem 240
381 PRINT" {RVS}S{OFF} SEARCH THE ROOM FOR HIDDEN
    {SPACE}ITEMS.":PRINT" {RVS}T{OFF} TAKE AN ITEM
    ."                                         :rem 143
384 PRINT"{2 DOWN}ENTER LETTER FOR ACTION DESIRED!
    {3 DOWN}"                                   :rem 89
387 GETA$:IFA$<"A"ORA$>"T"THEN387              :rem 131
390 PRINT"{CLR}":A=ASC(A$):ONA-64GOTO700,375,375,8
    00                                          :rem 30
393 IFA$="H"THEN970                             :rem 43
396 IFA$<"N"THEN375                             :rem 50
400 ONA-77GOTO450,375,880,375,375,930,820 :rem 154
```

```
450 PRINT"{2 DOWN}PRESS {RVS}{BLK} RETURN {OFF}E4}
     TO END YOUR TURN!"                          :rem 119
453 GETA$:IFA$<>CHR$(13)THEN453                  :rem 10
456 I=0:PRINT"{BLK}{CLR}{4 DOWN}PLAYER #"Q"=======
    ==== END TURN":GOSUB1710                     :rem 142
459 Q=Q+1:IFQ>P%(49)THENQ=1                      :rem 86
462 IFP%(Q+35)=0THEN459                          :rem 19
465 GOTO200                                      :rem 106
700 PRINT"{CLR}{DOWN}{BLK}{3 SPACES}***** MAKE AN
    {SPACE}ACCUSATION *****{DOWN}E4}":I=1 :rem 112
703 FORJ=1TO10:PRINTJ"{LEFT}:"TAB(5)"THE ";S$(J)".
    ":NEXT                                       :rem 163
706 PRINT"{3 DOWN}ENTER NUMBER OF MURDER VICTIM ";
    :INPUTJ                                      :rem 231
709 IFJ<>P%(35)THENI=0                           :rem 6
712 GOSUB1900                                    :rem 228
715 FORJ=1TO10:PRINTJ"{LEFT}:"TAB(5)"THE ";S$(J)".
    ":NEXT                                       :rem 166
718 PRINT"{3 DOWN}ENTER NUMBER OF MURDERER ";:INPU
    TJ                                           :rem 53
721 IFJ<>P%(32)THENI=0                           :rem 253
724 GOSUB1900                                    :rem 231
727 FORJ=1TO12:PRINTJ"{LEFT}:"TAB(5)"THE "S$(J+10)
    ".":NEXT                                     :rem 252
730 PRINT"{3 DOWN}ENTER NUMBER OF MURDER WEAPON ";
    :INPUTJ                                      :rem 226
733 IFJ<>P%(33)THENI=0                           :rem 1
736 GOSUB1900                                    :rem 234
739 FORJ=1TO14:PRINTJ"{LEFT}:"TAB(5)"THE "R$(J)"."
    :NEXT                                        :rem 116
742 PRINT"{3 DOWN}ENTER NUMBER OF MURDER ROOM ";:I
    NPUTJ                                        :rem 88
745 IFJ<>ABS(P%(34))THENI=0                      :rem 44
746 PRINT"{CLR}{5 DOWN}SUMMONING THE POLICE TO MAK
    E AN":PRINT"ARREST......"                    :rem 244
748 POKES+14,5:POKES+18,16:POKES+3,1:POKES+24,143:
    POKES+6,240:POKES+4,65:A=5389                :rem 163
751 FORJ=1TO200:R=A+PEEK(S+27)*3.5:POKES,RAND255:P
    OKES+1,INT(R/256):NEXT                       :rem 131
754 FORJ=0TO24:POKES+J,0:NEXT:POKES+24,15    :rem 44
757 FORJ=1TO2500:NEXT                            :rem 37
760 IFI=0THEN772                                 :rem 177
763 I=3:PRINT"YOUR SOLUTION IS CORRECT!":GOSUB1710
                                                 :rem 2
769 PRINT"{2 DOWN}PLAYER #"Q"HAS CRACKED THE CASE!
    ":GOTO787                                    :rem 158
772 I=2:PRINT"NO!...THAT WAS A FALSE ARREST!":GOSU
    B1710                                        :rem 232
775 GOSUB1800:P%(35+Q)=0:P%(50)=P%(50)+1:PRINT"YOU
    'RE OUT OF THE GAME!"                        :rem 85
```

```
778 IFP%(5Ø)<P%(49)THEN45Ø                          :rem 151
781 RESTORE:GOSUB1713:FORJ=1TO5ØØ:NEXT:GOSUB1713
                                                    :rem 9Ø
784 PRINT"{DOWN}ALL PLAYERS HAVE GIVEN INCORRECT":
    PRINT"SOLUTIONS TO THE CRIME!!"                 :rem 85
785 PRINT"{DOWN}NOBODY WINS !"                      :rem 51
787 PRINT"HERE IS THE CORRECT SOLUTION:":PRINT"THE
     "S$(P%(32))                                    :rem 192
789 PRINT"KILLED THE "S$(P%(35)):PRINT"IN THE "R$(
    ABS(P%(34)))","                                 :rem 19
791 PRINT"USING THE "S$(P%(33)+1Ø)".{2 DOWN}":END
                                                    :rem 254
8ØØ PRINT"{2 DOWN}{BLK}{3 SPACES}*** DROP AN ITEM
    {SPACE}***[4]":GOSUB18ØØ                        :rem 36
8Ø3 IFI=ØTHENPRINT"{DOWN}YOU WEREN'T CARRYING ANYT
    HING !":GOTO45Ø                                 :rem 88
8Ø6 PRINT"{DOWN}YOU DROP ";:GOSUB178Ø:PRINT".":GOT
    O45Ø                                            :rem 6Ø
82Ø PRINT"{2 DOWN}{BLK}{3 SPACES}*** TAKE AN ITEM
    {SPACE}***[4]":J=1:PRINT"{DOWN}THESE ITEMS ARE
     AVAILABLE:"                                    :rem 175
823 FORI=11TO31:IFP%(I)<>RTHEN829                   :rem 233
826 PRINTJ": ";:GOSUB178Ø:PRINT".":POKE9ØØ+J,I:J=J
    +1                                              :rem 7Ø
829 NEXT:IFJ=1THENPRINT"NO ITEMS.":GOTO45Ø :rem 6Ø
832 PRINT"{DOWN}ENTER NUMBER TO TAKE AN ITEM, OR":
    PRINT"ENTER ZERO TO TAKE NOTHING."              :rem 111
835 INPUT"WHAT ITEM DO YOU WANT";A:IFA<ØORA>=JTHEN
    835                                             :rem 137
838 IFA=ØTHENPRINT"{DOWN}NO ITEM TAKEN.":GOTO45Ø
                                                    :rem 234
841 GOSUB18ØØ:IFI<>ØTHENPRINT"YOU DROP ";:GOSUB178
    Ø:PRINT"."                                      :rem 82
844 I=PEEK(9ØØ+A):P%(I)=1ØØ+Q:P%(Q+41)=I   :rem 155
845 PRINT"YOU TAKE ";:GOSUB178Ø:PRINT".":GOTO45Ø
                                                    :rem 3Ø
88Ø PRINT"{2 DOWN}{BLK}{3 SPACES}*** PILFER FROM A
    NOTHER ***[4]":J=Ø                              :rem 46
881 PRINT"{DOWN}THESE PLAYERS ARE ALSO IN THE ROOM
    ..."                                            :rem 226
883 FORI=1TO6:IFP%(35+I)=RANDI<>QTHENPRINT"
    {3 SPACES}PLAYER #"I".":J=J+1                   :rem 141
886 NEXT:IFJ=ØTHENPRINT"NO OTHER PLAYERS ARE IN TH
    E ROOM!":GOTO45Ø                                :rem 222
889 PRINT"{DOWN}WHICH PLAYER WILL YOU STEAL FROM ?
    "
                                                    :rem 1Ø8
89Ø PRINT"ENTER NUMBER, OR PRESS ZERO."             :rem 1
892 INPUT"PILFER FROM PLAYER #";A:IFA<ØORA>P%(49)T
    HEN889                                          :rem 25Ø
893 IFA=ØTHENPRINT"NO THEFT.":GOTO45Ø               :rem 179
```

```
895 IFA=QTHENPRINT"YOU CAN'T STEAL FROM YOURSELF!"
    :GOTO892                                   :rem 43
898 IFP%(35+A)<>RTHENPRINT"PLAYER #"A"IS NOT HERE
    {SPACE}!":GOTO889                         :rem 129
901 GOSUB1800:IFI<>0THENPRINT"YOU DROP ";:GOSUB178
    0:PRINT"."                                 :rem 79
904 I=P%(A+41):IFI=0THENPRINT"PLAYER #"A"CARRIED N
    O ITEM!":GOTO450                           :rem 33
907 P%(Q+41)=I:P%(A+41)=0:P%(I)=100+Q         :rem 158
908 PRINT"YOU TAKE ";:GOSUB1780:PRINT".":GOTO450
                                               :rem 30
930 PRINT"{2 DOWN}{BLK}{3 SPACES}*** SEARCH THE RO
    OM ***{4}":J=0:PRINT"{DOWN}YOU FIND THE FOLLOW
    ING:"                                     :rem 125
933 FORI=1TO31:IFP%(I)<>-RTHEN942             :rem 227
936 IFRND(1)>.5THEN942                          :rem 6
939 J=J+1:PRINTTAB(4);:GOSUB1780:PRINT".":P%(I)=R
                                              :rem 203
942 NEXT:IFP%(34)<>-RORRND(1)>.5THEN948        :rem 73
945 J=1:PRINT"{4 SPACES}THE BODY OF THE "S$(P%(35)
    )".":P%(34)=R                             :rem 200
948 IFJ=0THENPRINT"{2 SPACES}------NOTHING !"
                                              :rem 177
951 GOTO450                                   :rem 113
970 PRINT"{2 DOWN}{BLK}{3 SPACES}*** HIDE ITEM OR
    {SPACE}SUSPECT ***{4}":J=1                 :rem 57
971 PRINT"{DOWN}THESE CAN BE HIDDEN:"         :rem 187
973 FORI=1TO31:IFP%(I)<>RTHEN979              :rem 196
976 PRINTJ": ";:GOSUB1780:PRINT".":POKE900+J,I:J=J
    +1                                         :rem 76
979 NEXT:I=P%(Q+41):IFI=0THEN985              :rem 163
982 PRINTJ": ";:GOSUB1780:PRINT" (YOU CARRY IT).":
    POKE900+J,Q+41:J=J+1                       :rem 77
985 IFP%(34)=RTHENPRINTJ": THE BODY OF THE "S$(P%(
    35))".":POKE900+J,34:J=J+1                :rem 211
988 IFJ=1THENPRINT"NOTHING HERE CAN BE HIDDEN!":GO
    TO450                                     :rem 221
991 PRINT"{DOWN}ENTER NUMBER OF ITEM TO HIDE, OR":
    PRINT"ENTER ZERO TO HIDE NOTHING."        :rem 101
994 INPUT"WHAT WILL YOU HIDE";A:IFA<0ORA>=JTHEN994
                                              :rem 235
997 IFA=0THENPRINT"NOTHING HIDDEN.":GOTO450:rem 99
1000 I=PEEK(900+A):IFI>34THEN1009             :rem 114
1003 P%(I)=-R:IFI=34THENPRINT"YOU HIDE THE BODY.":
     GOTO450                                   :rem 37
1006 PRINT"YOU HIDE ";:GOSUB1780:PRINT".":GOTO450
                                               :rem 57
1009 I=P%(Q+41):PRINT"YOU HIDE THE OBJECT YOU CARR
     Y....":GOSUB1780:PRINT"."                 :rem 42
1012 P%(Q+41)=0:P%(I)=-R:GOTO450              :rem 233
1700 GETA$:IFA$=""THEN1700                    :rem 179
```

```
1703 GETB$:IFB$=""THEN1703                         :rem 187
1706 A$=A$+B$:RETURN                               :rem 128
1710 J=1:GOSUB1730                                   :rem 6
1713 READW,I,J:POKES+2,I:POKES+3,J:READI,J:POKES+5
     ,I:POKES+6,J                                  :rem 129
1716 READZ:IFZ<0THENRETURN                         :rem 227
1719 POKES+1,INT(Z/256):POKES,ZAND255:READZ:POKES+
     4,W                                            :rem 61
1722 FORJ=1TOZ*100:NEXT:POKES+4,0:GOTO1716          :rem 85
1730 P=DA(I):IFJ=1THEN1736                         :rem 248
1733 FORI=1TOJ-1:P=PEEK(P)+PEEK(P+1)*256:NEXT
                                                   :rem 209
1736 P=P-1:POKE66,INT(P/256):POKE65,PAND255:RETURN
                                                    :rem 62
1760 A=0:I=P%(41+Q):IFI<23ORI>31THENRETURN          :rem 49
1763 IFI=23THENA=-1:RETURN                         :rem 112
1766 I=I-17:IFI=D%(Z,1)ORI=D%(Z,2)THENA=1          :rem 111
1769 RETURN                                        :rem 183
1770 IFD%(Z,0)=0THEND%(Z,0)=-1:RETURN              :rem 143
1773 D%(Z,0)=0:RETURN                              :rem 201
1780 IFI=0THENPRINT"NO ITEM";:RETURN                :rem 54
1783 IFI<23THENPRINT"THE "S$(I);:RETURN            :rem 147
1786 IFI=23THENPRINT"THE SKELETON KEY";:RETURN
                                                   :rem 212
1789 PRINT"THE "R$(I-17)" KEY";:RETURN              :rem 50
1800 I=P%(Q+41):IFI=0THENRETURN                    :rem 132
1803 R=P%(Q+35):P%(I)=R:P%(Q+41)=0:RETURN           :rem 69
1900 PRINT"{CLR}{DOWN}{BLK}{3 SPACES}***** MAKE AN
      ACCUSATION *****{DOWN}{4}":RETURN            :rem 204
1910 FORI=1TO2200:NEXT:GOTO330                      :rem 82
2000 DATA17,0,0,0,240,14435,1,12860,1,14435,7,0,4
                                                   :rem 122
2005 DATA12860,1,11457,1,10814,1,9634,1,9094,6,963
     4,8,0,8,-1                                    :rem 196
2020 DATA17,0,0,0,240,7217,1,6430,1,7217,8,0,7
                                                   :rem 236
2025 DATA5407,6,5728,6,4547,6,4817,24,-1           :rem 247
3000 DATA"COOK","BUTLER","GARDENER","CHAUFFER","DU
     KE","DUCHESS","NANNY"                         :rem 131
3005 DATA"OPERA STAR","AMBASSADOR","PRIME MINISTER
     ","CARVING KNIFE","ROPE"                        :rem 9
3010 DATA"BOX OF WEED KILLER","ANTIQUE MACE","DUEL
     LING PISTOL","FENCING FOIL"                   :rem 216
3015 DATA"ICE PICK","PLASTIC BAG","CHAIN SAW","HED
     GE TRIMMERS","POLO MALLET"                    :rem 208
3020 DATA"GARDEN SPADE","ENTRY FOYER","CORRIDOR","
     HALL","PANTRY","DINING ROOM"                   :rem 97
3025 DATA"KITCHEN","STUDY","BEDROOM","BATHROOM","C
     LOSET","GREENHOUSE","GARDEN"                  :rem 187
```

```
3030 DATA"POOL","GARAGE",2,13,2,14,3,7,3,8,3,11,7,
     9,8,9,8,10,11,12,12,13,13,14               :rem 163
3035 DATA"NORTH","EAST","SOUTH","WEST"           :rem 7
4000 DATA33,0,0,88,89,1804,6,2025,3,2145,6,2703,3
                                                :rem 149
4005 DATA2408,1,2551,1,2408,1,2551,1,2408,1,2551,1
     ,2408,1,2551,1,2703,8,-1                   :rem 81
5000 DATA5,3,0,2                                :rem 45
5005 DATA4,1,101,100                            :rem 240
5010 DATA104,103,102,1                          :rem 81
5015 DATA0,6,2,0                                :rem 49
5020 DATA6,0,1,0                                :rem 44
5025 DATA0,0,5,4                                :rem 51
5030 DATA102,105,0,0                            :rem 239
5035 DATA0,107,106,103                          :rem 94
5040 DATA106,0,0,105                            :rem 244
5045 DATA0,0,0,107                              :rem 148
5050 DATA0,0,104,108                            :rem 246
5055 DATA0,108,109,0                            :rem 0
5060 DATA109,100,0,110                          :rem 86
5065 DATA101,110,0,0                            :rem 242
6000 DATA65,255,0,9,0,1804,6,1804,4.4,1804,1.5,180
     4,6,2145,4.5,2025,1.5                      :rem 202
6005 DATA2025,4.5,1804,1.5,1804,4.5,1804,1.5,1804,
     12,-1                                      :rem 177
7000 DATA33,0,0,88,89,2408,4,3215,12,3608,1.33,240
     8,1.33,3608,1.33                           :rem 223
7005 DATA4050,4,4050,4,4050,4,4050,1.33,4291,1.33,
     3215,1.33                                  :rem 116
7010 DATA4050,6,3608,2,3215,8,-1                :rem 77
```

Screen-80
80 Columns for the 64

Gregg Peele and Kevin Martin

Did you ever wish for an 80-column screen? "Screen-80" transforms your 64 into an 80-column machine without affecting the normal screen-editing keys. We've also included "Custom-80," which allows you to create your own 80-column character set with a joystick.

"Screen-80" offers a full 80-column screen and gives you the ability to use your Commodore 64 to write, edit, and even run BASIC programs (including some commercial software), all with an 80-column display. You can even use all the cursor controls of the normal screen editor. It runs concurrently with the normal system, allowing a quick switch between 40- and 80-column modes.

Best of all, little memory is used by Screen-80. The program consists of approximately 3K of machine language which goes into RAM "underneath" ROM. Another 43 bytes are placed in a little-used area of RAM (locations 710–753). Since the bulk of Screen-80 uses the same memory locations as the operating system, and the locations of the 43 bytes used from RAM are normally unused anyway, Screen-80 works without *any* apparent loss of programming space.

Enter and Sign In

Like many machine language programs in COMPUTE! books, Screen-80 is listed in MLX format. MLX makes it much simpler to type in machine language programs, and virtually insures a working copy the first time. Before you begin entering Screen-80, then, you must first type in the MLX program found in Appendix D. You'll want to have a copy of MLX, since it's used to enter machine language programs in COMPUTE! books, *COMPUTE!* magazine, and *COMPUTE!'s Gazette*.

MLX even has a built-in numeric keypad to ease the burden of typing all those numbers.

If you're using tape to store programs, you must make one slight change to MLX before you save it. *This change is only for MLX, and is only necessary when you type in Screen-80.* Change line 763 of MLX to read:

POKE780,1:POKE781,DV:POKE782,0:SYS65466

The only change is that the POKE782,1 in the original listing has been altered to a POKE782,0. Save this version of MLX for entering Screen-80. When you've typed in, saved, and tested Screen-80, you can change the POKE782 back to its original form so that you can use MLX to enter other machine language programs from this book. *Remember, this change is only for tape users.*

After you've loaded and run MLX, you'll be asked for the starting and ending addresses of Screen-80. Those addresses are:

Starting address: 49152
Ending address: 52811

As soon as you've provided those addresses, you can begin typing in Program 1. Just follow the directions in Appendix D, and you shouldn't have any problems. You can even enter the program in several sessions if you want. Once you've completed the typing and saved the program to tape or disk, turn your computer off and then on again.

Now load the program from disk or tape using the normal format for loading BASIC programs:

LOAD *"filename"*,8 (for disk)
LOAD *"filename"* (for tape)

Notice that you can load the program without the ,1 that generally accompanies ML programs. If the program loaded correctly, you can list it. You should see one line, line 0, with a SYS command (SYS 2061). Simply type RUN and you'll have Screen-80. To disable the program and return to normal 40-column mode, press RUN/STOP–RESTORE; typing SYS 710 and pressing RETURN will reenable Screen-80.

You can make a backup copy of the program by simply saving it as you would any BASIC program:

SAVE *"filename"*,8 (for disk)
SAVE *"filename"* (for tape)

To begin programming in 80 columns, just type NEW.

Screen-80 is still in memory, but now the bothersome line 0 you saw earlier has been erased.

Using 80 Columns

Once you enter 80-column mode, the first thing you're likely to notice is the smaller size characters. Since increasing the size of the screen is impossible, adding 40 more columns to the 64 makes it necessary to halve the size of each character. Some televisions may not produce a clear enough picture to make these smaller characters readable, so you may find it difficult to read text in 80 columns. We recommend using a video monitor with the color turned off. You may want to change the character set to suit your personal taste or needs. "Custom-80" (discussed later in the article) is designed to let you do just that.

Screen-80 provides a different cursor than does the normal 40-column mode. Rather than a blinking block, it uses an underline character; like the normal cursor, it can move anywhere on the screen. In fact, you can use all the cursor control keys, just as you would normally, to insert or delete, home the cursor, clear the screen, or create BASIC program lines.

Both uppercase/graphics and lower/uppercase modes are supported in Screen-80, but you cannot toggle between these modes with the SHIFT-Commodore key combination. Instead, you can put the screen editor in lower/uppercase mode by pressing the CTRL and N keys simultaneously, or by printing CHR$(14). (You can do this either through a program, or in direct mode. Simply type PRINT CHR$(14) in direct mode and the display changes to lower/uppercase.) To return to uppercase/graphics mode, print CHR$(142) to the screen, again either through a program or in direct mode. These methods affect only characters printed after these commands. Thus, you may have both sets (for example, graphics and lowercase) on the screen at the same time for increased programming flexibility.

You can change the color of the background, text, or border by simply POKEing the appropriate color number into location 53281 (for the background), location 646 (for text), or location 53280 (for the border). Changing text color changes the color of all text on the screen. If you want to change the background or text color during program mode, print a CHR$(13) after POKEing the appropriate location. Since color memory is fixed on the 64, it's impossible to have true 80-column color. Therefore, Screen-80 does not recognize color codes in PRINT statements as being any different from other graphics characters. All printing to the screen uses the color specified in location 646.

Graphics and Sound Routines

Screen-80 can be used with sprites, high-resolution graphics, and sound—just like the normal Commodore 64 screen. Since this program actually uses a hi-res screen, you can also use it for other graphics displays. You can even have text and hi-res graphics on the screen at the same time. (Check your *Commodore 64 Programmer's Reference Guide* for more detailed information on how to plot points on the hi-res screen.)

To plot points (or do anything else) to the hi-res screen, it's important to know how to POKE and PEEK to the screen. The hi-res screen for Screen-80 is located at 57344 ($E000). Since this screen memory shares addresses with ROM, you may POKE graphics safely to the screen, but attempting to PEEK from the screen will give you values from the ROMs. To get the equivalent of PEEKing these screen locations, type in and run the following routine. Make sure Screen-80 is already in memory.

```
10 FORT= 49152TO49175:READE:POKET,E:NEXT:POKE785,0
   :POKE786,192
20 DATA 32,247,183,120,162,53,134,1,160,0
30 DATA 177,20,162,55,134,1,88,168,169,0
40 DATA 32,145,179,96
```

Instead of using the normal PRINT PEEK (*location*), use:

PRINT USR (*location*)

The value returned is the content of the specified screen location. *Technical note: This routine is completely relocatable. Simply change the range of the FOR-NEXT loop and the values POKEd into addresses 785 and 786 in line 10 to match the routine's new location. Notice that the values POKEd into 785 and 786 are in low byte/high byte format.*

Using sprites in Screen-80 requires that all sprite data be kept within the same 16K block as the hi-res screen. Locations 49152 ($C000) to 53247 ($CFFF) are perfect places to put sprite data. The sprite pointers for Screen-80 are located at 53248+1016 (54264) to 53248+1023 (54271). To cause sprite 0 to get its data from 49152 ($C000), put a zero into location

53248+1016 (54264). Since POKEs to this area of memory are normally intercepted by the I/O chip, you must disable interrupts and I/O to put a value into these locations. The following lines will put a sprite onto the 80-column screen. Type it in and run it to see the effect.

```
10 V=53248
20 POKE V,100:POKE V+1,100
30 POKE V+39,2
40 POKE 56334,PEEK(56334)AND254
50 POKE 1,PEEK(1)AND251
60 POKE 53248+1016,0
70 POKE 1,PEEK(1)OR4
80 POKE 56334,PEEK(56334)OR1
90 POKE V+21,1
```

Creating sound from within Screen-80 is done exactly the same way as from the normal screen. In fact, since you POKE the information to the SID chip (in the I/O area) to create sound, you don't have to disable interrupts or do any bank switching, as was necessary for hi-res graphics or sprites. The normal POKEs will do.

Using Other Programs with Screen-80

This program is designed to intercept any calls to the normal Kernal PRINT routine ($FFD2). Software which bypasses this routine or POKEs directly to the screen will not work correctly with Screen-80. An example of a program which bypasses the PRINT vector is the DOS wedge program (on the *TEST/DEMO* disk which comes with Commodore's 1541 disk drives). Fortunately, this problem can easily be fixed by changing all PRINTs to pass through the standard vector. The routine below, when used in place of the normal DOS boot program ("C-64 Wedge"), changes these references.

```
10 IF A=0 THEN A=1:LOAD"DOS 5.1",8,1
20 FOR I=1 TO 7:READ A:POKEA,210:POKE A+1,255:NEXT
30 DATA 52644,52650,52712,52726,52752,52765,53075
40 SYS 52224
```

With these changes, the DOS support program will work with Screen-80.

Programs which depend on sprites should be avoided, as should programs which move screen memory or otherwise change the normal configuration of the 64.

SpeedScript, COMPUTE!'s popular word processing program, does not use the PRINT vector at $FFD2 to update the screen, so it's incompatible with Screen-80. Sorry.

Custom-80: Creating Your Own Character Set

Program 2, Custom-80, allows you to create your own character set for use with Screen-80. It's easy to use and requires a joystick plugged into port 2.

Custom-80 "borrows" the character set from Screen-80 and then moves it to a safe location in memory for editing. After editing, you can return the custom characters to the Screen-80 program, or save your new character set to disk or tape. Like Screen-80, it's in MLX format. After you've loaded and run the MLX program, enter these two numbers for starting and ending addresses:

Starting address: 49152
Ending address: 51245

Then begin typing in Program 2. Once you're finished, save a copy to tape or disk, turn your computer off, then on again. To load Custom-80, type:

LOAD"CUSTOM-80",8,1 (for disk)
LOAD"CUSTOM-80",1,1 (for tape)

(This assumes you used *CUSTOM-80* for the filename. Note that Custom-80 requires the ,1 notation, unlike Screen-80.)

After loading Custom-80 into memory, type NEW to reset the BASIC pointers. Next, load Screen-80 into memory and type SYS 49152. This puts you in Custom-80 and, at the same time, accesses the character set included with Screen-80.

The Screen-80 character set is displayed in the lower half of the screen, where the character being edited is framed by a yellow cursor. In the upper-left corner of the screen, the character is enlarged for editing; brief instructions are provided to the right.

Customizing Characters

You can choose which character you want to edit by moving the yellow cursor around the bottom display using either the joystick or the cursor keys. The cursor keys are faster. The flashing blue square in the upper-left display indicates the current pixel in the character you are editing. To set the pixel, press the fire button on the joystick. If it was blank, it becomes filled. Hitting the fire button again blanks the pixel.

Press SHIFT and CLR/HOME to clear all the pixels in the character you're editing. (This will not affect the characters previously edited.) To home the cursor to the first character, press CLR/HOME without pressing SHIFT.

You can copy a character from one position to another, by pressing the f1 key to store the current character into the buffer. Move the yellow cursor to the new position of the character and press f7 to retrieve it from the buffer.

Pressing the S key saves the character set to tape or disk as a short program file. It can be loaded back into memory by hitting the the L key. When loading or saving, you'll first be asked for the name of the file, then asked to press T for tape or D for disk. If an error occurs during a disk operation, the program will display the message.

If you wish to make the new character set a permanent part of Screen-80, press X. This puts the redefined character set back into Screen-80 and exits to BASIC. You can then save the new version of Screen-80 to disk with the redefined characters already in the program by entering:

SAVE"*filename*",8 (for disk)
SAVE"*filename*",1 (for tape)

where *filename* is your new name for Screen-80. (You'll probably want to scratch the old version of Screen-80 to prevent any possible confusion.) The next time you run Screen-80, you'll have your new character set in the program.

If you wish to use various character sets with Screen-80, you should save the character sets to tape or disk with Custom-80's S option, then load the individual character sets by using Program 3 while in Screen-80. This program loads the new character set into Screen-80 *after* it's activated. When the program prompts you for the name of the character set you want to load, enter the filename, comma, and the number of the device you want to load the character set from. Use 8 for disk, 1 for tape.

One important note: You cannot SYS to Custom-80 from Screen-80. You must press RUN/STOP–RESTORE to leave Screen-80 before typing SYS 49152 to run Custom-80.

How It Works

First, Custom-80 performs a block memory move of the character set data from Screen-80 to location 12288 ($3000 in hex). This is done to make it easier to display the character set at the bottom of the screen.

Next, a raster interrupt splits the screen to show both the redefined character set and the normal character set. The instructions and the enlarged character are printed on the top half of the screen. The enlarged character is a 4 × 8 matrix of reverse SHIFT-Os. Before entering the main loop, all variables are initialized.

The main loop has two major routines. The first one checks the joystick and keyboard. If a key is pressed, the appropriate flag is set. Pressing X sends the program to the routine that moves the character set back into Screen-80. The S key saves a character set, while the L key loads a character set.

The second routine prints the enlarged character on the screen. If any flags were set, this routine handles them. It takes care of the save-to-buffer routine, the get-from-buffer routine, the clear-character routine, and the routine that handles the flashing of the blue cursor in the enlarged character.

Program 1. Screen-80

For easy entry of the next two machine language programs, be sure to use "The Machine Language Editor: MLX," Appendix D.

```
49152 :011,008,000,000,158,050,227
49158 :048,054,049,000,000,000,157
49164 :160,044,185,065,008,153,115
49170 :198,002,136,192,255,208,241
49176 :245,160,000,169,160,133,123
49182 :252,132,251,169,008,133,207
49188 :254,169,109,133,253,177,107
49194 :253,145,251,200,208,249,068
49200 :165,252,201,173,240,007,062
49206 :230,254,230,252,076,042,114
49212 :008,076,198,002,169,054,055
49218 :133,001,032,000,160,169,049
49224 :055,133,001,096,072,169,086
49230 :054,133,001,104,032,028,174
49236 :162,072,169,055,133,001,164
49242 :104,096,072,169,054,133,206
49248 :001,104,032,148,161,072,102
49254 :169,055,133,001,104,096,148
49260 :169,090,141,250,255,169,158
49266 :169,141,251,255,173,002,081
```

```
49272 :221,009,003,141,002,221,205
49278 :169,252,045,000,221,141,186
49284 :000,221,169,032,013,017,072
49290 :208,141,017,208,169,072,185
49296 :141,024,208,169,000,141,059
49302 :244,173,169,011,141,134,254
49308 :002,169,000,141,243,173,116
49314 :133,212,141,236,173,169,202
49320 :015,141,033,208,169,015,237
49326 :141,032,208,032,244,160,223
49332 :032,003,164,169,210,141,131
49338 :038,003,169,002,141,039,066
49344 :003,169,226,141,036,003,002
49350 :169,002,141,037,003,032,070
49356 :099,160,096,160,000,185,136
49362 :116,160,141,227,173,032,035
49368 :042,162,200,192,129,208,125
49374 :242,096,147,013,029,029,010
49380 :029,029,029,029,029,029,146
49386 :029,029,029,029,029,029,152
49392 :029,029,029,029,029,029,158
49398 :029,029,029,029,056,048,210
49404 :032,067,079,076,085,077,156
49410 :078,083,032,070,079,082,170
49416 :032,084,072,069,032,067,108
49422 :079,077,077,079,068,079,217
49428 :082,069,032,054,052,017,070
49434 :017,157,157,157,157,157,060
49440 :157,157,157,157,157,157,206
49446 :157,157,157,157,157,157,212
49452 :157,157,157,157,157,066,127
49458 :089,032,071,082,069,071,208
49464 :071,032,080,069,069,076,197
49470 :069,017,017,157,157,157,124
49476 :157,157,157,157,157,157,242
49482 :157,157,157,157,157,157,248
49488 :065,078,068,032,075,069,211
49494 :086,073,078,032,077,065,241
49500 :082,084,073,078,160,000,057
49506 :173,033,208,041,015,170,226
49512 :173,134,002,010,010,010,187
49518 :010,141,237,173,138,013,054
49524 :237,173,032,058,169,153,170
49530 :000,208,153,000,209,153,077
49536 :000,210,200,008,032,074,140
49542 :169,040,208,216,160,231,134
49548 :032,058,169,153,000,211,251
49554 :032,074,169,136,192,255,236
49560 :208,242,096,072,169,001,172
49566 :141,244,173,104,032,042,126
```

```
49572 :162,169,ØØØ,141,244,173,Ø29
49578 :165,198,24Ø,252,12Ø,Ø32,153
49584 :18Ø,229,2Ø1,131,2Ø8,Ø16,117
4959Ø :162,ØØ9,12Ø,134,198,189,226
49596 :23Ø,236,157,118,ØØ2,2Ø2,1Ø9
496Ø2 :2Ø8,247,24Ø,228,2Ø1,Ø13,Ø51
496Ø8 :2Ø8,2Ø9,16Ø,ØØ7,Ø32,Ø58,1Ø6
49614 :169,177,251,Ø77,223,173,252
4962Ø :145,251,Ø32,Ø74,169,16Ø,Ø19
49626 :Ø79,132,2Ø8,Ø32,Ø58,169,128
49632 :177,2Ø9,2Ø1,Ø32,2Ø8,ØØ3,Ø3Ø
49638 :136,2Ø8,244,2ØØ,132,2ØØ,Ø7Ø
49644 :16Ø,ØØØ,132,211,132,212,Ø59
4965Ø :165,2Ø2,Ø48,Ø6Ø,165,2Ø2,Ø6Ø
49656 :133,211,197,2ØØ,144,Ø52,161
49662 :176,Ø94,165,153,2Ø8,Ø14,Ø4Ø
49668 :165,ØØ9,133,2Ø2,173,222,14Ø
49674 :173,133,2Ø1,133,214,Ø76,172
4968Ø :19Ø,161,Ø32,Ø74,169,Ø76,2Ø6
49686 :1Ø2,241,16Ø,ØØ7,Ø32,Ø58,11Ø
49692 :169,177,251,Ø77,223,173,Ø74
49698 :145,251,Ø32,Ø74,169,Ø76,Ø13
497Ø4 :Ø62,161,152,Ø72,138,Ø72,185
4971Ø :165,2Ø8,24Ø,23Ø,Ø32,Ø58,211
49716 :169,164,211,177,2Ø9,133,Ø91
49722 :215,Ø32,Ø74,169,Ø41,Ø63,14Ø
49728 :ØØ6,215,Ø36,215,Ø16,ØØ2,Ø42
49734 :ØØ9,128,144,ØØ4,166,212,221
4974Ø :2Ø8,ØØ4,112,ØØ2,ØØ9,Ø64,219
49746 :Ø32,Ø74,169,23Ø,211,Ø32,Ø62
49752 :132,23Ø,196,2ØØ,2Ø8,Ø26,Ø56
49758 :169,ØØØ,133,2Ø8,169,Ø13,Ø18
49764 :166,153,224,ØØ3,24Ø,ØØ6,124
4977Ø :166,154,224,ØØ3,24Ø,ØØ3,128
49776 :Ø32,Ø42,162,169,Ø13,Ø32,Ø5Ø
49782 :Ø74,169,133,215,1Ø4,17Ø,215
49788 :1Ø4,168,165,215,2Ø1,222,175
49794 :2Ø8,ØØ2,169,255,Ø24,Ø96,116
498ØØ :Ø72,165,154,2Ø1,ØØ3,2Ø8,171
498Ø6 :ØØ4,1Ø4,Ø76,Ø42,162,Ø76,Ø94
49812 :213,241,Ø72,141,227,173,191
49818 :152,Ø72,138,Ø72,169,ØØØ,245
49824 :141,235,173,Ø32,Ø7Ø,162,2Ø5
4983Ø :Ø32,Ø68,168,Ø32,146,168,Ø12
49836 :1Ø4,17Ø,1Ø4,168,1Ø4,Ø96,15Ø
49842 :173,227,173,Ø32,132,23Ø,121
49848 :2Ø8,ØØ6,169,ØØ1,141,235,176
49854 :173,Ø96,173,227,173,2Ø1,2Ø9
4986Ø :Ø32,144,ØØ3,Ø76,Ø97,162,198
49866 :Ø76,194,162,2Ø1,Ø96,176,Ø83
```

```
49872 :023,201,064,176,003,076,239
49878 :174,162,201,128,240,082,177
49884 :056,173,227,173,233,064,122
49890 :141,227,173,076,174,162,155
49896 :201,127,144,009,240,044,229
49902 :201,160,144,060,076,149,004
49908 :162,056,173,227,173,233,244
49914 :032,141,227,173,076,174,049
49920 :162,201,192,176,012,056,031
49926 :173,227,173,233,064,141,249
49932 :227,173,076,174,162,024,080
49938 :173,227,173,105,128,141,197
49944 :227,173,173,243,173,240,229
49950 :004,206,243,173,096,173,157
49956 :241,173,208,005,169,000,064
49962 :141,242,173,096,173,243,086
49968 :173,005,212,240,035,173,118
49974 :227,173,201,032,176,041,136
49980 :201,013,240,110,201,020,077
49986 :240,004,165,212,208,013,140
49992 :173,243,173,208,008,169,022
49998 :001,141,235,173,076,078,014
50004 :163,076,028,163,173,241,160
50010 :173,208,005,169,000,141,018
50016 :242,173,076,066,163,173,221
50022 :227,173,201,141,240,066,126
50028 :201,148,208,012,165,212,030
50034 :208,008,169,001,141,240,113
50040 :173,076,066,163,056,173,059
50046 :227,173,233,064,141,227,167
50052 :173,076,028,163,173,243,220
50058 :173,208,017,169,000,141,078
50064 :243,173,165,212,208,011,132
50070 :169,000,141,242,173,076,183
50076 :058,163,206,243,173,169,144
50082 :001,141,242,173,169,000,120
50088 :141,235,173,076,186,163,118
50094 :169,001,141,235,173,169,038
50100 :000,141,240,173,133,212,055
50106 :173,227,173,201,032,176,144
50112 :102,201,008,208,005,160,108
50118 :128,140,145,002,201,009,055
50124 :208,005,160,000,140,145,094
50130 :002,201,013,208,005,072,199
50136 :032,053,165,104,201,014,017
50142 :208,005,160,001,140,236,204
50148 :173,201,017,208,008,238,049
50154 :222,173,072,032,206,164,079
50160 :104,201,018,208,008,160,171
50166 :001,140,242,173,140,241,159
```

```
50172 :173,201,019,208,017,160,006
50178 :000,132,009,140,222,173,166
50184 :072,032,206,164,169,240,123
50190 :141,223,173,104,201,029,117
50196 :208,007,230,009,072,032,066
50202 :210,164,104,201,020,208,165
50208 :005,072,032,092,165,104,246
50214 :096,201,141,208,005,072,249
50220 :032,053,165,104,201,142,229
50226 :208,005,160,000,140,236,031
50232 :173,201,145,208,008,206,229
50238 :222,173,072,032,206,164,163
50244 :104,201,146,208,008,160,127
50250 :000,140,242,173,140,241,242
50256 :173,201,147,208,005,072,118
50262 :032,003,164,104,201,148,226
50268 :208,005,072,032,080,166,143
50274 :104,201,157,208,007,198,205
50280 :009,072,032,210,164,104,183
50286 :096,032,058,169,169,000,122
50292 :133,251,169,224,133,252,254
50298 :169,000,141,225,173,141,203
50304 :226,173,141,036,164,169,013
50310 :224,141,037,164,169,000,101
50316 :170,168,138,153,255,255,255
50322 :136,208,249,238,037,164,154
50328 :173,037,164,201,255,208,166
50334 :239,160,064,169,000,153,175
50340 :000,255,136,016,250,169,222
50346 :000,133,009,141,222,173,080
50352 :169,240,141,223,173,173,015
50358 :244,173,240,006,160,007,244
50364 :169,240,145,251,162,024,155
50370 :024,189,196,169,105,212,065
50376 :141,107,164,189,170,169,116
50382 :141,106,164,169,032,160,210
50388 :079,153,255,255,136,192,002
50394 :255,208,248,202,224,255,074
50400 :208,224,032,210,164,032,070
50406 :074,169,096,169,000,141,111
50412 :226,173,165,009,074,010,125
50418 :046,226,173,010,046,226,201
50424 :173,010,046,226,173,141,249
50430 :225,173,172,222,173,185,124
50436 :118,169,133,251,024,185,116
50442 :144,169,109,226,173,133,196
50448 :252,024,173,225,173,101,196
50454 :251,133,251,169,000,101,159
50460 :252,133,252,024,165,252,082
50466 :105,224,133,252,165,009,154
```

```
50472 :041,001,240,008,169,015,002
50478 :141,223,173,076,205,164,004
50484 :169,240,141,223,173,096,070
50490 :169,255,133,202,165,009,223
50496 :133,211,048,014,201,080,239
50502 :144,021,169,000,133,009,034
50508 :238,222,173,076,241,164,166
50514 :230,009,206,222,173,048,202
50520 :024,169,079,133,009,173,163
50526 :222,173,133,214,048,013,129
50532 :201,025,144,012,206,222,142
50538 :173,032,135,167,076,008,185
50544 :165,238,222,173,169,001,056
50550 :141,234,173,173,244,173,232
50556 :240,015,160,007,032,058,124
50562 :169,177,251,077,223,173,176
50568 :145,251,032,074,169,174,213
50574 :222,173,189,170,169,133,174
50580 :209,024,189,196,169,105,016
50586 :212,133,210,032,168,168,053
50592 :096,238,222,173,169,000,034
50598 :133,009,141,243,173,141,238
50604 :242,173,141,241,173,032,150
50610 :210,164,173,033,208,041,239
50616 :015,205,246,173,240,003,042
50622 :032,244,160,173,033,208,016
50628 :141,246,173,096,032,058,174
50634 :169,169,001,141,244,173,075
50640 :165,009,208,003,076,066,223
50646 :166,160,007,177,251,077,028
50652 :223,173,145,251,056,165,209
50658 :251,233,008,133,253,165,245
50664 :252,233,000,133,254,165,245
50670 :009,041,001,208,025,160,170
50676 :007,177,251,041,240,074,010
50682 :074,074,074,141,228,173,246
50688 :177,253,041,240,013,228,184
50694 :173,145,253,136,016,233,194
50700 :172,222,173,200,024,185,220
50706 :144,169,105,224,141,238,015
50712 :173,056,185,118,169,233,190
50718 :001,141,230,173,173,238,218
50724 :173,233,000,141,231,173,219
50730 :169,008,141,229,173,160,154
50736 :004,173,230,173,141,222,223
50742 :165,173,231,173,141,223,136
50748 :165,056,169,080,229,009,000
50754 :074,105,000,170,024,008,191
50760 :040,046,255,255,008,056,220
50766 :173,222,165,233,008,141,252
```

```
50772 :222,165,173,223,165,233,241
50778 :000,141,223,165,202,208,005
50784 :231,136,240,004,040,076,055
50790 :197,165,040,206,230,173,089
50796 :208,003,206,231,173,206,111
50802 :229,173,208,185,160,007,052
50808 :177,251,077,223,173,145,142
50814 :251,174,222,173,189,170,025
50820 :169,133,253,024,189,196,072
50826 :169,105,212,133,254,056,043
50832 :169,079,229,009,170,164,196
50838 :009,177,253,136,145,253,099
50844 :200,200,202,224,255,208,165
50850 :244,169,032,160,079,145,223
50856 :253,198,009,032,210,164,010
50862 :169,000,141,234,173,032,155
50868 :074,169,169,000,141,244,209
50874 :173,096,032,058,169,172,118
50880 :222,173,200,056,185,118,122
50886 :169,233,008,133,253,185,155
50892 :144,169,233,000,133,254,113
50898 :024,165,254,105,224,133,091
50904 :254,160,007,177,253,041,084
50910 :015,240,003,076,116,167,071
50916 :136,016,244,160,007,177,200
50922 :251,077,223,173,141,245,064
50928 :173,173,244,173,240,008,227
50934 :173,245,173,145,251,076,029
50940 :146,166,165,009,041,001,012
50946 :240,018,024,165,251,105,037
50952 :008,141,230,173,165,252,209
50958 :105,000,141,231,173,076,228
50964 :180,166,165,251,141,230,129
50970 :173,165,252,141,231,173,137
50976 :169,008,141,229,173,160,144
50982 :004,173,230,173,141,210,201
50988 :166,173,231,173,141,211,115
50994 :166,056,169,080,229,009,247
51000 :074,170,024,008,040,110,226
51006 :255,255,008,024,173,210,219
51012 :166,105,008,141,210,166,096
51018 :173,211,166,105,000,141,102
51024 :211,166,202,208,231,136,210
51030 :240,004,040,076,187,166,031
51036 :040,238,230,173,208,003,216
51042 :238,231,173,206,229,173,068
51048 :208,187,024,165,251,105,020
51054 :008,133,253,165,252,105,002
51060 :000,133,254,165,009,041,206
51066 :001,240,031,160,007,177,226
```

```
51Ø72 :251,Ø41,Ø15,Ø1Ø,Ø1Ø,Ø1Ø,2Ø9
51Ø78 :Ø1Ø,141,228,173,177,253,Ø92
51Ø84 :Ø41,Ø15,Ø13,228,173,145,243
51Ø9Ø :253,177,251,Ø41,24Ø,145,229
51Ø96 :251,136,Ø16,227,16Ø,ØØ7,181
511Ø2 :177,251,Ø77,223,173,141,176
511Ø8 :245,173,173,244,173,24Ø,132
51114 :ØØ5,173,245,173,145,251,138
5112Ø :Ø32,21Ø,164,Ø32,Ø58,169,Ø73
51126 :174,222,173,189,17Ø,169,255
51132 :133,253,Ø24,189,196,169,128
51138 :1Ø5,212,133,254,Ø56,169,Ø99
51144 :Ø79,229,ØØ9,17Ø,16Ø,Ø78,157
5115Ø :177,253,2ØØ,145,253,136,Ø9Ø
51156 :136,2Ø2,224,255,2Ø8,244,2Ø1
51162 :169,Ø32,164,ØØ9,145,253,222
51168 :169,ØØØ,141,234,173,Ø32,2Ø5
51174 :Ø74,169,173,243,173,2Ø1,239
5118Ø :Ø8Ø,24Ø,ØØ3,238,243,173,189
51186 :Ø96,Ø32,Ø58,169,169,224,222
51192 :141,158,167,Ø24,169,224,1Ø7
51198 :1Ø5,ØØ1,141,155,167,16Ø,215
512Ø4 :ØØØ,185,Ø64,255,153,ØØØ,149
5121Ø :255,2ØØ,2Ø8,247,238,155,Ø33
51216 :167,238,158,167,173,155,Ø5Ø
51222 :167,2Ø1,255,2Ø8,234,169,232
51228 :ØØØ,16Ø,ØØØ,153,ØØØ,254,Ø83
51234 :2ØØ,2Ø8,25Ø,16Ø,192,153,173
5124Ø :ØØØ,255,136,192,255,2Ø8,Ø62
51246 :248,Ø56,165,251,233,Ø64,Ø39
51252 :133,251,165,252,233,ØØ1,Ø63
51258 :133,252,162,ØØ1,189,17Ø,197
51264 :169,141,246,167,Ø24,189,232
5127Ø :196,169,1Ø5,212,141,247,116
51276 :167,2Ø2,189,17Ø,169,141,Ø9Ø
51282 :249,167,Ø24,189,196,169,Ø52
51288 :1Ø5,212,141,25Ø,167,162,1Ø1
51294 :ØØ8,16Ø,ØØØ,185,255,255,189
513ØØ :153,255,255,2ØØ,2Ø8,247,138
513Ø6 :238,247,167,238,25Ø,167,133
51312 :2Ø2,2Ø8,238,162,Ø24,189,111
51318 :17Ø,169,141,Ø29,168,Ø24,Ø51
51324 :189,196,169,1Ø5,212,141,112
5133Ø :Ø3Ø,168,169,Ø32,16Ø,Ø79,ØØØ
51336 :153,255,255,136,192,255,1Ø2
51342 :2Ø8,248,Ø32,Ø74,169,169,Ø18
51348 :127,141,ØØØ,22Ø,173,ØØ1,Ø42
51354 :22Ø,2Ø1,251,ØØ8,169,127,1Ø6
5136Ø :141,ØØØ,22Ø,Ø4Ø,2Ø8,ØØ9,Ø1Ø
51366 :16Ø,ØØØ,234,2Ø2,2Ø8,252,198
```

```
51372 :136,208,249,096,169,000,006
51378 :133,254,032,058,169,173,229
51384 :227,173,041,001,240,008,106
51390 :169,015,141,224,173,076,220
51396 :095,168,169,240,141,224,209
51402 :173,173,227,173,074,010,008
51408 :038,254,010,038,254,010,044
51414 :038,254,133,253,173,236,021
51420 :173,208,014,024,169,222,006
51426 :101,253,133,253,169,169,024
51432 :101,254,076,140,168,024,227
51438 :169,222,101,253,133,253,089
51444 :169,171,101,254,133,254,046
51450 :032,074,169,096,173,235,005
51456 :173,208,016,169,000,141,195
51462 :234,173,032,168,168,032,045
51468 :044,169,230,009,032,210,194
51474 :164,096,032,125,164,032,119
51480 :058,169,160,007,174,234,058
51486 :173,240,005,169,000,076,181
51492 :191,168,177,253,045,224,070
51498 :173,174,224,173,224,240,226
51504 :208,004,074,074,074,074,044
51510 :141,228,173,173,223,173,141
51516 :201,015,240,010,173,228,159
51522 :173,010,010,010,010,141,164
51528 :228,173,169,255,174,234,025
51534 :173,208,005,173,223,173,009
51540 :073,255,049,251,013,228,185
51546 :173,192,007,208,008,174,084
51552 :244,173,240,003,077,223,032
51558 :173,174,234,173,208,023,063
51564 :174,241,173,208,005,174,059
51570 :242,173,240,013,077,223,058
51576 :173,072,173,227,173,009,179
51582 :128,141,227,173,104,145,020
51588 :251,136,016,148,173,243,075
51594 :173,005,212,240,005,169,174
51600 :000,141,242,173,032,074,038
51606 :169,096,164,009,173,227,220
51612 :173,032,058,169,145,209,174
51618 :032,074,169,096,072,120,213
51624 :173,014,220,041,254,141,243
51630 :014,220,169,052,133,001,251
51636 :104,096,072,169,054,133,040
51642 :001,173,014,220,009,001,092
51648 :141,014,220,088,104,096,087
51654 :072,152,072,138,072,169,105
51660 :169,072,169,109,072,008,035
51666 :032,074,169,120,076,071,240
```

```
51672 :254,Ø32,Ø58,169,1Ø4,17Ø,235
51678 :1Ø4,168,1Ø4,Ø64,ØØØ,Ø64,214
51684 :128,192,ØØØ,Ø64,128,192,164
5169Ø :ØØØ,Ø64,128,192,ØØØ,Ø64,17Ø
51696 :128,192,ØØØ,Ø64,128,192,176
517Ø2 :ØØØ,Ø64,128,192,ØØØ,Ø64,182
517Ø8 :ØØØ,ØØ1,ØØ2,ØØ3,ØØ5,ØØ6,Ø13
51714 :ØØ7,ØØ8,Ø1Ø,Ø11,Ø12,Ø13,Ø63
5172Ø :Ø15,Ø16,Ø17,Ø18,Ø2Ø,Ø21,115
51726 :Ø22,Ø23,Ø25,Ø26,Ø27,Ø28,165
51732 :Ø3Ø,Ø31,ØØØ,Ø8Ø,16Ø,24Ø,Ø49
51738 :Ø64,144,224,Ø48,128,2Ø8,Ø74
51744 :Ø32,112,192,Ø16,Ø96,176,144
5175Ø :ØØØ,Ø8Ø,16Ø,24Ø,Ø64,144,214
51756 :224,Ø48,128,2Ø8,ØØØ,ØØØ,14Ø
51762 :ØØØ,ØØØ,ØØ1,ØØ1,ØØ1,ØØ2,Ø55
51768 :ØØ2,ØØ2,ØØ3,ØØ3,ØØ3,ØØ4,Ø73
51774 :ØØ4,ØØ4,ØØ5,ØØ5,ØØ5,ØØ5,Ø9Ø
5178Ø :ØØ6,ØØ6,ØØ6,ØØ7,ØØ7,ØØ7,1Ø7
51786 :Ø68,17Ø,17Ø,174,138,138,164
51792 :1Ø6,ØØØ,196,17Ø,168,2ØØ,152
51798 :168,17Ø,196,ØØØ,2Ø6,168,226
518Ø4 :168,174,168,168,2Ø6,ØØØ,2Ø8
5181Ø :228,138,136,234,138,138,Ø86
51816 :132,ØØØ,174,164,164,228,198
51822 :164,164,174,ØØØ,234,Ø42,12Ø
51828 :Ø42,Ø44,Ø42,17Ø,Ø74,ØØØ,232
51834 :138,142,142,138,138,138,19Ø
5184Ø :234,ØØØ,2Ø6,17Ø,17Ø,17Ø,Ø54
51846 :17Ø,17Ø,174,ØØØ,196,17Ø,246
51852 :17Ø,2Ø2,138,138,132,ØØ2,154
51858 :198,168,168,196,162,162,176
51864 :172,ØØØ,234,Ø74,Ø74,Ø74,Ø12
5187Ø :Ø74,Ø74,Ø78,ØØØ,17Ø,17Ø,212
51876 :17Ø,17Ø,174,174,Ø74,ØØØ,158
51882 :17Ø,17Ø,Ø74,Ø68,Ø68,164,116
51888 :164,ØØØ,23Ø,Ø36,Ø68,Ø68,23Ø
51894 :Ø68,132,23Ø,ØØØ,Ø7Ø,162,Ø76
519ØØ :13Ø,194,13Ø,13Ø,23Ø,ØØØ,234
519Ø6 :Ø32,114,Ø36,Ø47,Ø36,Ø34,237
51912 :Ø32,Ø32,ØØ4,ØØ4,ØØ4,ØØ4,Ø24
51918 :ØØ4,ØØØ,ØØ4,ØØØ,16Ø,17Ø,Ø32
51924 :Ø14,Ø1Ø,Ø14,Ø1Ø,ØØØ,ØØØ,ØØ4
5193Ø :Ø74,226,132,228,Ø36,232,122
51936 :Ø74,ØØØ,Ø66,162,164,Ø64,242
51942 :16Ø,16Ø,Ø96,Ø16,Ø4Ø,Ø68,ØØ2
51948 :13Ø,13Ø,13Ø,Ø68,Ø4Ø,ØØØ,222
51954 :ØØØ,16Ø,Ø68,238,Ø68,16Ø,168
5196Ø :ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,Ø14,ØØ6
51966 :ØØØ,Ø96,Ø32,Ø64,ØØ1,ØØ1,192
```

```
51972 :002,006,004,008,072,000,096
51978 :068,172,164,164,164,164,138
51984 :078,000,078,162,036,066,180
51990 :130,138,228,000,174,168,092
51996 :238,034,034,042,038,000,158
52002 :078,162,130,196,164,168,164
52008 :072,000,068,170,170,070,078
52014 :162,164,072,000,000,000,188
52020 :068,000,000,068,004,008,200
52026 :016,032,078,128,078,032,166
52032 :016,000,132,074,034,020,084
52038 :036,064,132,000,004,004,054
52044 :014,254,010,004,014,000,116
52050 :032,032,032,047,032,032,033
52056 :032,032,000,015,240,000,151
52062 :000,000,000,000,004,004,102
52068 :004,004,004,004,244,004,108
52074 :032,032,032,044,038,034,062
52080 :034,034,034,034,054,028,074
52086 :000,000,000,000,136,136,134
52092 :132,132,130,130,129,241,250
52098 :031,024,040,040,072,072,153
52104 :136,136,240,016,022,031,205
52110 :031,022,016,016,000,009,236
52116 :015,015,015,006,240,000,183
52122 :064,064,064,065,067,066,032
52128 :066,066,144,144,102,105,019
52134 :105,102,144,144,098,098,089
52140 :146,146,098,098,242,002,136
52146 :002,066,066,239,226,066,075
52152 :066,002,066,130,066,130,132
52158 :066,130,066,130,015,007,092
52164 :023,099,163,163,161,001,038
52170 :012,012,012,012,012,012,018
52176 :012,012,015,000,000,000,247
52182 :240,240,240,240,008,008,166
52188 :008,008,008,008,008,248,252
52194 :161,081,161,081,161,081,184
52200 :161,081,015,014,012,012,015
52206 :172,088,168,088,050,050,086
52212 :050,051,050,050,050,050,033
52218 :002,002,002,003,048,048,099
52224 :048,048,000,000,000,224,064
52230 :032,032,047,047,002,002,168
52236 :002,063,032,032,032,032,205
52242 :002,002,002,254,034,034,090
52248 :034,034,140,140,140,140,140
52254 :140,140,140,140,063,063,204
52260 :048,048,048,048,048,048,068
52266 :240,240,240,000,000,015,009
```

```
52272 :015,015,016,016,016,016,142
52278 :028,028,028,252,050,050,234
52284 :050,062,000,000,000,000,172
52290 :204,204,204,204,003,003,120
52296 :003,003,064,160,172,162,124
52302 :142,138,110,000,128,128,212
52308 :198,168,168,168,198,000,216
52314 :032,032,100,170,174,168,254
52320 :102,000,032,064,068,234,084
52326 :074,070,066,004,128,132,064
52332 :192,164,164,164,164,000,188
52338 :008,040,010,042,044,042,044
52344 :170,064,192,064,074,078,250
52350 :078,074,234,000,000,000,000
52356 :196,170,170,170,164,000,234
52362 :000,000,198,170,170,198,106
52368 :130,130,000,000,206,168,010
52374 :142,130,142,000,000,064,116
52380 :234,074,074,074,078,000,178
52386 :000,000,170,170,174,174,082
52392 :074,000,000,000,170,170,070
52398 :070,162,162,012,006,004,078
52404 :228,036,068,132,230,000,106
52410 :070,162,130,194,130,130,234
52416 :230,000,032,114,036,047,139
52422 :036,034,032,032,004,004,084
52428 :004,004,004,000,004,000,220
52434 :160,170,014,010,014,010,076
52440 :000,000,074,226,132,228,108
52446 :036,232,074,000,066,162,024
52452 :164,064,160,160,096,016,120
52458 :040,068,130,130,130,068,032
52464 :040,000,000,160,068,238,234
52470 :068,160,000,000,000,000,218
52476 :000,014,000,096,032,064,202
52482 :001,001,002,006,004,008,024
52488 :072,000,068,172,164,164,136
52494 :164,164,078,000,078,162,148
52500 :036,066,130,138,228,000,106
52506 :174,168,238,034,034,042,204
52512 :038,000,078,162,130,196,124
52518 :164,168,072,000,068,170,168
52524 :170,070,162,164,072,000,170
52530 :000,000,068,000,000,068,186
52536 :004,008,016,032,078,128,066
52542 :078,032,016,000,132,074,138
52548 :034,020,036,064,132,000,098
52554 :004,010,010,254,010,010,116
52560 :010,000,196,170,168,200,056
52566 :168,170,196,000,206,168,226
```

```
52572 :168,174,168,168,206,000,208
52578 :228,138,136,234,138,138,086
52584 :132,000,174,164,164,228,198
52590 :164,164,174,000,234,042,120
52596 :042,044,042,170,074,000,232
52602 :138,142,142,138,138,138,190
52608 :234,000,206,170,170,170,054
52614 :170,170,174,000,196,170,246
52620 :170,202,138,138,132,002,154
52626 :198,168,168,196,162,162,176
52632 :172,000,234,074,074,074,012
52638 :074,074,078,000,170,170,212
52644 :170,170,174,174,074,000,158
52650 :170,170,074,068,068,164,116
52656 :164,000,226,034,066,079,233
52662 :066,130,226,002,066,130,034
52668 :066,130,066,130,066,130,008
52674 :082,169,084,162,089,164,176
52680 :082,169,012,012,012,012,243
52686 :012,012,012,012,015,000,013
52692 :000,000,240,240,240,240,148
52698 :008,008,008,008,008,008,010
52704 :008,248,161,081,161,081,196
52710 :161,081,161,081,004,009,215
52716 :002,004,169,082,164,089,234
52722 :050,050,050,051,050,050,031
52728 :050,050,002,002,002,003,101
52734 :048,048,048,048,000,000,190
52740 :000,224,032,032,047,047,130
52746 :002,002,002,063,032,032,143
52752 :032,032,002,002,002,254,084
52758 :034,034,034,034,140,140,182
52764 :140,140,140,140,140,140,100
52770 :063,063,048,048,048,048,096
52776 :048,048,240,240,240,000,088
52782 :000,015,015,015,000,032,123
52788 :032,032,172,108,044,012,196
52794 :050,050,050,062,000,000,014
52800 :000,000,204,204,204,204,112
52806 :003,003,003,003,000,013,095
```

Program 2. Custom-80

```
49152 :169,000,032,144,255,169,001
49158 :132,133,178,169,003,133,242
49164 :179,169,075,133,251,169,220
49170 :018,133,252,169,000,133,211
49176 :253,169,048,133,254,160,017
49182 :000,177,251,145,253,200,032
49188 :208,249,230,252,230,254,179
```

```
49194 :165,252,2Ø1,Ø23,2Ø8,239,1Ø6
492ØØ :169,Ø11,141,Ø33,2Ø8,169,Ø11
492Ø6 :ØØØ,141,134,ØØ2,141,Ø32,248
49212 :2Ø8,169,147,Ø32,21Ø,255,Ø57
49218 :169,ØØØ,141,Ø62,ØØ3,141,Ø7Ø
49224 :17Ø,195,141,16Ø,195,141,Ø5Ø
4923Ø :172,195,141,173,195,169,Ø99
49236 :ØØ8,Ø32,21Ø,255,169,ØØ5,251
49242 :141,165,195,169,Ø13,141,146
49248 :248,ØØ7,169,ØØ7,141,Ø39,195
49254 :2Ø8,169,ØØ1,141,Ø21,2Ø8,Ø82
4926Ø :169,ØØØ,168,153,Ø64,ØØ3,153
49266 :2ØØ,192,Ø64,2Ø8,248,169,171
49272 :252,141,Ø64,ØØ3,141,Ø91,Ø44
49278 :ØØ3,16Ø,ØØ3,169,132,153,234
49284 :Ø64,ØØ3,2ØØ,2ØØ,2ØØ,192,223
4929Ø :Ø26,144,246,Ø32,Ø73,199,Ø9Ø
49296 :Ø32,159,192,Ø32,198,194,183
493Ø2 :Ø32,248,194,Ø32,Ø49,194,131
493Ø8 :Ø76,144,192,162,ØØØ,16Ø,122
49314 :ØØØ,Ø24,Ø32,24Ø,255,173,118
4932Ø :16Ø,195,Ø41,ØØ1,2Ø1,ØØ1,255
49326 :24Ø,ØØ5,169,24Ø,Ø76,183,Ø63
49332 :192,169,Ø15,141,163,195,Ø31
49338 :173,16Ø,195,Ø74,Ø1Ø,133,163
49344 :251,169,ØØØ,133,252,ØØ6,235
4935Ø :251,Ø38,252,ØØ6,251,Ø38,Ø1Ø
49356 :252,169,Ø48,Ø24,1Ø1,252,Ø26
49362 :133,252,173,163,195,Ø73,175
49368 :255,141,166,195,16Ø,ØØØ,1Ø9
49374 :169,Ø18,Ø32,21Ø,255,177,Ø59
4938Ø :251,Ø45,163,195,141,162,161
49386 :195,162,ØØØ,173,163,195,Ø98
49392 :2Ø1,Ø15,24Ø,Ø12,Ø78,162,18Ø
49398 :195,Ø78,162,195,Ø78,162,Ø92
494Ø4 :195,Ø78,162,195,173,162,193
4941Ø :195,Ø41,ØØ8,24Ø,ØØ5,169,148
49416 :ØØ1,Ø76,Ø14,193,169,ØØØ,2Ø5
49422 :Ø32,146,193,141,134,ØØ2,15Ø
49428 :169,2Ø7,Ø32,21Ø,255,173,Ø42
49434 :162,195,Ø41,ØØ4,24Ø,ØØ5,161
4944Ø :169,ØØ1,Ø76,Ø39,193,169,167
49446 :ØØØ,232,Ø32,146,193,141,Ø14
49452 :134,ØØ2,169,2Ø7,Ø32,21Ø,Ø3Ø
49458 :255,173,162,195,Ø41,ØØ2,11Ø
49464 :24Ø,ØØ5,169,ØØ1,Ø76,Ø65,1ØØ
4947Ø :193,169,ØØØ,232,Ø32,146,Ø66
49476 :193,141,134,ØØ2,169,2Ø7,146
49482 :Ø32,21Ø,255,173,162,195,Ø77
49488 :Ø41,ØØ1,24Ø,ØØ5,169,ØØ1,Ø25
```

```
49494 :Ø76,Ø91,193,169,ØØØ,232,Ø79
495ØØ :Ø32,146,193,141,134,ØØ2,228
495Ø6 :169,2Ø7,Ø32,21Ø,255,169,116
49512 :Ø13,Ø32,21Ø,255,173,163,182
49518 :195,2Ø1,Ø15,24Ø,Ø12,Ø14,Ø19
49524 :162,195,Ø14,162,195,Ø14,Ø9Ø
4953Ø :162,195,Ø14,162,195,177,ØØ3
49536 :251,Ø45,166,195,Ø13,162,192
49542 :195,145,251,2ØØ,192,ØØ8,1Ø1
49548 :24Ø,ØØ3,Ø76,222,192,Ø96,2Ø1
49554 :141,164,195,14Ø,169,195,126
4956Ø :173,17Ø,195,24Ø,ØØ8,169,Ø83
49566 :ØØØ,141,164,195,141,162,193
49572 :195,173,172,195,24Ø,ØØ6,121
49578 :173,162,195,153,178,ØØ2,ØØ9
49584 :173,173,195,24Ø,ØØ6,185,124
4959Ø :178,ØØ2,141,162,195,2Ø4,Ø4Ø
49596 :Ø61,ØØ3,2Ø8,1Ø6,236,Ø6Ø,Ø94
496Ø2 :ØØ3,2Ø8,1Ø1,238,Ø62,ØØ3,Ø41
496Ø8 :173,ØØØ,22Ø,Ø41,Ø16,2Ø8,Ø9Ø
49614 :Ø67,2Ø5,Ø63,ØØ3,24Ø,Ø65,Ø81
4962Ø :141,Ø63,ØØ3,169,ØØ4,Ø56,136
49626 :237,Ø6Ø,ØØ3,168,169,ØØ1,Ø88
49632 :136,24Ø,ØØ4,Ø1Ø,Ø76,224,146
49638 :193,141,168,195,Ø73,255,231
49644 :141,167,195,173,162,195,245
4965Ø :Ø45,168,195,2Ø8,Ø15,173,Ø22
49656 :162,195,Ø45,167,195,Ø13,ØØ1
49662 :168,195,141,162,195,Ø76,167
49668 :Ø21,194,173,162,195,Ø45,Ø26
49674 :167,195,141,162,195,Ø76,178
4968Ø :Ø21,194,141,Ø63,ØØ3,173,Ø99
49686 :Ø62,ØØ3,2Ø1,Ø5Ø,144,Ø14,24Ø
49692 :2Ø1,1ØØ,144,ØØ5,169,ØØØ,135
49698 :141,Ø62,ØØ3,169,Ø14,141,Ø52
497Ø4 :164,195,173,164,195,172,Ø79
4971Ø :169,195,Ø96,2Ø6,165,195,Ø48
49716 :2Ø8,Ø65,173,ØØØ,22Ø,Ø41,247
49722 :Ø15,141,162,195,Ø41,ØØ1,1Ø1
49728 :2Ø8,ØØ3,2Ø6,Ø61,ØØ3,173,2Ø6
49734 :162,195,Ø41,ØØ2,2Ø8,ØØ3,169
4974Ø :238,Ø61,ØØ3,173,162,195,14Ø
49746 :Ø41,ØØ4,2Ø8,ØØ3,2Ø6,Ø6Ø,Ø92
49752 :ØØ3,173,162,195,Ø41,ØØ8,158
49758 :2Ø8,ØØ3,238,Ø6Ø,ØØ3,173,Ø11
49764 :162,195,2Ø1,Ø15,24Ø,ØØ8,153
4977Ø :169,Ø51,141,Ø62,ØØ3,Ø32,Ø52
49776 :12Ø,194,169,ØØ5,141,165,138
49782 :195,Ø96,173,Ø6Ø,ØØ3,2Ø1,Ø78
49788 :255,2Ø8,ØØ8,169,ØØ3,141,14Ø
```

```
49794 :060,003,206,160,195,173,159
49800 :060,003,201,004,208,008,108
49806 :169,000,141,060,003,238,241
49812 :160,195,173,061,003,201,173
49818 :255,208,014,169,007,141,180
49824 :061,003,173,160,195,056,040
49830 :233,064,141,160,195,173,108
49836 :061,003,201,008,208,014,155
49842 :169,000,141,061,003,173,213
49848 :160,195,024,105,064,141,105
49854 :160,195,169,016,141,063,166
49860 :003,096,173,160,195,074,129
49866 :074,074,074,074,074,141,201
49872 :053,003,173,160,195,041,065
49878 :063,141,052,003,173,053,187
49884 :003,010,010,010,024,105,126
49890 :153,141,001,208,173,052,186
49896 :003,010,010,024,105,055,183
49902 :141,000,208,169,000,042,030
49908 :141,016,208,096,169,000,106
49914 :141,170,195,141,171,195,239
49920 :141,172,195,141,173,195,249
49926 :032,228,255,208,001,096,058
49932 :201,147,208,006,169,001,232
49938 :141,170,195,096,201,019,072
49944 :208,006,169,000,141,160,196
49950 :195,096,201,157,208,008,127
49956 :169,255,141,060,003,076,228
49962 :120,194,201,029,208,008,034
49968 :169,004,141,060,003,076,245
49974 :120,194,201,145,208,008,162
49980 :169,255,141,061,003,076,253
49986 :120,194,201,017,208,008,046
49992 :169,008,141,061,003,076,018
49998 :120,194,201,088,208,003,124
50004 :076,124,195,201,133,208,253
50010 :006,169,001,141,172,195,006
50016 :096,201,136,208,006,169,144
50022 :001,141,173,195,096,201,141
50028 :083,208,004,032,125,197,245
50034 :096,201,076,208,004,032,219
50040 :046,197,096,096,169,075,031
50046 :133,251,169,018,133,252,058
50052 :169,000,133,253,169,048,136
50058 :133,254,160,000,177,253,091
50064 :145,251,200,208,249,230,147
50070 :252,230,254,165,252,201,224
50076 :023,208,239,000,000,000,114
50082 :000,000,000,000,000,000,162
50088 :000,000,000,000,000,000,168
```

```
50094 :158,029,029,029,029,029,221
50100 :029,029,029,029,029,029,098
50106 :029,029,029,029,029,067,142
50112 :085,083,084,079,077,045,133
50118 :056,048,013,144,029,029,005
50124 :029,029,029,029,067,076,207
50130 :082,032,045,032,067,076,032
50136 :069,065,082,032,067,085,104
50142 :082,082,069,078,084,032,137
50148 :067,072,065,082,065,067,134
50154 :084,069,082,013,029,029,028
50160 :029,029,029,029,072,079,251
50166 :077,069,032,045,032,071,060
50172 :079,032,084,079,032,070,116
50178 :073,082,083,084,032,067,167
50184 :072,065,082,065,067,084,187
50190 :069,082,013,029,029,029,009
50196 :029,029,029,067,085,082,085
50202 :083,079,082,032,075,069,190
50208 :089,083,032,077,079,086,222
50214 :069,032,065,082,079,085,194
50220 :078,068,032,067,072,065,170
50226 :082,032,083,069,084,013,157
50232 :029,029,029,029,029,029,230
50238 :070,049,032,045,032,083,117
50244 :084,079,082,069,032,067,225
50250 :072,065,082,065,067,084,253
50256 :069,082,032,073,078,032,190
50262 :066,085,070,070,069,082,016
50268 :013,029,029,029,029,029,250
50274 :029,070,055,032,045,032,105
50280 :071,069,084,032,067,072,243
50286 :065,082,065,067,084,069,030
50292 :082,032,070,082,079,077,026
50298 :032,066,085,070,070,069,002
50304 :082,013,029,029,029,029,083
50310 :029,029,088,032,045,032,133
50316 :080,085,084,032,082,069,060
50322 :068,069,070,073,078,069,061
50328 :068,032,067,072,065,082,026
50334 :065,067,084,069,082,083,096
50340 :032,073,078,013,029,029,162
50346 :029,029,029,029,032,032,094
50352 :083,067,082,069,069,078,112
50358 :032,056,048,013,029,029,133
50364 :029,029,029,029,074,079,201
50370 :089,083,084,073,067,075,153
50376 :032,067,079,078,084,082,110
50382 :079,076,083,032,067,085,116
50388 :082,083,079,082,032,077,135
```

```
50394 :079,086,069,077,069,078,164
50400 :084,013,029,029,029,029,181
50406 :029,029,032,032,065,082,243
50412 :079,085,078,068,032,069,135
50418 :088,080,065,078,068,069,178
50424 :068,032,067,072,065,082,122
50430 :065,067,084,069,082,032,141
50436 :065,078,068,013,029,029,030
50442 :029,029,029,029,032,032,190
50448 :066,085,084,084,079,078,236
50454 :032,083,069,084,083,032,149
50460 :065,078,068,032,082,069,166
50466 :083,069,084,083,032,080,209
50472 :073,088,069,076,083,000,173
50478 :032,224,197,008,173,215,127
50484 :198,208,002,040,096,040,124
50490 :176,031,169,008,170,160,004
50496 :000,032,186,255,173,215,157
50502 :198,162,199,160,198,032,251
50508 :189,255,169,000,162,000,083
50514 :160,048,032,213,255,032,054
50520 :234,198,096,032,203,199,026
50526 :169,008,162,001,160,000,082
50532 :032,186,255,173,215,198,135
50538 :162,199,160,198,032,189,022
50544 :255,169,000,170,160,048,146
50550 :032,213,255,032,236,199,061
50556 :096,032,224,197,008,173,086
50562 :215,198,208,002,040,096,121
50568 :040,176,042,032,045,199,158
50574 :169,008,170,160,255,032,168
50580 :186,255,173,215,198,162,057
50586 :199,160,198,032,189,255,163
50592 :169,048,133,252,169,000,163
50598 :133,251,169,251,162,000,108
50604 :160,056,032,216,255,032,155
50610 :234,198,096,032,203,199,116
50616 :169,008,162,001,160,000,172
50622 :032,186,255,173,215,198,225
50628 :162,199,160,198,032,189,112
50634 :255,169,048,133,252,169,204
50640 :000,133,251,169,251,162,150
50646 :000,160,056,032,216,255,165
50652 :032,236,199,096,160,000,175
50658 :162,011,024,032,240,255,182
50664 :169,032,162,040,032,210,109
50670 :255,202,208,250,160,000,033
50676 :162,011,024,032,240,255,200
50682 :162,000,189,192,198,032,255
50688 :210,255,232,224,007,208,112
```

```
50694 :245,162,000,169,164,032,010
50700 :210,255,138,072,032,228,179
50706 :255,168,104,170,152,201,044
50712 :000,240,243,201,020,240,200
50718 :042,201,034,240,235,201,215
50724 :013,240,065,201,032,144,219
50730 :227,201,128,176,223,224,197
50736 :016,240,219,157,199,198,053
50742 :232,072,169,157,032,210,158
50748 :255,104,032,210,255,169,061
50754 :164,032,210,255,076,014,049
50760 :198,224,000,240,193,169,072
50766 :157,032,210,255,169,032,165
50772 :032,210,255,169,157,032,171
50778 :210,255,032,210,255,202,230
50784 :169,164,032,210,255,076,234
50790 :014,198,142,215,198,160,005
50796 :000,162,011,024,032,240,065
50802 :255,162,017,169,032,032,013
50808 :210,255,202,208,250,174,139
50814 :215,198,208,001,096,160,236
50820 :000,162,011,024,032,240,089
50826 :255,162,000,189,216,198,134
50832 :032,210,255,232,224,018,091
50838 :208,245,032,228,255,240,078
50844 :251,201,068,240,009,201,102
50850 :084,208,243,056,008,076,069
50856 :172,198,024,008,160,000,218
50862 :162,011,024,032,240,255,130
50868 :162,017,169,032,032,210,034
50874 :255,202,208,250,040,096,213
50880 :159,078,065,077,069,058,186
50886 :155,000,000,000,000,000,097
50892 :000,000,000,000,000,000,204
50898 :000,000,000,000,000,000,210
50904 :153,018,084,146,065,080,250
50910 :069,032,079,082,032,018,022
50916 :068,146,073,083,075,063,224
50922 :032,183,255,041,191,208,120
50928 :001,096,162,011,160,000,158
50934 :024,032,240,255,169,018,216
50940 :032,210,255,169,150,032,076
50946 :210,255,169,000,032,189,089
50952 :255,169,015,162,008,160,009
50958 :015,032,186,255,032,192,214
50964 :255,162,015,032,198,255,169
50970 :032,207,255,032,210,255,249
50976 :201,013,208,246,169,015,116
50982 :032,195,255,032,204,255,243
50988 :096,169,002,160,199,162,064
```

```
50994 :071,032,189,255,169,015,013
51000 :168,162,008,032,186,255,099
51006 :032,192,255,169,015,032,245
51012 :195,255,096,073,048,120,087
51018 :169,127,141,013,220,169,145
51024 :001,141,026,208,173,060,177
51030 :003,141,018,208,169,027,140
51036 :141,017,208,169,199,141,199
51042 :021,003,169,250,141,020,190
51048 :003,088,169,147,032,210,241
51054 :255,160,000,169,195,133,254
51060 :252,169,174,133,251,177,248
51066 :251,240,011,032,210,255,097
51072 :200,208,246,230,252,076,060
51078 :121,199,169,008,133,251,247
51084 :169,006,133,252,165,251,092
51090 :133,253,165,252,024,105,054
51096 :212,133,254,162,000,160,049
51102 :004,138,145,251,169,000,097
51108 :145,253,232,200,192,036,198
51114 :208,243,165,251,024,105,142
51120 :040,133,251,165,252,105,098
51126 :000,133,252,165,253,024,241
51132 :105,040,133,253,165,254,114
51138 :105,000,133,254,224,128,014
51144 :208,211,096,120,169,000,236
51150 :141,026,208,169,255,141,122
51156 :013,220,169,049,141,020,056
51162 :003,169,234,141,021,003,021
51168 :169,000,141,021,208,088,083
51174 :169,147,032,210,255,096,115
51180 :032,073,199,169,001,141,083
51186 :021,208,169,004,141,136,153
51192 :002,096,173,018,208,201,178
51198 :146,208,021,169,000,141,171
51204 :018,208,169,028,141,024,080
51210 :208,169,001,141,025,208,250
51216 :104,168,104,170,104,064,218
51222 :169,146,141,018,208,169,105
51228 :021,141,024,208,169,001,080
51234 :141,025,208,076,049,234,255
51240 :000,000,000,000,000,000,040
```

Program 3. Custom Character Loader

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C.

```
10 INPUT"FILENAME:";N$,D                          :rem 205
20 F$=N$:ZK=PEEK(53)+256*PEEK(54)-LEN(F$):POKE 782
   ,ZK/256                                         :rem 180
25 POKE781,ZK-PEEK(782)*256:POKE780,LEN(F$):SYS654
   69                                               :rem 39
30 POKE780,1:POKE781,D:POKE782,0:SYS65466 :rem 177
40 POKE780,0:POKE781,222:POKE782,169:SYS65493
                                                   :rem 115
50 CLOSE1:PRINT:PRINT"{CLR}"CHR$(142)            :rem 90
```

Screen Headliner

Todd Heimarck

This short machine language routine expands a letter to four times its normal size. The large character can then be used in a headline or for a variety of other purposes. The program is compatible with Commodore printers and can even be used with "Screen-80," the 80-column program which precedes this article.

Oversized characters can be useful—on a title screen, in a children's alphabet or math program, or for visually impaired computer users. Finding the right combination of graphics characters usually takes time; you have to experiment. And creating a whole alphabet can use up a lot of memory.

The simplest method for displaying huge letters without experimenting or wasting memory is to PEEK the character generator in ROM and print a solid block (reverse space) for each bit that is *on*. If the bit is *off*, you print a space. The one major disadvantage to this method is that each character expands to eight times its normal size. Very little space remains on the screen—your 64 would suddenly turn into a five-column screen. But by keeping in mind the idea of reading character ROM, we can sidestep this problem with some special Commodore characters.

The Quarter-Square Solution

Hold down the Commodore key and type IKBVDCF. These seven characters, plus a blank space, make up half of the quarter-square graphics set. The other half is accessed by typing the same keys while reverse is turned on. There are 16 different characters, one for each combination of quarter squares turned on or off.

Quarter squares enable you to set up what amounts to a medium-resolution screen. It's less complicated to program than a high-resolution screen, and has better resolution than the usual low-resolution character set. Instead of making characters turn on and off, you control big pixels (each of which is one-fourth of a character). A Commodore 64 suddenly has the capability to address 80 × 50 big pixels.

The 16 characters are the starting point for the "Screen Headliner." The basic idea is to read the character ROM, translate each bit into a big pixel, and print the equivalent quarter-square graphics character. You can do it in BASIC with a lot of PEEKs and POKEs, but machine language is faster and more elegant.

The program is easy to use. After entering and saving it, type RUN. A short machine language program is POKEd into memory. To make it work, you need two POKEs and a SYS:

POKE 249,0: POKE 250,1: SYS 828

You should see a large capital *A*, four characters wide and four deep. Now simultaneously press the Commodore and SHIFT keys to switch to the upper/lowercase set. Cursor up to the POKEs, press RETURN, and you'll see a large lowercase *a*. Now try putting a 129 into location 250; the result is the same character printed in reverse.

If you've saved a copy of Headliner, type NEW to erase the BASIC loader program. (It won't affect the ML program, which is safely tucked into the cassette buffer.) Now type this in:

```
1 MK=7
5 PRINT"{CLR}";
1Ø FORX=ØTO255
2Ø Y=(XANDMK)*4:POKE249,Y
25 IFXANDMKTHENPRINT"{4 UP}";
3Ø POKE25Ø,X:SYS828
4Ø NEXT
```

(Note: Tape users should not save this example program; tape operations erase Headliner from the cassette buffer.) Type RUN, and the whole Commodore character set will parade down the screen.

Making Letters

The top of the large character is printed wherever the cursor happens to be when you SYS. The POKE to 249 determines how far the cursor spaces over before it begins. The number must be between 0 and 35.

Next, POKE the letter's screen code into 250. Ignore the ASCII value, you want the screen code—the number you use when POKEing a character to the screen. Numbers 1 through

26 are the letters A–Z, 48–57 are the characters 0–9, and so on. To get a reversed character, add 128 to the screen code. (You can find a list of screen codes in Appendix E of the *Commodore 64 User's Guide,* the manual that came with your computer.)

After you've POKEd into 249 and 250, enter SYS 828. The oversize character appears almost instantly.

Three Bonuses and a Drawback

The original version of this routine (used in two programs published in *COMPUTE!'s Gazette* magazine—"Aardvark Attack" a year ago, and more recently "Campaign Manager") figured out the shape of the large character and POKEd the appropriate quarter-square graphics to the screen. But Headliner now PRINTs (using the Kernal PRINT routine at $FFD2) instead of POKEing. It's necessary to turn reverse on and off repeatedly to get all the quarter squares, which is a little cumbersome. But there are some major advantages to sending everything through $FFD2.

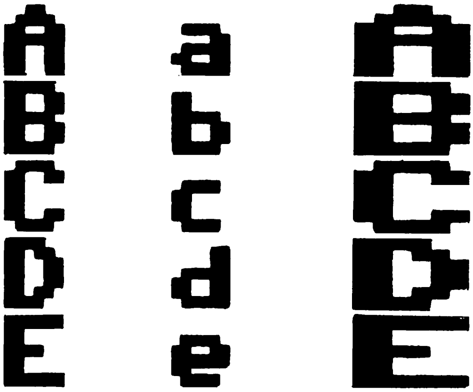
The first advantage is that you can send large characters to a Commodore printer, although you need to change one value to print spaces instead of cursor-rights (see line 951 of the program listing at the end of the article). Enter this to make a printout:

OPEN 4,4: CMD4: POKE 249,*xx*: POKE 250,*yy*: SYS 828

Remember to replace *xx* with the location where you want to print, and substitute the screen code for *yy*. If you can, adjust your printer's line spacing to zero—so there's no extra space between the characters. When you're finished printing, PRINT#4:CLOSE4 properly closes the file to the printer. Unfortunately, printers do not allow cursor up movements; you're limited to one large character per line. To get around this limitation, you could manually move the paper back, or use a screen dump program, or (if you're feeling ambitious) use CMD to send output to a tape or disk file and then read the data back into an array for dumping to the printer.

The figure below illustrates some of the large characters created by a Commodore printer. The first two columns show upper- and lowercase letters, while the third column shows the large letters expanded by the printer.

Normal and Expanded Samples



Another bonus of PRINTing rather than POKEing is that Screen Headliner is completely compatible with "Screen-80" (the 80-column program which appears in the article immediately preceding this); you can use large letters, up to 19 per line, in combination with 80-column text on your Commodore 64.

Finally, the flexibility of the PRINT command is at your fingertips: You can print almost anywhere on the screen, in any color you like (just change the cursor color). You can even mix large uppercase, lowercase, and graphics characters on the same screen.

A slight drawback is that each line has to be followed by a carriage return, which means you cannot put a character at the right edge of the screen. Nor can you print the large character at the bottom of the screen (it always scrolls up one line).

How It Works

There are two sets of POKEs in the BASIC loader program. The first loop (688 to 703) contains the modified ASCII values of the quarter-square graphics characters. Since there is no such thing as an ASCII value of a reversed character, the reverse flag has to be turned on and off. Bit 6 of each character is used to signal whether or not the character is reversed; the number is then ANDed with $BF (191) to turn off bit 6 before the character is printed.

The second loop (828 to 1006) is the machine language routine. It goes into the cassette buffer, but is written to be relocatable—if you need the cassette buffer for another ML program, or if you are using a Datassette, you can move the routine anywhere else in memory (the first loop has to stay where it is, however). If you put it in BASIC RAM, you'll have to protect it from being overwritten.

If you're interested in machine language, here's a brief explanation of how Headliner works. The main routine first checks which character set is being used and sets a zero page pointer accordingly. The screen code number is then multiplied by eight and added to the pointer. Once the pointer is set, the bytes from character ROM are loaded in two by two. By alternately shifting left the bytes (ASL) and rotating left the accumulator (ROL), a number from 0 to 15 is generated. This is used as an offset to look up the appropriate quarter-square graphics character in the table at 688. Bit 6 is checked (if set, reverse is turned on), and finally, a JSR to $FFD2 prints the character. The program then loops back to get the next set of bits.

Screen Headliner

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C.

```
5 PRINT"{CLR}PLEASE WAIT A MOMENT"          :rem 153
1Ø T=Ø:FORJ=688TO7Ø3:READK:T=T+K:POKEJ,K:NEXT
                                            :rem 134
15 IFT<>3Ø78THENPRINT"ERROR IN DATA STATEMENTS":ST
   OP                                       :rem 88
2Ø T=Ø:FORJ=828TO1ØØ6:READK:T=T+K:POKEJ,K:NEXT
                                            :rem 176
25 IFT<>2Ø3Ø6THENPRINT"ERROR IN DATA STATEMENTS":S
   TOP                                      :rem 13Ø
3Ø POKE249,Ø                                :rem 141
688 DATA32,188,19Ø,226,172,225,191,251      :rem 148
696 DATA187,255,161,236,162,254,252,96      :rem 158
828 DATA 169,2Ø8,133,ØØ4,173,Ø24             :rem 46
834 DATA 2Ø8,Ø41,ØØ2,24Ø,ØØ4,169             :rem 32
84Ø DATA 216,133,ØØ4,169,ØØØ,162             :rem 31
846 DATA ØØ3,ØØ6,25Ø,Ø42,2Ø2,2Ø8             :rem 28
852 DATA 25Ø,Ø24,1Ø1,ØØ4,133,ØØ4             :rem 19
858 DATA 165,25Ø,133,ØØ3,173,Ø14             :rem 4Ø
864 DATA 22Ø,Ø41,254,141,Ø14,22Ø             :rem 27
87Ø DATA 165,ØØ1,Ø41,251,133,ØØ1             :rem 23
876 DATA 169,ØØØ,133,25Ø,169,ØØ5             :rem 46
882 DATA 133,ØØ2,16Ø,ØØØ,177,ØØ3             :rem 26
888 DATA 133,ØØ5,23Ø,ØØ3,177,ØØ3             :rem 36
894 DATA 133,ØØ6,23Ø,ØØ3,198,ØØ2             :rem 36
9ØØ DATA 24Ø,Ø28,162,ØØ4,169,ØØØ             :rem 28
9Ø6 DATA ØØ6,ØØ6,Ø42,ØØ6,ØØ6,Ø42             :rem 25
912 DATA ØØ6,ØØ5,Ø42,ØØ6,ØØ5,Ø42             :rem 2Ø
```

```
918 DATA 164,250,153,048,002,230              :rem 38
924 DATA 250,202,208,232,240,210              :rem 26
930 DATA 165,001,009,004,133,001              :rem 20
936 DATA 173,014,220,009,001,141              :rem 28
942 DATA 014,220,160,000,166,249              :rem 33
948 DATA 240,008,169                         :rem 229
951 DATA 029:REM 032 IF USING A PRINTER      :rem 129
952 DATA 032,210                              :rem 14
954 DATA 255,202,208,250,169,004              :rem 45
960 DATA 133,006,185,048,002,170              :rem 38
966 DATA 189,176,002,133,005,041              :rem 46
972 DATA 064,240,005,169,018,032              :rem 43
978 DATA 210,255,165,005,041,191              :rem 46
984 DATA 032,210,255,169,146,032              :rem 47
990 DATA 210,255,200,198,006,208              :rem 43
996 DATA 221,169,013,032,210,255              :rem 43
1002 DATA 192,016,208,196,096                :rem 153
```

Reversi

Keith Day

This nineteenth-century game of strategy can be learned in minutes, but becoming an expert at it is another matter. You can play against another person or against the computer. You can even sit back and watch the computer wage a strategic battle against itself. One joystick required.

Reversi, originally a board game for two players, was first published in London about 1888. It's as popular today as it was then. In fact, national and international competitions are held each year where thousands of players compete for fame and glory.

The attraction of Reversi is that, although the rules are few and easy to learn, and play is very simple, the strategy and thought that go into a game can be quite involved. And this computer version makes the rules even easier to learn. The computer just won't let you break them! Illegal moves are not allowed; it's as simple as that. If you don't know if a move is legal, simply try it. If it's allowed, the computer executes it. If not, nothing happens. The question of which move is best, however, is left entirely up to you.

One or Two Players

Type in and save the game, using "The Automatic Proofreader" program you'll find in Appendix C. The Proofreader makes it almost impossible to mistype "Reversi."

After loading the game program from tape or disk, enter RUN. The screen clears and you're asked if you want to play against the computer. Answer *Y* for a computer opponent or *N* to play against another person. If you want to watch the computer play against itself, press the C key instead.

As soon as you select your opponent, you'll see the game screen. It's divided into 64 squares—an 8 × 8 grid. The object of Reversi is to strategically place discs on the squares so that more discs of your color are on the screen at the end of the game than your opponent's color.

Flipping Discs

Black always moves first. The moves are made by using a joystick plugged into port 1. If you're playing another person, you'll have to share the joystick. When the computer plays against itself, of course, you don't need to plug in a joystick.

The first four discs are automatically placed on the screen at the beginning of the game. You place your discs by moving the cursor (black or white) to the square of your choice and then pressing the fire button. The computer will allow only legal moves. This means that at least one of your opponent's discs must be "outflanked" as a result.

After you place your disc, the fun begins. The computer flips (reverses the color) all your opponent's discs that have been outflanked. (Outflanked discs are those that lie in a line between the disc just placed and another disc of the same color, so long as there is not a break in the line.) A disc may outflank any number of opposing pieces in vertical, horizontal, or diagonal lines.

If you're unable to set a piece anywhere on the screen, you have to forfeit the turn. That's done by pressing the space bar. You can tell there are no legal moves available when the computer won't place a disc no matter what square you have the cursor on. If you're playing the computer, and it's the computer's turn, it will hand over control to you if it has no move.

White's score is displayed in the upper left corner of the screen, while black's is in the upper right corner. The score is updated after each turn.

At the end of the game, the computer will ask if you want to play again. Enter *Y* to play again or *N* to exit the program.

The Program

Even though this program has a very graphic "feel," it was written entirely without the help of sprite graphics or other special graphics routines. Only the graphic characters found on the Commodore keyboard are used. In fact, the program is a very good demonstration of what can be done in BASIC with just the keyboard graphics and a little imagination.

A quick inspection of the program also reveals that the GOSUB command is used extensively. That's because the logic of the game seems to be an exercise in doing the same thing over and over, with only slight changes. For example, only one subroutine or section of the program is used to flip the discs. Variables are preset to indicate which disc is to be flipped and which color to flip it to. Once these variables are set, a subroutine is called with the GOSUB command and the disc changes color. The same single subroutine logic is also applied to performing the other tasks in the program, such as moving the cursor or reading the joystick.

Playing Tips

The best squares to occupy are the edges, since they can't be outflanked on all sides. The corners, in fact, seem to be the best squares to take, because they can't be outflanked from any direction. The best strategy, therefore, seems to be one of controlling the corners and edges. Be careful, though; even when you are way ahead, a few moves by your opponent can dramatically change the outcome of the game.

That's why Reversi is such a popular game. It's fast, enjoyable, and full of changes in fortune. And since the computer does the tedious work of flipping the discs, you can concentrate on strategy, working toward that ingenious move to turn several discs at once.

Reversi

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C.

```
100 DATA 1,8,7,1,8,7,1,8,7,4,5,1              :rem 9
110 DATA 4,5,1,1,8,7,1,8,7,3,6,3              :rem 6
120 DATA 3,6,3,1,8,7,3,6,3,3,6,1              :rem 3
130 DATA 4,5,1,3,6,3,1,8,7,2,7,5              :rem 6
140 DATA 2,7,5,1,8,7,3,6,1,2,7,5              :rem 9
150 DATA 2,7,5,3,6,1,2,7,5,2,7,5,0          :rem 100
160 DIM G(10,9):GOTO 1240                     :rem 9
170 POKE S1+X+40*Y,P:POKE C1+X+40*Y,C:RETURN
                                             :rem 87
180 P=98:C4=C:V1=X:V2=Y:X=X*4+1:Y=Y*3-2      :rem 60
190 P3=PEEK(S1+X+40*Y):C5=PEEK(C1+X+40*Y):GOSUB 17
    0                                       :rem 169
200 Y=Y+1:P=P+128                           :rem 186
210 P4=PEEK(S1+X+40*Y):C6=PEEK(C1+X+40*Y):GOSUB 17
    0                                       :rem 164
220 X=X1*4+1:Y=Y1*3-2                       :rem 130
230 P=P1:C=C2:GOSUB 170                      :rem 36
240 Y=Y+1:P=P2:C=C3:GOSUB 170               :rem 172
```

```
250 X1=V1:Y1=V2:P1=P3:P2=P4:C2=C5:C3=C6:X=V1:Y=V2:
    C=C4:RETURN                                   :rem 40
260 V1=X:V2=Y:X=X*4:Y=Y*3-2:P=233:GOSUB 170:rem 39
270 X=X+1:P=224:GOSUB 170                        :rem 147
280 X=X+1:P=223:GOSUB 170                        :rem 147
290 X=X-2:Y=Y+1:P=95 :GOSUB 170                  :rem 243
300 X=X+1:P=224:GOSUB 170                        :rem 141
310 X=X+1:P=105:GOSUB 170                        :rem 140
320 X=V1:Y=V2                                     :rem 9
330 IF C=1 THEN WS=WS+1:G(X,Y)=1                 :rem 15
340 IF C=0 THEN BS=BS+1:G(X,Y)=-1                :rem 18
350 IF FL=1 AND C=1 THEN BS=BS-1                 :rem 159
360 IF FL=1 AND C=0 THEN WS=WS-1                 :rem 201
370 RETURN                                       :rem 122
380 PRINT"{HOME}{DOWN}{3 SPACES}{3 LEFT}{WHT}";WS:
                                                 :rem 139
390 PRINT"{HOME}{DOWN}";TAB(36);"{3 SPACES}
    {3 LEFT}{BLK}";BS:RETURN                      :rem 45
400 IFBS+WS=64ORBS=0ORWS=0THEN1150               :rem 133
410 IF C=0 OR{2 SPACES}CO$="N"THEN GOSUB 450
                                                  :rem 66
420 IF C=1 AND CO$="Y"THEN GOSUB 870             :rem 135
430 IF CO$="C"THEN GOSUB 870                     :rem 238
440 GOTO 400                                     :rem 101
450 JV=PEEK(56321):FR=JVAND16:JV=15-(JVAND15)
                                                 :rem 170
460 IF JV=0 THEN 510                             :rem 251
470 IF JV=1 THEN Y=Y-1:GOTO 540                  :rem 192
480 IF JV=2 THEN Y=Y+1:GOTO 540                  :rem 192
490 IF JV=4 THEN X=X-1:GOTO 540                  :rem 195
500 IF JV=8 THEN X=X+1:GOTO 540                  :rem 189
510 X$="":GETX$:IF X$=" "THEN GOSUB 700           :rem 53
520 IF FR<>16ANDG(X,Y)=0 THEN GOSUB 590          :rem 161
530 RETURN                                       :rem 120
540 IF Y<1 THEN Y=8                              :rem 235
550 IF Y>8 THEN Y=1                              :rem 238
560 IF X<0 THEN X=9                              :rem 235
570 IF X>9 THEN X=0                              :rem 238
580 GOSUB 180:RETURN                             :rem 208
590 HX=X:HY=Y:OK=0                               :rem 133
600 XD=-1:YD=-1:GOSUB 740                        :rem 148
610 XD=0:GOSUB 740                               :rem 245
620 XD=1:GOSUB 740                               :rem 247
630 YD=0:GOSUB 740                               :rem 248
640 YD=1:GOSUB 740                               :rem 250
650 XD=0:GOSUB 740                               :rem 249
660 XD=-1:GOSUB 740                               :rem 40
670 YD=0:GOSUB 740                               :rem 252
680 IF OK=0 THEN RETURN                           :rem 67
```

```
690 GOSUB 380:P1=160:P2=160:GOSUB 700:RETURN
                                              :rem 68
700 IF G(X,Y)=1 THEN C2=1:C3=1               :rem 57
710 IF G(X,Y)=-1THEN C2=0:C3=0               :rem 101
720 IF TN=-1THEN TN= 1:C=1:X=0:Y=3:GOSUB 180:RETUR
    N                                         :rem 254
730 IF TN= 1THEN TN=-1:C=0:X=9:Y=3:GOSUB 180:RETUR
    N                                         :rem 7
740 FL=0:BR=0                                 :rem 213
750 IF X<1 OR X>8 THEN GOTO 820               :rem 97
760 IF Y<1 OR Y>8 THEN GOTO 820               :rem 100
770 X=X+XD:Y=Y+YD                             :rem 67
780 IF FL=1 AND G(X,Y)=-TN THEN GOSUB 260:GOTO 770
                                              :rem 218
790 IF G(X,Y)=-TN THEN BR=1:GOTO 770          :rem 242
800 IF G(X,Y)=TN AND BR=1 AND OK=0 THEN GOSUB 830
                                              :rem 116
810 IF G(X,Y)=TN AND BR=1 THEN OK=1:FL=1:BR=2:X=HX
    :Y=HY:GOTO 770                            :rem 240
820 X=HX:Y=HY:RETURN                          :rem 90
830 SX=X:SY=Y:X=HX:Y=HY                       :rem 55
840 IF C=1 AND CO$="Y"THEN GOSUB 1030         :rem 178
850 IF CO$="C"THEN GOSUB 1030                 :rem 25
860 GOSUB 260:X=SX:Y=SY:RETURN                :rem 198
870 OK=0:Z=0:RESTORE                          :rem 5
880 IF(WS+BS)>8{2 SPACES}THEN 900             :rem 40
890 FOR TM=1 TO 30:READ Q:NEXT                :rem 189
900 READ N1                                   :rem 52
910 IF N1=0 THEN GOSUB 720:RETURN             :rem 119
920 READ N2,N3,N4,N5,N6                       :rem 241
930 IF RND(0)>.5 THEN SW=N1:N1=N2:N2=SW:N3=-N3
                                              :rem 131
940 FOR Y=N1 TO N2 STEP N3                    :rem 121
950 IF RND(0)>.5 THEN SW=N4:N4=N5:N5=SW:N6=-N6
                                              :rem 151
960 FOR X=N4 TO N5 STEP N6                    :rem 131
970 REM:GOSUB 180                             :rem 215
980 IF G(X,Y)=0 THEN GOSUB 590                :rem 95
990 IF OK=1 THEN GOTO 1020                    :rem 100
1000 NEXT:NEXT                                :rem 121
1010 GOTO 900                                 :rem 148
1020 RETURN                                   :rem 163
1030 Q2=X:R2=Y:Y=Y1                           :rem 138
1040 IFQ2-X1<>0THENXA=(Q2-X1)/ABS(Q2-X1)      :rem 85
1050 IF C=1ANDQ2=1 THEN GOTO 1090             :rem 252
1060 IF C=0ANDQ2=8 THEN GOTO 1090             :rem 3
1070 X3=X1+XA                                 :rem 221
1080 FORX=X3TOQ2-XASTEPXA:GOSUB180:NEXT       :rem 93
1090 IF Y1=R2 THEN GOSUB 180:GOTO1120         :rem 35
1100 YA=(R2-Y1)/ABS(R2-Y1)                    :rem 182
```

```
1110 FOR Y=Y1TOR2STEPYA:GOSUB 180:NEXT     :rem 147
1120 X=Q2:Y=R2                              :rem 48
1130 FOR TM=1 TO 400:NEXT TM                :rem 12
1140 RETURN                                 :rem 166
1150 PRINT"{HOME}{7}{3 RIGHT}";             :rem 215
1160 IF CO$="Y" AND WS<BS THEN PRINT"THAT WAS TOUG
     H.";:GOTO 1210                         :rem 120
1170 IF CO$="Y" AND WS>BS THEN PRINT"THAT WAS A BR
     EEZE.";:GOTO 1210                      :rem 242
1180 IF BS>WS THENPRINT"BLACK WINS.";       :rem 221
1190 IF WS>BS THENPRINT"WHITE WINS.";       :rem 2
1200 IF BS=WS THENPRINT"A TIE! A TIE!";     :rem 145
1210 PRINT" PLAY AGAIN? Y/N";               :rem 123
1220 GETX$:IFX$<>"Y"ANDX$<>"N"THEN 1220     :rem 204
1230 IF X$="N" THEN PRINT"{CLR}{HOME}";:END
                                            :rem 146
1240 X1=0:Y1=0:BS=0:WS=0:S1=1024:C1=55296:C=14:FL=
     0:PRINT"{CLR}{HOME}"                   :rem 60
1250 FOR X=1 TO 10:FOR Y=1 TO 9             :rem 61
1260 G(X,Y)=0                               :rem 171
1270 NEXT:NEXT                              :rem 130
1280 PRINT"PLAY AGAINST COMPUTER?{2 SPACES}Y/N "
                                            :rem 93
1290 GET CO$                                :rem 98
1300 IF CO$<>"Y"ANDCO$<>"N"ANDCO$<>"C" THEN GOTO 1
     290                                    :rem 115
1310 PRINT"{CLR}{HOME}";                    :rem 119
1320 PRINT"{3 SPACES}{A}***{R}***{R}***{R}***{R}**
     *{R}***{R}***{R}***{S}"                :rem 211
1330 FOR X=1 TO 8                           :rem 79
1340 PRINT"{3 SPACES}-{3 SPACES}-{3 SPACES}-
     {3 SPACES}-{3 SPACES}-{3 SPACES}-{3 SPACES}-
     {3 SPACES}-{3 SPACES}-"                :rem 94
1350 PRINT"{3 SPACES}-{3 SPACES}-{3 SPACES}-
     {3 SPACES}-{3 SPACES}-{3 SPACES}-{3 SPACES}-
     {3 SPACES}-{3 SPACES}-"                :rem 95
1360 IF X<8 THEN PRINT"{3 SPACES}{Q}***+***+***+**
     *+***+***+***+***{W}"                  :rem 128
1370 NEXT X                                 :rem 98
1380 PRINT"{3 SPACES}{Z}***{E}***{E}***{E}***{E}**
     *{E}***{E}***{E}***{X}";               :rem 25
1390 X=11:Y=6:P=87:GOSUB 170                :rem 82
1400 X=27:GOSUB 170                         :rem 21
1410 Y=18:GOSUB 170                         :rem 23
1420 X=11:GOSUB 170                         :rem 16
1430 Y=6:GOSUB 170                          :rem 230
1440 X=4:Y=4:C=1:GOSUB 260                  :rem 211
1450 X=5:Y=4:C=0:GOSUB 260                  :rem 212
```

```
1460 X=5:Y=5:C=1:GOSUB 260                    :rem 215
1470 X=4:Y=5:C=0:GOSUB 260                    :rem 214
1480 GOSUB 380                                :rem 232
1490 P1=96:P2=96:C2=6:C3=6                    :rem 168
1500 X=9:Y=3:P=98:C=0:TN=-1:GOSUB 180         :rem 131
1510 GOTO 400                                 :rem 148
```

Family Tree

Mark Haney

Your computer is the perfect tool for keeping records. Storing and retrieving information, displaying it on the screen (or on paper), and letting you easily change the data are some of the most efficient uses of your Commodore 64. And genealogy is just a mass of information: names, dates, and relationships. With "Family Tree," you can use the 64's record-keeping power to trace your family's roots. For tape or disk users.

Have you ever tried to create a family tree? Usually, you have to create a diagram-like chart and then write each name down in the proper blank. Adding or changing the chart can be almost impossible without redoing it all. That's one of the disadvantages of paper and pencil.

Your Commodore 64 can help you trace your ancestors, without all the trouble of constantly redrawing charts. "Family Tree," a sophisticated record-keeping program for the 64, lets you enter names, dates of birth, and relationships. You can save the information to tape or disk, allowing you access to your genealogy at any time. Changing or deleting entries is done with a keypress or two. Adding more names is just as simple. And you can even create a copy of the chart if you have a printer. (If you have a Commodore MPS 801 printer, see page 92.)

A family tree is a very personal piece of history. Your father's version neglects half your heritage, your son's includes people of only academic interest to you. Some first cousins you see several times in a year, others you may never recall meeting.

This is not to say that you would wish to purge anyone from your family tree. But given the limitations of printed genealogy charts, it's difficult to make any sense of a document containing anything more than perhaps a few score of names. And the task of maintaining or copying such a record is formidable indeed.

Tracing

Family Tree has two functions, maintenance and display, that operate together at all times. Storage and retrieval are taken care of by the LOAD and SAVE commands, as the program is self-modifying.

Type in and save Family Tree to tape or disk. It's much easier to enter the program if you use "The Automatic Proofreader," found in Appendix C. You can insure an error-free copy of the program if you use the Proofreader.

When you first run Family Tree, you'll see a screen with instructions. The letters at the top indicate keys to press when you create the chart. We'll talk about them in a moment. After a short wait while the program initializes variables, you'll be asked to enter the filename of the tree you want to display. If this is your first time using Family Tree, then just hit the RETURN key. Press any key and the initial entry message appears at the bottom of the screen.

This is where you start. It may be worthwhile, before you begin, that you have an idea of how you're going to trace your genealogy. Begin with your name, and then work backwards to your distant ancestors? Or start with a great-great-great-grandmother and work towards your closer relatives?

Whatever you decide, type in the initial entry. First name, last name, and birth year need to be separated with commas. If you don't know the year of birth, you can leave it out, but you still need the comma after the last name. Later, when you determine the birth year, you can return to the entry and put it in.

As soon as you hit the RETURN key, the screen scrolls up and a shortened version of the name appears in the middle left of the display. There should be two large blinking cursors bracketing the name. At the bottom, in reverse video, will appear the full name, as well as birth year.

(When you've created a family tree file, saved it, then later loaded it back into memory, the bracketed name and full entry is what you'll first see on the screen.)

Relative Spots

The *current person* is noted by the cursor. Now you're ready to enter and display relatives of the current person by pressing the following keys:

Key	Relative
M	Mother
F	Father
S	Spouse
P	Previous sibling
N	Next sibling
C	Child

After the initial entry appears, then you can type in that person's relatives by pressing one of the above keys. Hit the *M* key, for instance, and enter the current person's mother's name and birth year. Take care that you place commas between the three items. (If you suddenly decide you don't want to make an entry, hit the RETURN key and the cursors move back to the last entry.) The mother then becomes the current person, as indicated by the cursors. You can continue to enter more names and birth years in this way.

Backtracking, say to the initial entry, can be done in one of two ways. You can use the cursor keys on the Commodore keyboard to move the blinking cursors to that name. Or you can press the correct key from the table above. Let's say you have three names on the screen: the initial entry, and his or her mother and father. To move back to the initial entry, assuming the cursor is on the father's name, all you have to do is press the C (for Child) key and the cursors return to the first name.

Don't worry about going off the screen as you enter several names. The display moves as necessary.

Existing Trees

If you've already created and saved a family tree file, and then want to modify it later, all you have to do is specify the filename when you see the first screen display. Make sure the disk or tape with the file is in the drive or cassette, and type in the file's name. You have to specify tape or disk by entering *T* or *D*. The file will load into memory and you'll see only the initial entry on the screen. Don't worry, the rest is there. You just have to hit a few keys to display it.

The initial entry is on the screen. What now? Just press the correct keys from the Relative table and the names appear in the chart. For example, if you earlier entered the initial entry, plus that person's father, mother, and spouse, pressing *F*, *M*, and *S* (interrupted with some cursor movements) makes the three names display.

Saving and Loading Trees

Once you've created a tree that you want to save, just press *Q*. If you change your mind, you can hit the RETURN key and the screen appears as before. Press the *Y* key, however, and a prompt asking for a new filename shows at the bottom of the screen. *You can save out a tree only if you've made changes.* That's logical—why else would you want to save a file?

As already discussed, to load a previously created tree file, all you have to do is enter its name in the first screen display. It will load as soon as you've pressed T or D.

More Keys

Several other useful keys are:

Key	Function
CLR	Clear screen, leaving only current person
DEL	Delete current person from display
f1	Delete entry from tree
O	Output current display to printer
Q	Quit
D	Change data for current person

The cursor controls operate normally.

Quit

When you decide to quit the program and hit the *Q* key, there are two possibilities: The tree has merely been displayed, or changes have been made. In the latter case, new DATA statements must be created. It's essential that you wait for this process to occur and save the program after its completion. Otherwise, no record will be made of your changes.

I've used Family Tree to create a tree of 360 names. One suggestion for larger families is the creation of separate trees representing different family branches. An estimate of memory requirements is given by 8000+77*n for n entries. This depends on name length and whether birth years are in all cases included.

Family Tree

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C.

```
1 PRINTCHR$(147)"{12 RIGHT}THE FAMILY TREE":PRINT
                                             :rem 230
2 PRINT"M--MOTHER":PRINT"F--FATHER":PRINT"S--SPOUS
  E":PRINT"P--PREVIOUS SIBLING"              :rem 175
3 PRINT"N--NEXT SIBLING":PRINT"C--CHILD":PRINT
                                             :rem 198
4 PRINT"[CLR]--CLEAR SCREEN":PRINT"[DEL]--DELETE E
  NTRY FROM SCREEN"                          :rem 74
5 PRINT"[F1]--DELETE ENTRY FROM TREE"        :rem 56
6 PRINT"O--OUTPUT SCREEN":PRINT"Q--QUIT":PRINT"D--
  CHANGE DATA"                               :rem 195
7 PRINT"CURSOR CONTROLS NORMAL":PRINT        :rem 234
9 PRINT"PLEASE WAIT FOR INITIALIZATION":GOTO400
                                              :rem 15
10 GOSUB500:IFA$=""THEN10                    :rem 181
12 GOSUB550                                  :rem 125
15 ONJGOSUB110,120,130,140,150,160,170,180,190,200
   ,210,220,230,240,250,260                  :rem 186
20 GOTO10                                    :rem 252
110 IFY<MYTHENY=Y+1:N=SC(X,Y):GOSUB650       :rem 180
111 RETURN                                   :rem 115
120 IFY>1THENY=Y-1:N=SC(X,Y):GOSUB650         :rem 68
121 RETURN                                   :rem 116
130 IFX<MXTHENX=X+1:N=SC(X,Y):GOSUB650       :rem 178
131 RETURN                                   :rem 117
140 IFX>1THENX=X-1:N=SC(X,Y):GOSUB650         :rem 67
141 RETURN                                   :rem 118
150 M1=1:T1=1:T2=2:GOSUB850:RETURN            :rem 79
160 M1=2:T1=2:T2=1:GOSUB850:RETURN            :rem 81
170 M1=3:T1=3:T2=1:GOSUB800:RETURN            :rem 79
180 M1=6:T1=4:T2=4:GOSUB800:RETURN            :rem 87
190 M1=4:T1=5:T2=3:GOSUB800:RETURN            :rem 86
200 IFN=0THENRETURN                          :rem 235
201 N=FT%(SC(X,Y),6):IFN=0THENM1=5:GOSUB950:IFN=0T
    HENN=LN:GOSUB650:RETURN                  :rem 150
202 IFOS%(N)<>0THENX=INT(OS%(N)/10):Y=OS%(N)-10*X:
    GOSUB650:RETURN                          :rem 222
203 TN=FT%(LN,6)                             :rem 128
204 IFOS%(TN)<>0THENX=INT(OS%(TN)/10):Y=OS%(TN)-10
    *X:N=TN:GOSUB650:RETURN                   :rem 67
```

```
205 IFFT%(TN,5)<>0THENTN=FT%(TN,5):GOTO204  :rem 13
206 IFX=1THENM1=4:GOSUB700:D=-2:GOSUB600:RETURN
                                           :rem 224
207 IFSC(X-1,Y)<>0THENGOSUB900:RETURN       :rem 86
208 X=X-1:D=-2:GOSUB600:RETURN             :rem 105
210 FORJ=1TOMY:FORK=1TOMX:SC(K,J)=0:NEXT:NEXT
                                           :rem 222
211 FORJ=1TOMN:OS%(J)=0:NEXT               :rem 243
212 PRINTCHR$(19);:FORJ=1TONR:PRINTBL$:NEXT:rem 39
213 X=1:Y=4:D=0:GOSUB600:RETURN            :rem 181
220 GOSUB350:PRINTD$:A$="":INPUT"QUIT/Y, NO/CR";A$
    :IFA$="Y"THEN222                       :rem 206
221 N=LN:GOSUB650:RETURN                    :rem 41
222 IFFL=0THENGOSUB350:PRINTD$:END         :rem 171
223 GOSUB350:PRINTD$:PRINT"CHANGES HAVE BEEN MADE"
                                            :rem 55
224 INPUT"NEW FILE NAME";N$:INPUT"TAPE OR DISK";A$
                                           :rem 232
225 IFLEFT$(A$,1)="T"THENOPEN1,1,1,N$:GOTO227
                                            :rem 87
226 OPEN1,8,2,N$+",SEQ,W"                  :rem 100
227 PRINT#1,MN:FORJ=1TOMN:PRINT#1,N$(J,0)R$N$(J,1)
                                           :rem 234
228 FORK=1TO6:PRINT#1,FT%(J,K):NEXT:PRINT#1,DT%(J)
    :NEXT:CLOSE1:END                       :rem 176
230 GOSUB350:PRINTD$:A$="":INPUT"OUTPUT/O OR CR";A
    $:IFA$=""THENGOSUB350:RETURN            :rem 95
231 OPEN4,4,4:CMD4:PRINTCHR$(27);CHR$(109);CHR$(4)
    ;                                      :rem 163
232 FORJ=0TONR-3:FORK=0TONC-2:T1=PEEK(SC+NC*J+K)
                                           :rem 242
233 IFT1=64THENPRINTCHR$(133);:GOTO239      :rem 68
234 IFT1=93THENPRINTCHR$(134);:GOTO239      :rem 72
235 IFT1=107THENPRINTCHR$(132);:GOTO239    :rem 115
236 IFT1=115THENPRINTCHR$(131);:GOTO239    :rem 114
237 IFT1=32THENPRINTCHR$(32);:GOTO239       :rem 17
238 PRINTCHR$(T1+64);                      :rem 209
239 NEXT:PRINT:NEXT:PRINT#4:CLOSE4:RETURN   :rem 57
240 FL=1:M1=8:GOSUB785:IFN1$=""THENRETURN   :rem 29
241 GOSUB775:GOSUB650:RETURN                :rem 41
250 TP=SC+X*8+Y*120-169:FORJ=0TO3:FORK=0TO8:POKETP
    +J*40+K,32:NEXT:NEXT                   :rem 108
252 OS%(SC(X,Y))=0:SC(X,Y)=0:N=0:GOSUB650:RETURN
                                           :rem 177
260 GOSUB350:PRINTD$:A$="":INPUT"DELETE FROM TREE/
    Y, NO/CR";A$                           :rem 217
261 IFA$<>"Y"THENN=LN:GOSUB650:RETURN      :rem 103
262 FL=1:IFFT%(FT%(N,1),6)<>NANDFT%(FT%(N,2),6)<>N
    THEN264                                :rem 236
```

```
263 TN=FT%(N,5):FT%(FT%(N,1),6)=TN:FT%(FT%(N,2),6)
    =TN                                      :rem 198
264 FT%(FT%(N,4),5)=FT%(N,5):FT%(FT%(N,5),4)=FT%(N
    ,4):FT%(FT%(N,3),3)=Ø                    :rem 254
265 TN=FT%(N,6)                               :rem 6Ø
266 IFTN=ØTHENN=Ø:GOTO25Ø                     :rem 48
267 IFFT%(TN,1)=NTHENFT%(TN,1)=Ø:TN=FT%(TN,5):GOTO
    266                                      :rem 176
268 FT%(TN,2)=Ø:TN=FT%(TN,5):GOTO266          :rem 9Ø
35Ø PRINTD$:PRINTBL$:PRINTBL$CHR$(145):RETURN
                                             :rem 133
4ØØ POKE51,2ØØ:POKE55,2ØØ:POKE52,PEEK(52)-1:POKE56
    ,PEEK(56)-1:CLR                            :rem 9
4Ø1 X=1:Y=4:N=1:A$="":T1=Ø:T2=Ø:T3=Ø:T4=Ø:D=Ø:DR=Ø
    :M1=Ø:J=Ø:K=Ø:TN=Ø                       :rem 164
4Ø2 MX=5:MY=8:NR=25:NC=4Ø:LE=7:LM(1)=8ØØ:LM(2)=8ØØ
    :LM(3)=912:LM(4)=912                     :rem 225
4Ø3 SC=256*PEEK(648):MS=1ØØØ                  :rem 33
4Ø4 T5=255:T6=256:U8=128:P1=Ø:P2=Ø:P3=Ø:P4=Ø
                                             :rem 221
4Ø5 DIMFT%(MS,6),OS%(MS),DT%(MS),N$(MS,1),SC(MX+1,
    MY+1),OP$(16)                             :rem 24
4Ø6 FORJ=1TO8:READTP$(J):NEXT                 :rem 7Ø
4Ø7 DATAMOTHER,FATHER,SPOUSE,NEXT,CHILD,PREVIOUS,I
    NITIAL ENTRY,NEW DATA                     :rem 17
4Ø8 FORJ=1TO16:READOP$(J):NEXT               :rem 114
4Ø9 REM [DOWN],[UP],[RIGHT],[LEFT],,,,,,,[CLR],,,[
    DEL],[F1]                                :rem 164
41Ø DATA"{DOWN}","{UP}","{RIGHT}","{LEFT}",M,F,S,P
    ,N,C,"{CLR}",Q,O,D,"{DEL}","{F1}"         :rem 82
411 FR(1)=SC+LM(1):FR(2)=SC+3*NC:FR(3)=SC+LE+1:FR(
    4)=SC+LM(4)                              :rem 19Ø
415 TG(1)=SC+LM(1)+3*NC:TG(2)=SC:TG(3)=SC:TG(4)=SC
    +LM(4)+LE+1                              :rem 2Ø6
42Ø T1=PEEK(55)+T6*PEEK(56):FORJ=T1TOT1+45:READT2:
    POKEJ,T2:NEXT                            :rem 155
421 DATA16Ø,Ø,177,251,145,253,24,165,251,1Ø1,142,1
    33,251,165,252,1Ø1,143,133               :rem 183
422 DATA252,24,165,253,1Ø1,142,133,253,165,254,1Ø1
    ,143,133,254,165,14Ø,2Ø8                  :rem 95
423 DATA4,198,141,48,5,198,14Ø,24,144,211,96
                                             :rem 177
425 MN=Ø:N$(Ø,Ø)="EMPTY":N$(Ø,1)="SPOT"      :rem 141
426 PRINT:N$="":INPUT"FAMILY TREE FILE NAME
    {2 SPACES}(CR IF NONE)";N$               :rem 233
427 IFN$=""THEN433                           :rem 232
428 INPUT"TAPE OR DISK";A$                     :rem 8
429 IFLEFT$(A$,1)="T"THENOPEN1,1,Ø,N$:GOTO431
                                              :rem 89
```

```
430 OPEN1,8,2,N$+",SEQ,R"                              :rem 92
431 INPUT#1,MN:FORJ=1TOMN:INPUT#1,N$(J,Ø),N$(J,1)
                                                       :rem 163
432 FORK=1TO6:INPUT#1,FT%(J,K):NEXT:INPUT#1,DT%(J)
    :NEXT:CLOSE1                                       :rem 162
433 PRINT:PRINT"PRESS ANY KEY WHEN READY
    {1Ø SPACES}"                                       :rem 55
434 GETA$:IFA$=""THEN434                               :rem 89
435 IFMN=ØTHENM1=7:GOSUB95Ø                            :rem 172
436 D$=CHR$(19):FORJ=1TONR-3:D$=D$+CHR$(17):NEXT
                                                       :rem 31
437 BL$="":FORJ=1TONC-1:BL$=BL$+" ":NEXT:R$=CHR$(1
    3)                                                 :rem 162
438 CH$(1)=CHR$(125):CH$(2)=CHR$(96):CH$(3)=CHR$(1
    71):CH$(4)=CHR$(179)                               :rem 148
44Ø FORJ=1TONR:PRINTBL$:NEXT:GOSUB6ØØ:GOTO1Ø
                                                       :rem 144
5ØØ P1=SC+(X-1)*(LE+1)+(Y-1)*3*NC:P2=P1+LE-1:P3=P2
    +NC-LE+1:P4=P3+LE-1                                :rem 76
5Ø1 GETA$:IFA$<>""THENRETURN                           :rem 214
5Ø5 T1=PEEK(P1):T2=PEEK(P2):T3=PEEK(P3):T4=PEEK(P4
    ):GETA$:IFA$<>""THENRETURN                         :rem 178
51Ø POKEP1,T1+U8:POKEP2,T2+U8:POKEP3,T3+U8:POKEP4,
    T4+U8                                              :rem 18Ø
515 FORJ=1TO5Ø:GETA$:IFA$=""THENNEXT                   :rem 222
52Ø POKEP1,T1:POKEP2,T2:POKEP3,T3:POKEP4,T4
                                                       :rem 213
525 IFA$=""THENFORJ=1TO5Ø:GETA$:IFA$=""THENNEXT
                                                       :rem 131
53Ø RETURN                                             :rem 12Ø
55Ø FORJ=1TO16:IFA$<>OP$(J)THENNEXT                    :rem 183
551 RETURN                                             :rem 123
6ØØ GOSUB65Ø:PRINTLEFT$(D$,(Y-1)*3+1)TAB((LE+1)*(X
    -1))LEFT$(N$(N,Ø),LE)                              :rem 2Ø6
6Ø1 PRINTTAB((LE+1)*(X-1))LEFT$(N$(N,1),LE)
                                                       :rem 134
6Ø5 SC(X,Y)=N:OS%(N)=1Ø*X+Y                            :rem 46
61Ø IFABS(D)<>2THEN62Ø                                 :rem 4
615 PRINTLEFT$(D$,(Y-1)*3+2)TAB((LE+1)*(X+(D>Ø))-1
    )CH$(ABS(D));                                      :rem 138
616 PRINTCHR$(145)CHR$(157)CH$(ABS(D)):RETURN
                                                       :rem 138
62Ø PRINTLEFT$(D$,3*(Y-(D>Ø))-3)TAB((LE+1)*(X-1)+2
    )CH$(ABS(D))CH$(ABS(D))                            :rem 186
625 RETURN                                             :rem 125
65Ø GOSUB35Ø:LN=N:PRINTD$:PRINTCHR$(18)N$(N,Ø)"
    {2 SPACES}"N$(N,1)"{2 SPACES}"DT%(N):RETURN
                                                       :rem 156
7ØØ GOSUB35Ø:POKE14Ø,LM(M1)ANDT5:POKE141,LM(M1)/T6
                                                       :rem 233
```

```
701 POKE142,1:POKE143,Ø:IFFR(M1)<TG(M1)THENPOKE142
    ,T5:POKE143,T5                              :rem 78
702 POKE251,FR(M1)ANDT5:POKE252,FR(M1)/T6 :rem 157
703 POKE253,TG(M1)ANDT5:POKE254,TG(M1)/T6:SYS(PEEK
    (55)+T6*PEEK(56))                          :rem 21Ø
705 ONM1GOTO71Ø,715,72Ø,725                    :rem 224
71Ø FORJ=1TOMX:FORK=1TOMY-1:OS%(SC(J,K))=OS%(SC(J,
    K))+1:NEXTK                                 :rem 23
711 OS%(SC(J,MY))=Ø:NEXTJ                      :rem 228
712 FORJ=MYTO1STEP-1:FORK=1TOMX:SC(K,J)=SC(K,J-1):
    NEXT:NEXT                                   :rem 85
713 PRINTCHR$(19)BL$:PRINTBL$:PRINTBL$:RETURN
                                               :rem 162
715 FORJ=1TOMX:FORK=2TOMY:OS%(SC(J,K))=OS%(SC(J,K)
    )-1:NEXTK                                  :rem 193
716 OS%(SC(J,1))=Ø:NEXTJ                       :rem 116
717 FORJ=1TOMY:FORK=1TOMX:SC(K,J)=SC(K,J+1):NEXT:N
    EXT                                        :rem 19Ø
718 PRINTLEFT$(D$,3*(MY-1))BL$:PRINTBL$:PRINTBL$:R
    ETURN                                      :rem 2Ø9
72Ø FORJ=1TOMY:FORK=2TOMX:OS%(SC(K,J))=OS%(SC(K,J)
    )-1Ø:NEXTK                                 :rem 237
721 OS%(SC(1,J))=Ø:NEXTJ                       :rem 112
722 FORJ=1TOMY:FORK=1TOMX:SC(K,J)=SC(K+1,J):NEXT:N
    EXT                                        :rem 186
723 PRINTCHR$(19);:FORJ=1TO3*MY-1:PRINTTAB(NC-LE-1
    )LEFT$(BL$,LE):NEXT                         :rem 33
724 FORJ=SC+NC-1TOSC+NC-1+NR*NCSTEPNC:POKEJ,32:NEX
    T:RETURN                                    :rem 56
725 FORJ=1TOMY:FORK=1TOMX-1:OS%(SC(K,J))=OS%(SC(K,
    J))+1Ø:NEXTK                                :rem 77
726 OS%(SC(MX,J))=Ø:NEXTJ                      :rem 233
727 FORJ=1TOMY:FORK=MXTO1STEP-1:SC(K,J)=SC(K-1,J):
    NEXT:NEXT                                   :rem 91
728 PRINTCHR$(19);:FORJ=1TO3*MY-1:PRINTLEFT$(BL$,L
    E+1):NEXT                                  :rem 173
729 FORJ=SC+NC-1TOSC+NC-1+NR*NCSTEPNC:POKEJ,32:NEX
    T:RETURN                                    :rem 61
75Ø TN=LN:N=MN+1:MN=N:GOSUB775                 :rem 142
751 IFFT%(TN,5)<>ØTHENTN=FT%(TN,5):GOTO751 :rem 26
755 FT%(TN,A)=N:IFFT%(TN,4)<>ØTHENTN=FT%(TN,4):GOT
    O755                                         :rem 4
76Ø FT%(N,3)=FT%(TN,B):FT%(N,6)=TN:IFFT%(TN,B)<>ØT
    HENFT%(FT%(TN,B),3)=N                      :rem 1Ø8
761 RETURN                                     :rem 126
775 N$(N,Ø)=N1$:N$(N,1)=N2$:DT%(N)=DT%:RETURN
                                                :rem 35
785 GOSUB35Ø:PRINTD$:PRINT:PRINT"TYPE FIRST NAME,
    {SPACE}LAST NAME, BIRTH YEAR";              :rem 58
```

```
790 PRINTCHR$(145)CHR$(145):PRINTTP$(M1);:N1$="":D
    T%=Ø:INPUTN1$,N2$,DT%:RETURN                :rem 16Ø
8ØØ IFN=ØTHENRETURN                             :rem 241
8Ø2 N=FT%(SC(X,Y),T1):IFN=ØTHENGOSUB95Ø:IFN=ØTHENN
    =LN:GOSUB65Ø:RETURN                         :rem 194
8Ø5 IFOS%(N)<>ØTHENX=INT(OS%(N)/1Ø):Y=OS%(N)-X*1Ø:
    GOSUB65Ø:RETURN                             :rem 231
8Ø7 DR=1:IFT1=4ORT1=3ANDY<5THENDR=-1            :rem 245
81Ø IFSC(X,Y+DR)=ØAND(Y+DR)>ØAND(Y+DR)<MY+1THENY=Y
    +DR:D=T2*-DR:GOSUB6ØØ:RETURN                :rem 185
815 IFSC(X,Y-DR)=ØAND(Y-DR)>ØAND(Y-DR)<MY+1THENY=Y
    -DR:D=T2*DR:GOSUB6ØØ:RETURN                 :rem 153
82Ø IFY=1THENM1=1:GOSUB7ØØ:D=T2:GOSUB6ØØ:RETURN
                                                  :rem 7
825 IFY=8THENM1=2:GOSUB7ØØ:D=-T2:GOSUB6ØØ:RETURN
                                                 :rem 65
83Ø GOSUB9ØØ:RETURN                             :rem 2Ø6
85Ø IFN=ØTHENRETURN                             :rem 246
852 N=FT%(SC(X,Y),T1):IFN=ØTHENGOSUB95Ø:IFN=ØTHENN
    =LN:GOSUB65Ø:RETURN                         :rem 199
855 IFOS%(N)<>ØTHENX=INT(OS%(N)/1Ø):Y=OS%(N)-1Ø*X:
    GOSUB65Ø:RETURN                             :rem 236
86Ø TX=INT(OS%(FT%(SC(X,Y),T2))/1Ø):TY=OS%(FT%(SC(
    X,Y),T2))-1Ø*TX:IFTX=ØTHEN87Ø               :rem 199
865 IFTY=1ORTY=MYORSC(TX,TY-1)=ØORSC(TX,TY+1)=ØTHE
    NX=TX:Y=TY:GOSUB17Ø:RETURN                  :rem 133
87Ø IFX=MXTHENM1=3:GOSUB7ØØ:D=2:GOSUB6ØØ:RETURN
                                                 :rem 45
875 IFSC(X+1,Y)<>ØTHENGOSUB9ØØ:RETURN            :rem 95
88Ø X=X+1:D=2:GOSUB6ØØ:RETURN                    :rem 64
9ØØ GOSUB35Ø:PRINTD$CHR$(17)"INSUFFICIENT SPACE ON
     SCREEN"                                     :rem 46
9Ø5 PRINT"SHOULD CLEAR OR DELETE"CHR$(145):RETURN
                                                :rem 255
95Ø IFMN<MS-1THEN952                            :rem 2Ø9
951 GOSUB35Ø:PRINTD$:PRINT"INSUFFICIENT MEMORY":FO
    RJ=1TO1ØØØ:NEXT:RETURN                       :rem 33
952 GOSUB785:IFN1$=""THENRETURN                 :rem 192
955 FL=1:ONM1GOTO956,96Ø,965,97Ø,975,98Ø,99Ø
                                                :rem 168
956 A=1:B=2:GOSUB75Ø:RETURN                     :rem 174
96Ø A=2:B=1:GOSUB75Ø:RETURN                     :rem 169
965 TN=LN:N=MN+1:MN=N:GOSUB775                  :rem 15Ø
966 FT%(TN,3)=N:FT%(N,3)=TN:FT%(N,6)=FT%(TN,6):IFF
    T%(N,6)=ØTHENRETURN                         :rem 1Ø3
967 TN=FT%(N,6):A=1:IFFT%(FT%(N,6),2)=ØTHENA=2
                                                :rem 113
968 FT%(TN,A)=N:TN=FT%(TN,5):IFTN<>ØTHEN968
                                                :rem 1Ø4
969 RETURN                                      :rem 136
```

```
970 TN=LN:N=MN+1:MN=N:GOSUB775                        :rem 146
971 FT%(N,1)=FT%(TN,1):FT%(N,2)=FT%(TN,2):FT%(N,4)
    =TN:FT%(TN,5)=N:RETURN                            :rem 180
975 TN=LN:N=MN+1:MN=N:GOSUB775                        :rem 151
976 IFFT%(TN,6)<>0THENA=2+(FT%(FT%(TN,6),1)=TN):GO
    TO978                                             :rem 201
977 GOSUB350:PRINTD$CHR$(17)CHR$(17)"GENDER OF PAR
    ENT; MOTHER/1, FATHER/2";                         :rem 166
978 PRINTD$CHR$(17)N$(LN,0);:INPUTA:B=2+(A=2)
                                                      :rem 109
979 FT%(N,A)=TN:FT%(N,B)=FT%(TN,3):FT%(TN,6)=N:FT%
    (FT%(TN,3),6)=N:RETURN                            :rem 226
980 TN=LN:N=MN+1:MN=N:GOSUB775                        :rem 147
981 FT%(N,1)=FT%(TN,1):FT%(N,2)=FT%(TN,2):FT%(N,5)
    =TN:FT%(TN,4)=N                                   :rem 155
982 T3=FT%(TN,1):IFT3<>0THENFT%(T3,6)=N               :rem 35
983 T3=FT%(TN,2):IFT3<>0THENFT%(T3,6)=N               :rem 37
984 RETURN                                            :rem 133
990 MN=MN+1:N=MN:GOSUB775:RETURN                      :rem 72
```

NOTE:
In order for this program to work with a Commodore MPS 801 printer, change the following lines.

```
231 OPEN4,4:CMD4                                      :rem 158
233 IFT1=64THENPRINTCHR$(96);:GOTO239                 :rem 28
234 IFT1=93THENPRINTCHR$(125);:GOTO239                :rem 72
235 IFT1=107THENPRINTCHR$(171);:GOTO239               :rem 118
236 IFT1=115THENPRINTCHR$(179);:GOTO239               :rem 126
```

Supertank

Boris Litinsky

In this unusual action game, your goal is to prevent hostile tanks from crossing your territory. By aiming carefully and avoiding direct hits, you may make it to the rank of Marshal. Joystick required.

Your orders are to stop the enemy tanks. But in the back of your mind, you know there's more to it than just following orders. You know that if you don't stop them, they'll stop you. Your goal is simply to survive.

Your commander has been kind enough to give you a choice of three different tanks. The Tiger has strong armor, which is great for helping you survive, but it moves sluggishly. The T-34 has moderate armor and speed, but lacks firepower. If you choose the Sherman, you'll have a quick tank with a good cannon, but almost no defense (armor). If you're a new recruit, you might want to choose the T-34 because of its defensive capabilities. Advanced players who are sure of their abilities may prefer the Sherman, although a single hit by the enemy can be devastating.

Controlling the Tank

Using "The Automatic Proofreader" (found in Appendix C), type in and save "Supertank." Load and run it, and in a few moments you'll see the tank outfitting display. Notice the different strengths and weaknesses each tank has. Pressing the appropriate key (1 for Tiger, 2 for T-34, or 3 for a Sherman) begins the game.

Your tank quickly moves onto the battlefield. Enemy tanks move across the screen from left to right. Using a joystick plugged into port 1, you can control the crosshairs of your cannon. Get the enemy tank in your sights and press the fire button to fire a salvo. Remember that it takes time for the shots to travel; you'll have to adjust accordingly, shooting slightly in front of your target.

After pressing the fire button, you'll see the shot fly (from the left edge of the screen). If it misses, nothing will happen. But if it hits the target, the enemy tank explodes and the screen changes color to simulate new terrain and new weather. You score ten points for each successful shot.

If you miss, *you* become the target. The enemy tank will turn toward you and fire. The enemy rarely misses—and you'll lose one armor point when you get hit. In the upper right-hand corner is a status indicator which displays how many points you've scored and how much armor you have left. When your defenses reach zero, your tank is destroyed. The viewport cracks, and the tank is reduced to scrap.

Sometimes, if you fire often enough, you can force the enemy tank to vanish at the right side of the screen. It's fled under your bombardment, without firing a shot. Unfortunately, another one immediately takes its place on the left. However, this can give you some breathing space, especially if you're using the Tiger, whose turret swings around so slowly.

Extra Chances

Losing a tank is not a total catastrophe, however. You manage to escape by the skin of your teeth, and make your way back to headquarters. You are awarded a rank, based on your performance, from Private (less than 50 points scored) to Marshal (over 1000). But if no points are scored, you're branded a Trainee. Whatever your rank, you're given another chance to do battle. Choose another tank and the game begins again; you may yet earn the exalted rank of Marshal.

Supertank

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C.

```
10 PRINT"{CLR}{HOME}":RESTORE:V=53248:POKEV+32,Ø:P
   OKEV+33,1:POKEV+17,PEEK(V+17)AND247     :rem 174
15 PRINTTAB(53)"{BLU}{RVS}W*E*L*C*O*M*E!":PRINTTAB
   (59)"{RED}{RVS}TO"                      :rem 147
17 PRINTTAB(55)"{RED}{RVS}SUPERTANK!":PRINT
                                           :rem 158
2Ø FORQ=1TO4:PRINTTAB(14)"{5}{N}":NEXT      :rem 5Ø
22 PRINTTAB(14)"{N}{RVS}£{8 SPACES}{*}":PRINTTAB(
   14)"{RVS}£ Q Q{2 SPACES}QQQ {*}{OFF}{2 I}{8 O}
   {3 I}"                                  :rem 2Ø9
24 PRINTTAB(13)"{RVS}£{2 SPACES}Q Q{3 SPACES}M
   {3 SPACES}{OFF}{2 U}{8 Y}{3 U}"         :rem 2Ø7
```

```
26 PRINTTAB(13)"{RVS}{3 SPACES}QQQ{2 SPACES}QQQ
   {2 SPACES}{OFF}"                              :rem 79
28 PRINT"{3 SPACES}{RVS}£{30 SPACES}E*3{OFF}"
                                                 :rem 103
30 PRINT"{2 SPACES}{RVS}£{32 SPACES}E*3{OFF}"
                                                 :rem 96
32 PRINT"{GRN}E2 +3E53{RVS}{34 SPACES}{OFF}{GRN}
   E3 +3"                                        :rem 236
34 PRINT"E3 +3E53M W{RVS}£ E*3{OFF}W{RVS}£ E*3
   {OFF}W{RVS}£ E*3{OFF}W{RVS}£ E*3{OFF}W{RVS}£
    E*3{OFF}W{RVS}£ E*3{OFF}W{RVS}£ E*3{OFF}WN
   {GRN}E4 +3"                                   :rem 255
36 PRINT"E4 +3E53M {RVS} Q {OFF} {RVS} Q {OFF}
   {RVS} Q {OFF} {RVS} Q {OFF} {RVS} Q {OFF} {RVS}
    Q {OFF} {RVS} Q {OFF}N{GRN}E5 +3"            :rem 148
38 PRINT"E5 +3E53ME*3{RVS} {OFF}£WE*3{RVS} {OFF}
   £WE*3{RVS} {OFF}£WE*3{RVS} {OFF}£WE*3{RVS}
   {OFF}£WE*3{RVS} {OFF}£WE*3{RVS} {OFF}£{GRN}
   E6 +3"                                        :rem 31
40 PRINT"E6 +3E53E26 Y3{GRN}E7 +3"               :rem 239
42 FORQ=1TO2:PRINT"E39 +3":NEXT                  :rem 175
43 GOSUB800                                      :rem 127
48 S=54272:FORL=STOS+24:POKEL,0:NEXT             :rem 14
50 PRINT"{CLR}{HOME}{WHT}":POKEV+32,1:POKEV+33,0:P
   RINTTAB(120)                                  :rem 187
52 GOSUB1000                                     :rem 168
60 PRINT"{CLR}{HOME}{WHT}":PRINTTAB(90)"TANK SPECI
   FICATIONS"                                    :rem 192
62 PRINTTAB(49)"{RVS}{GRN} STRONG {OFF}{3 SPACES}
   {RVS}{YEL} MEDIUM {OFF}{2 SPACES}{RVS}{WHT} WEA
   K {OFF}"                                      :rem 65
64 PRINTTAB(40)"{GRN}{RVS}1.TIGER{OFF}{4 SPACES}AR
   MOR{5 SPACES}{YEL}FIRE{5 SPACES}{WHT}SPEED"
                                                 :rem 30
66 PRINTTAB(40)"{YEL}{RVS}2.T-34{OFF}{5 SPACES}
   {GRN}SPEED{5 SPACES}{YEL}ARMOR{4 SPACES}{WHT}FI
   RE"                                           :rem 44
68 PRINTTAB(40)"{WHT}{RVS}3.SHERMAN{OFF}{2 SPACES}
   {GRN}FIRE{6 SPACES}{YEL}SPEED{4 SPACES}{WHT}ARM
   OR":PRINTTAB(120)                             :rem 62
70 PRINT:INPUT"{HOME}{15 DOWN}{3 SPACES}WHICH TANK
    DO YOU CHOOSE";TA                            :rem 214
72 IFTA<1ORTA>3THEN80                            :rem 56
78 PRINT"{4 DOWN}{13 SPACES}GET READY !":FORQ=1TO5
   00STEP.5:NEXT:GOTO85                          :rem 10
80 PRINT"{HOME}{15 DOWN}{10 SPACES}YOU CAN'T DO TH
   AT!{4 SPACES}":GOSUB1300:GOTO70               :rem 210
85 V=53248:GOSUB1100                             :rem 130
90 PRINT"{CLR}{HOME}":POKEV+32,0:POKEV+33,1:rem 56
```

```
92 PRINTTAB(7)"{RVS}E5}£E*}{OFF}{4 SPACES}{RVS}£
   E*}{OFF}{7 SPACES}{RVS}£E*}{OFF}{4 SPACES}
   {RVS}£E*}{OFF}{4 SPACES}{RVS}{BLK}SCORE" :rem 1
94 PRINT"{2 SPACES}{RVS}E5}£E*}{OFF}{2 SPACES}
   {RVS}£{2 SPACES}{OFF}{3 SPACES}{RVS}£
   {2 SPACES}E*}{OFF}{5 SPACES}{RVS}£{2 SPACES}
   E*}{OFF}{2 SPACES}{RVS}£{2 SPACES}E*}{OFF}
   {2 SPACES}{BLK}";SC                      :rem 52
96 PRINT" {RVS}E5}£{2 SPACES}E*}£{3 SPACES}E*}
   {OFF}{2 SPACES}{RVS}{4 SPACES}E*}{OFF}
   {2 SPACES}{RVS}£{5 SPACES}{OFF} {RVS}£
   {4 SPACES}E*}{OFF}{2 SPACES}{RVS}{BLK}ARMOR"
                                           :rem 171
98 PRINT"{RVS}E5}£{9 SPACES}E*}£{5 SPACES}E*}£
   {6 SPACES}£{6 SPACES}E*}{OFF}{BLK}";AR   :rem 29
100 FORQ=1TO17:PRINT"{RVS}{GRN}E39 +}":NEXT:rem 64
110 PRINT"{RVS}E5}EQ}CCCCCCCCCEW}{RIGHT}£
    {13 SPACES}E*}{RIGHT}EQ}CCCCCCCCCEW}"   :rem 65
112 PRINT"{RVS}E5}EQ}CCCCCCCCCEW}{17 SPACES}EQ}CCC
    CCCCCCEW}"                             :rem 129
114 GOSUB420                               :rem 172
120 CB=1:TI$="000000"                       :rem 34
150 POKEV+21,15:POKE2040,13:POKEV+39,0:POKEV,170:P
    OKEV+1,150:Y=170:X=150:SH=0             :rem 27
151 POKE2042,193:POKEV+41,11:RF=0:UT=110:POKEV+42,
    0:POKE2043,195                         :rem 206
152 POKE2041,14:POKEV+40,0:POKEV+2,X1:POKEV+3,Y1
                                           :rem 109
180 S=NOTPEEK(56321)AND15:U=SAND1:D=SAND2:L=SAND4:
    R=SAND8:Y1=0:X1=0                      :rem 165
182 POKEV+23,0:POKEV+29,0                  :rem 189
185 IFUTHENX=X-M1:IFX<110THENX=X+M1        :rem 253
187 IFDTHENX=X+M1:IFX>180THENX=X-M1        :rem 247
189 IFRTHENY=Y+M1:IFY>245THENY=Y-M1         :rem 14
191 IFLTHENY=Y-M1:IFY<90THENY=Y+M1         :rem 205
200 POKEV,Y:POKEV+1,X                       :rem 59
210 J=NOTPEEK(56321)AND16:IFJ=16THENGOSUB245
                                           :rem 189
230 BO=BO+.5:GOSUB310:GOTO180              :rem 220
245 SH=SH+1:X1=X:MR=Y:HH=Y/2:GOSUB400       :rem 64
247 FORDD=DDTOHHSTEP5:POKEV+2,DD:POKEV+3,X1:GOSUB3
    30:NEXT                                :rem 224
250 POKE2041,15:FORDD=DDTOMRSTEP5:POKEV+2,DD:POKEV
    +3,X1:GOSUB330:NEXT                    :rem 171
251 IF(PEEK(V+30)AND4)>0THENIF(PEEK(V+30)AND4)>0TH
    ENGOSUB253                             :rem 110
252 X1=0:DD=0:Y1=0:POKEV+2,X1:POKEV+3,MR:POKE2041,
    14:RETURN                              :rem 131
253 POKE2041,192:POKEV+23,2:POKEV+29,2     :rem 184
```

```
254 POKEV+3,X1-10:POKEV+2,DD-12:GOSUB410:GOSUB495
                                          :rem 204
258 FORRE=1TO500:NEXT:POKEV+23,0:POKEV+29,0
                                          :rem 152
260 X1=0:DD=0:Y1=0:POKEV+2,X1:POKEV+3,MR:POKE2041,
    14:RETURN                             :rem 130
310 RF=RF+5:IFRF>215THEN350                :rem 95
315 POKEV+4,RF:POKEV+5,UT:RETURN           :rem 79
330 RF=RF+1.8:IFRF>215THENRF=0             :rem 48
333 IFRF=0THEN340                         :rem 243
335 POKEV+4,RF:POKEV+5,UT:RETURN           :rem 81
340 RF=0:UT=110:RS=INT(RND(0)*60):UT=UT+RS:GOTO330
                                           :rem 61
350 F1=RF:F2=UT:POKE2042,194:POKEV+6,F1:POKEV+7,F2
    :GOSUB400                              :rem 77
355 FORQ=F2TO150STEP.4:POKEV+7,Q:NEXT:POKEV+29,8:P
    OKEV+23,8:POKEV+6,F1-12                :rem 55
360 F2=150:FORQ=F2TO230STEP.6:POKEV+7,Q:NEXT:GOSUB
    410                                    :rem 73
390 POKEV+6,0:POKEV+7,0:POKEV+4,0:POKEV+5,0:RF=0:P
    OKEV+23,0:POKEV+29,0                  :rem 235
391 POKE2042,193:GOTO499                  :rem 118
400 S=54272:FORL=STOS+24:POKEL,0:NEXT:POKES+5,9:PO
    KES+6,16:POKES+24,15                   :rem 70
405 POKES+4,129:POKES+1,34:POKES,75:RETURN :rem 91
410 S=54272:FORL=STOS+24:POKEL,0:NEXT:POKES+5,11:P
    OKES+6,16:POKES+24,15                 :rem 112
415 POKES+4,129:POKES+1,54:POKES,111:RETURN
                                          :rem 133
420 S=54272:FORL=STOS+24:POKEL,0:NEXT:POKES+5,11:P
    OKES+6,56:POKES+24,15                 :rem 117
425 POKES+4,129:POKES+1,51:POKES,97:RETURN :rem 96
495 SC=SC+10:CB=0:RN=INT(RND(0)*15):CB=CB+RN:GOTO5
    00                                     :rem 95
499 AR=AR-1                               :rem 103
500 V=53248:PRINT"{HOME}":POKEV+32,0:POKEV+33,CB
                                          :rem 248
501 RF=0:UT=110:RS=INT(RND(0)*60):UT=UT+RS:RF=RF+1
    .5:                                   :rem 211
502 PRINTTAB(7)"{RVS}E5}£E*}{OFF}{4 SPACES}{RVS}
    £E*}{OFF}{7 SPACES}{RVS}£E*}{OFF}{4 SPACES}
    {RVS}£E*}{OFF}{4 SPACES}{RVS}{BLK}SCORE"
                                           :rem 45
504 PRINT"{2 SPACES}{RVS}E5}£E*}{OFF}{2 SPACES}
    {RVS}£{2 SPACES}{OFF}{3 SPACES}{RVS}£
    {2 SPACES}E*}{OFF}{5 SPACES}{RVS}£{2 SPACES}
    E*}{OFF}{2 SPACES}{RVS}£{2 SPACES}E*}{OFF}
    {2 SPACES}{BLK}";SC                    :rem 96
```

```
506 PRINT" {RVS}E5 £{2 SPACES}E* £{3 SPACES}E* 
    {OFF}{2 SPACES}{RVS}{4 SPACES}E* {OFF}
    {2 SPACES}{RVS}£{5 SPACES}{OFF} {RVS}£
    {4 SPACES}E* {OFF}{2 SPACES}{RVS}{BLK}ARMOR"
                                          :rem 215
508 PRINT"{RVS}E5 £{9 SPACES}E* £{5 SPACES}E* £
    {6 SPACES}£{6 SPACES}E* {OFF}{BLK}";AR:rem 73
509 IFAR=0THEN549                         :rem 254
510 RETURN                                :rem 118
549 LL=18:BL=12:BB=15                     :rem 169
550 PRINT"{HOME}":POKEV+32,0:POKEV+33,1   :rem 214
558 FORQ=5TO7:PRINTTAB(Q)"{BLK}M"SPC(10)"M":NEXT
                                          :rem 41
560 PRINTTAB(7)"{BLK}N"SPC(11)"M"SPC(4)"NM":rem 29
561 PRINTTAB(6)"N"SPC(13)"M"SPC(2)"N"SPC(2)"M"
                                          :rem 58
562 PRINTTAB(5)"{BLK}N"SPC(15)"{BLK}MN"SPC(4)"M"
                                          :rem 177
563 PRINTTAB(5)"M"SPC(21)"N"SPC(3)"NM"    :rem 142
564 PRINTTAB(6)"M"SPC(6)"NM"SPC(11)"N"SPC(3)"N"SPC
    (2)"M"                                :rem 136
565 PRINTTAB(7)"M"SPC(4)"N"SPC(2)"M"SPC(10)"M"SPC(
    2)"N"SPC(4)"M"                        :rem 52
566 PRINTTAB(8)"M"SPC(2)"N"SPC(4)"M"SPC(10)"MN"SPC
    (6)"M"                                :rem 139
567 PRINTTAB(9)"MN"SPC(6)"M"SPC(17)"N"    :rem 158
568 PRINTTAB(18)"M"SPC(15)"N":FORLB=1TO6:PRINTTAB(
    LL)"N"SPC(14)"N":LL=LL-1:NEXT         :rem 60
570 FORQ=1TO5:PRINTTAB(BL)"N"SPC(BB)"M":BL=BL-1:BB
    =BB+2:NEXT                            :rem 187
580 RESTORE:POKEV+23,0:POKEV+29,0:POKEV+21,0:GOSUB
    420:FORQ=1TO500STEP.1:NEXT            :rem 66
585 S=54272:FORL=STOS+24:POKEL,0:NEXT:GOSUB1200
                                          :rem 193
588 V=53248: BO=BO/10:XX=INT(BO):SC=SC+XX:IFSC>HST
    HENHS=SC                              :rem 174
589 PRINT"{HOME}{CLR}":POKEV+32,0:POKEV+33,1:POKE5
    3281,1                                :rem 62
590 PRINTTAB(85)"{RVS}E2 B*O*N*U*S ";XX;SPC(3)"
    {RVS}SHOTS FIRED";SH:PRINTTAB(45)"{RVS}YOUR";
                                          :rem 201
591 PRINT" SCORE";SC;SPC(3)"{RVS}HIGH SCORE ";HS:P
    RINTTAB(49)"{RVS}YOUR RANK IS {BLK}";B$:rem 36
592 PRINTTAB(43)"{RVS}E2 YOUR TOTAL SURVIVING TIME
     ";TI$                                :rem 114
593 PRINTTAB(86)"{RVS}WANT TO PLAY AGAIN? (Y/N)"
                                          :rem 243
595 GETC$:IFC$=""THEN595                  :rem 109
596 SC=0:B$="":IFC$="Y"THEN599            :rem 168
597 IFC$="N"THENSYS64738                  :rem 164
```

```
598 C$="":GOTO595                                    :rem 164
599 SH=Ø:SC=Ø:BO=Ø:XX=Ø:POKEV+32,1:POKEV+33,Ø:GOTO
    6Ø                                               :rem 172
8ØØ S=54272:FORL=STOS+24:POKEL,Ø:NEXT:POKES+5,9:PO
    KES+6,Ø:POKES+24,15                              :rem 19
8Ø1 READHF,LF,DR:IFHF=-1THENRETURN                   :rem 196
8Ø4 POKES+1,HF:POKES,LF:POKES+4,33:FORT=1TODR:NEXT
    :POKES+4,32:GOTO8Ø1                              :rem 168
81Ø DATA18,2Ø9,1Ø24,15,21Ø,512,18,2Ø9,512,16,195,1
    Ø24,14,24,512,11,48,512                          :rem 45
811 DATA18,2Ø9,2ØØ,16,195,2ØØ,15,21Ø,2ØØ,14,24,2ØØ
    ,15,21Ø,512,22,96,512                            :rem 178
812 DATA16,195,1Ø24,11,48,1Ø24,15,21Ø,512,14,24,2Ø
    Ø,12,143,2ØØ,11,48,2ØØ                           :rem 222
813 DATA1Ø,143,2ØØ,11,48,2ØØ,12,143,2ØØ,11,48,512,
    16,195,512,14,239,512                            :rem 185
814 DATA11,48,512,15,21Ø,2ØØ,15,21Ø,2ØØ,14,24,2ØØ,
    12,143,2ØØ,11,48,2ØØ                             :rem 1Ø5
815 DATA1Ø,143,2ØØ,11,48,2ØØ,12,143,2ØØ,11,48,512,
    16,195,1Ø24,22,96,512                            :rem 186
816 DATA18,2Ø9,1Ø24,15,21Ø,512,18,2Ø9,512,16,195,1
    Ø24,14,24,512,11,48,512                          :rem 51
817 DATA18,2Ø9,2ØØ,16,195,2ØØ,15,21Ø,2ØØ,14,24,2ØØ
    ,15,21Ø,512,22,96,512                            :rem 184
818 DATA16,195,1Ø24,11,48,1Ø24,15,21Ø,512,14,24,2Ø
    Ø,12,143,2ØØ,11,48,2ØØ                           :rem 228
819 DATA1Ø,143,2ØØ,11,48,2ØØ,12,143,2ØØ,11,48,512,
    16,195,512,14,239,512                            :rem 191
82Ø DATA11,48,512,15,21Ø,2ØØ,15,21Ø,2ØØ,14,24,2ØØ,
    12,143,2ØØ,11,48,2ØØ                             :rem 1Ø2
821 DATA1Ø,143,2ØØ,11,48,2ØØ,12,143,2ØØ,11,48,512,
    11,48,1Ø24,-1,Ø,Ø                                :rem 21Ø
9ØØ DATA255,255,255,128,24,1,128,24,1,128,24,1,128
    ,24,1,128,24,1,128,24,1                          :rem 35
9Ø5 DATA128,24,1,128,36,1,128,66,1,255,129,255,128
    ,66,1,128,36,1,128,24,1,128                      :rem 1
91Ø DATA24,1,128,24,1,128,24,1,128,24,1,128,24,1,1
    28,24,1,255,255,255                              :rem 93
915 DATAØ,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,
    31,254,Ø,49,255,192,96,255                       :rem 88
92Ø DATA24Ø,196,127,252,2Ø6,127,255,2Ø6,127,255,19
    6,127,252,96,255,24Ø,49,255                      :rem 35
925 DATA192,31,254,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
                                                     :rem 1Ø9
926 DATAØ,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,
    63,24Ø,Ø,1Ø3,252,Ø,195,255                       :rem 66
927 DATAØ,219,255,192,195,255,Ø,1Ø3,252,Ø,63,24Ø,Ø
    ,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø                     :rem 2Ø7
928 DATAØ,Ø,Ø,Ø,Ø,Ø,Ø,Ø                              :rem 113
```

```
930 DATA0,0,2,34,128,4,0,64,2,146,128,16,0,16,10,7
    3,32,64,0,4,17,140,96,64,0,4              :rem 2
935 DATA17,17,16,64,0,4,8,136,136,64,0,4,17,17,16,
    32,0,8,8,136,128,16,0,16,2,72             :rem 80
940 DATA128,4,0,64,0,0,0                     :rem 184
945 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,64,0,0,64,0,0,64
    ,0,0,64,0,0,126,0,0,255,255,0            :rem 221
947 DATA255,0,63,255,252,127,255,254,255,255,255,1
    00,68,70,37,85,84,20,68,72               :rem 243
950 DATA15,255,240,0,0,0,0,0,0,0,0,0         :rem 228
955 DATA0,2,0,0,2,0,0,2,0,0,2,0,0,58,0,0,70,0,0,18
    6,0,0,130,0,0,254,0,1,255,0              :rem 128
960 DATA3,255,128,7,255,192,0,124,0,15,187,224,8,1
    86,32,15,187,224,8,130,32,15              :rem 52
965 DATA131,224,0,0,0,0,0,0,0,0,0             :rem 83
970 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,60,0
    ,0,126,0,0,255,0,0,255,0,0                :rem 41
975 DATA126,0,0,60,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,0,0,0,0,0                 :rem 178
1000 FORA1=832TO894:READQ1:POKEA1,Q1:NEXT   :rem 22
1010 FORA2=896TO958:READQ2:POKEA2,Q2:NEXT   :rem 38
1015 FORA3=960TO1022:READQ3:POKEA3,Q3:NEXT :rem 70
1020 FORA4=12288TO12350:READQ4:POKEA4,Q4:NEXT
                                             :rem 226
1025 FORA5=12352TO12414:READQ5:POKEA5,Q5:NEXT
                                             :rem 228
1030 FORA6=12416TO12478:READQ6:POKEA6,Q6:NEXT
                                             :rem 239
1035 FORA7=12480TO12542:READQ7:POKEA7,Q7:NEXT
                                             :rem 241
1090 RETURN                                  :rem 170
1100 IFTA=1THENAR=5:IFTA=1THENM1=1           :rem 111
1105 IFTA=2THENAR=3:IFTA=2THENM1=2           :rem 117
1110 IFTA=3THENAR=1:IFTA=3THENM1=3           :rem 114
1150 RETURN                                  :rem 167
1200 IFSC=0THENB$="TRAINEE"                  :rem 115
1201 IFSC>0ANDSC<51THENB$="PRIVATE"          :rem 147
1202 IFSC>52ANDSC<101THENB$="SERGEANT"        :rem 53
1204 IFSC>101ANDSC<201THENB$="LIEUTENANT"      :rem 3
1206 IFSC>201ANDSC<401THENB$="CAPTAIN"        :rem 15
1208 IFSC>401ANDSC<601THENB$="MAJOR"         :rem 142
1210 IFSC>601ANDSC<801THENB$="COLONEL"        :rem 30
1212 IFSC>801ANDSC<1001THENB$="* GENERAL *"
                                             :rem 145
1214 IFSC>1001THENB$="** MARSHAL **"         :rem 179
1216 RETURN                                  :rem 170
1232 GOTO500                                 :rem 150
1300 FORI=1TO1500:NEXT:RETURN                 :rem 94
```

Moving Message

Robert F. Lambiase

Scrolling messages across a screen can be used for advertising, simple reminders, or important notices at work and school. With "Moving Message," you can create, edit, save, load, and display messages up to 3000 characters long.

A message scrolling across a screen can be a real attention getter. It has all the right ingredients: motion and the ability to display more information than would fit on a single screen. You *could* flip through multiple screens, but the speed might be too fast or too slow for the viewer. A scrolling display paces the viewer and continuously gives new information.

"Moving Message" lets you create and even edit a message; your Commodore 64 then scrolls that message across your display screen.

Scrolling the 64

The 64 has both horizontal and vertical scrolling capability. For this application, you'll only need horizontal scrolling.

The computer's screen display is made up of 320 pixel-columns which are grouped into 40 character-columns, each with 8 pixel-columns. The first character column starts at the first pixel column. This can be changed, however, by altering the three least significant bits of address 53270. Sequencing these bits changes which pixel column (first through eighth) will be the starting point of the first character-column, and gives the effect of the character smoothly sliding over an entire character column. Sequencing up moves the characters to the right, and sequencing down moves the characters to the left.

First Scroll, Then Shift

Let's take a look at an example. Assume there is a single character on the right side of the screen that will be scrolled to the left. Sequencing the scroll bits from seven to zero will slide the character over to within a single pixel column of being a full character column from where it started. To move over that one additional pixel column, the scroll bits must be reset to seven, and the character must be simultaneously moved left one screen position by altering the screen memory.

Machine Language for Speed

Now it gets a little tricky. The computer can't simultaneously reset the scroll bits and alter the screen memory. For maximum speed, the use of machine language is essential. Unfortunately, not even the breakneck speed of machine language is enough. As the character scrolls across the screen, there would be occasional flashes of the character. This occurs when the video chip is displaying the character between the time the scroll bits are reset and the time the characters are shifted left. This problem can be overcome by permitting the scroll reset and shifting to be done only when the video chip is not writing on the screen. To do this, the raster register is used.

Raster Register to the Rescue

Reading the value in the raster register at location 53266 yields the current raster line being written. The machine language program used to reset the scroll bits and shift the characters left is preceded by a small loop checking for raster line 50. This raster line is just past where the characters are scrolling. The speed of the machine language program is sufficient to finish all operations before the screen finishes scanning its last line.

Filling the Ends

There's just one more detail to handle. Scrolling to the left leaves a gap on the right side to the screen. Scrolling to the right leaves a gap on the left. This is remedied by a special feature of the video chip. By resetting bit 3 of location 53270 to 0, the screen is reduced to 38 characters per line. The spaces on either side of the screen are no longer visible since they're obscured by the widened borders.

Putting It All Together

Moving Message lets you create a message up to 3000 characters in length, edit it, save it, recall it, and scroll it across the screen. When the last character scrolls off the screen, the first character scrolls onto the screen again. The message is stored in consecutive memory locations starting at location 50000, and may consist of letters, numbers, punctuation, and spaces. The end of the message is marked by pressing the space bar while the SHIFT key is held down. It appears as a normal space, but its ASCII code is 160 instead of 32, and its screen code (used for POKEs) is 96 instead of 32.

Using the Program

Make sure you use "The Automatic Proofreader," in Appendix C, to help you type in Moving Message. The Proofreader insures that you'll type the program in correctly the first time. Save it to tape or disk, then load and run it.

You're ready to enter your message. Simply type it in. As you enter the characters (which first appear at the arrow on the right side of the screen), the message moves to the left. End the message with the SHIFT-space key combination. The message automatically starts to scroll.

If you need to change anything in the message, hit any key and the scrolling stops. Use the cursor keys to position the arrow at the desired place in the message. The cursor-down key shifts the message to the left, while the cursor-right key moves the message to the right. This permits two-fingered operation.

Change a character by positioning it over the arrow and typing in the new character. You can even type over your previous end-of-message mark (SHIFTed space), but remember to add a new one. Characters can be inserted or deleted at the arrow by using the f1 and f3 keys respectively.

When your editing is complete, the f5 key is used to start the scrolling again.

Saving and Loading

Saving and loading of messages is possible with either tape or disk. Press the f4 key to save, the f6 key to load. You'll have to provide a filename and then press *T* for tape or *D* for disk. The message is read into memory and then begins to scroll across the screen. (If you're using tape and loading a message, sometimes you'll see unwanted characters between the end of the message and the next time it appears on the left. To eliminate these characters, hit any key to return to the main menu, then use the cursor-down key to move to the end of the message. Press the SHIFTed space combination again, and then f5 to start the scroll. The message should appear as you want.)

To keep up with the speed of the disk, it's saved as if it were a machine language program. Since the Datassette is slower, the data is stored byte by byte. When the tape file is read in, the end is recognized when the SHIFTed space is seen.

Enhancements

Many enhancements of this program are possible. It's not too difficult to have two messages scrolling across the screen simultaneously. With more modification, you should be able to scroll large characters.

Moving Message

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C.

```
140 DIMH(8):FORJ=1TO7:READH(J):NEXT          :rem 165
150 DATA 17,29,133,135,140,139,138           :rem 137
160 G=53270:POKEG,8:POKE53280,6:C$=CHR$(147):N$=CH
    R$(18):F$=CHR$(146):GOSUB460             :rem 143
170 IS=50000:I=IS:FORV=49960TO49999:POKEV,32:NEXT
                                               :rem 6
180 REM{2 SPACES}LOAD MACHINE LANGUAGE PROGS.
                                             :rem 143
190 FORJ=49152TO49193:READD:POKEJ,D:NEXT    :rem 240
200 DATA162,0,189,161,4,157,160,4,232,224,39,208,2
    45,96                                    :rem 198
210 DATA162,39,189,159,4,157,160,4,202,208,247,96
                                             :rem 173
220 DATA173,18,208,117,50,208,249,169,7,141,22,208
    ,32,0,192,96                              :rem 34
230 REM                                      :rem 121
240 REM{2 SPACES}MESSAGE INPUT                :rem 15
250 GETA$:IFA$=""THEN250                      :rem 81
260 A=ASC(A$):P=A+64*((A>63)AND(A<161))       :rem 45
270 IFA=140THEN1260                           :rem 51
280 IF((A<32ORA>132)ANDA<>160)THENPOKEI,96:IM=I:GO
    SUB320:GOTO630                           :rem 123
290 POKE1223,P:POKEI,P                        :rem 60
300 IFA=160THENIM=I:GOSUB320:GOTO350         :rem 220
310 SYS49152:POKE1223,32:I=I+1:GOTO250       :rem 201
320 FORJ=I+1TOI+41:POKEJ,32:NEXT:RETURN      :rem 251
330 REM                                      :rem 122
340 REM{2 SPACES}MESSAGE SCROLLING            :rem 45
350 I=IS:POKEG,7:PRINTC$:GOSUB450             :rem 78
```

```
360 POKE1223,PEEK(I):I=I+1                    :rem 224
370 IFPEEK(1184)=96THENI=IS                   :rem 106
380 FORJ=6TOØSTEP-1:POKEG,J:FORK=ØTO7:NEXT:NEXT
                                              :rem 119
390 SYS49178                                  :rem 168
400 GETB$:TD=TD:IFB$=""THEN360                :rem 249
410 B=ASC(B$):IFB=140THEN1260                 :rem 119
420 GOTO650                                   :rem 106
430 REM                                       :rem 123
440 REM{2 SPACES}INSTRUCTION DISPLAY          :rem 244
450 FORJ=55456TO55495:POKEJ,14:NEXT:RETURN:rem 150
460 PRINTC$;:GOSUB450                          :rem 28
470 POKE55535,14:POKE1263,30                   :rem 38
480 FORJ=1TO8:PRINT:NEXT:PRINTTAB(13)"MOVING MESSA
    GE"                                       :rem 194
490 PRINTTAB(83)"CHARACTERS ARE ENTERED AT THE ARR
    OW"                                       :rem 193
500 PRINTTAB(40)"USE: ";N$;"SHIFT SPACE";F$;" TO M
    ARK END OF MESSAGE"                       :rem 146
510 PRINTTAB(5)N$;"F1";F$;" TO INSERT A CHARACTER"
                                               :rem 82
520 PRINTTAB(5)N$;"F3";F$;" TO DELETE A CHARACTER"
                                               :rem 51
530 PRINTTAB(5)N$;"F4";F$;" TO SAVE"          :rem 227
540 PRINTTAB(5)N$;"F5";F$;" TO RESTART SCROLLING"
                                              :rem 136
550 PRINTTAB(5)N$;"F6";F$;" TO LOAD"          :rem 216
560 PRINTTAB(5)N$;"F8";F$;" TO EXIT"          :rem 245
570 PRINTTAB(40)"CURSOR ";N$;"DOWN";F$;" & ";N$;"R
    IGHT";F$;" KEYS ARE USED TO"               :rem 59
580 PRINT" POSITION THE CHARACTERS OVER THE ARROW"
                                               :rem 76
590 PRINT" FOR EDITING."                      :rem 136
600 RETURN                                    :rem 118
610 REM                                       :rem 123
620 REM{2 SPACES}MESSAGE EDITING              :rem 133
630 IFA=139THEN870                             :rem 17
640 IFA=136THEN1040                            :rem 53
650 I=IS:POKEG,8:PRINT"{CLR}":GOSUB460        :rem 195
660 POKE1223,PEEK(I)                          :rem 126
670 GETA$:IFA$=""THEN670                       :rem 93
680 A=ASC(A$):P=A+64*((A>63)AND(A<161))        :rem 51
690 Q=Ø:FORJ=1TO7:IFA=H(J)THENQ=J              :rem 65
700 NEXT                                      :rem 214
710 ONQGOTO770,800,820,350,1260,870,1040       :rem 92
720 IF(A=134ANDI<IM)THEN840                    :rem 72
730 IFA=134THEN670                             :rem 11
740 POKE1223,P:POKEI,P                         :rem 60
750 IFA=160THENIM=I:GOSUB320:GOTO350          :rem 229
760 GOTO780                                   :rem 117
```

```
770 IFPEEK(1223)=96THEN670                      :rem 227
780 I=I+1:SYS49152:POKE1223,PEEK(I):IFI>=IMTHENIM=
    IM+1                                        :rem 59
790 GOTO670                                     :rem 118
800 IFI=ISTHEN670                               :rem 21
810 I=I-1:SYS49166:POKE1184,PEEK(I-39):GOTO670
                                                :rem 212
820 FORJ=IMTOISTEP-1:POKEJ+1,PEEK(J):NEXT :rem 152
830 POKEI,32:POKE1223,32:IM=IM+1:POKEIM+40,32:GOTO
    670                                         :rem 148
840 FORJ=ITOIM-1:POKEJ,PEEK(J+1):NEXT           :rem 94
850 POKE1223,PEEK(I):POKEIM,32:IM=IM-1:GOTO670
                                                :rem 32
860 REM                                         :rem 130
870 REM{2 SPACES}LOAD ROUTINE                   :rem 201
880 INPUT"{CLR}FILE NAME";K$                    :rem 242
890 INPUT"TAPE (T) OR DISK (D)";A$              :rem 69
900 IFA$="T"THEN970                             :rem 49
910 OPEN15,8,15,"I0"                            :rem 17
920 OPEN3,8,0,"0:"+K$+",P,R"                    :rem 157
930 INPUT#15,EN,EM$,ET,ES                       :rem 223
940 IFEN<>0THENPRINT;EN,EM$,ET,ES:GOTO1240:rem 152
950 POKE185,0:POKE195,40:POKE196,195:SYS62631
                                                :rem 110
960 CLOSE15:CLOSE3:GOTO650                      :rem 108
970 OPEN3,1,0,K$                                :rem 89
980 X=IS-40                                     :rem 99
990 GET#3,A$                                    :rem 105
1000 A=ASC(A$+CHR$(0)):POKEX,A:X=X+1            :rem 42
1010 IFA<>96THEN990                             :rem 76
1020 CLOSE3:GOTO340                             :rem 118
1030 REM                                        :rem 168
1040 REM{2 SPACES}SAVE ROUTINE                  :rem 254
1050 INPUT"{CLR}FILE NAME";K$                   :rem 24
1060 U=IM+42:UH=INT(U/256):UL=U-256*UH          :rem 239
1070 INPUT"TAPE (T) OR DISK (D)";A$             :rem 108
1080 IFA$="T"THEN1160                           :rem 137
1090 OPEN15,8,15,"I0"                           :rem 65
1100 OPEN3,8,1,"0:"+K$+",P,W"                   :rem 202
1110 INPUT#15,EN,EM$,ET,ES                      :rem 6
1120 IFENTHENPRINT;EN,EM$,ET,ES:GOTO1240        :rem 21
1130 POKE193,40:POKE194,195                     :rem 243
1140 POKE174,UL:POKE175,UH:SYS62957             :rem 115
1150 CLOSE15:CLOSE3:GOTO650                     :rem 148
1160 OPEN3,1,1,K$                               :rem 130
1170 FORX=IS-40TOIM+40                          :rem 58
1180 PRINT#3,CHR$(PEEK(X));                     :rem 52
1190 NEXT:PRINT#3                               :rem 39
```

```
1200 CLOSE3:GOTO340                                :rem 118
1210 CLOSE1:INPUT"DO YOU WISH TO CONTINUE (Y/N)";D
     $                                              :rem 22
1220 IFD$="Y"THEN620                                :rem 93
1230 GOTO1260                                      :rem 200
1240 CLOSE15:CLOSE3:INPUT"DO YOU WISH TO CONTINUE
     {SPACE}(Y/N)";D$                               :rem 49
1250 IFD$="Y"THEN620                                :rem 96
1260 POKEG,8:END                                   :rem 180
```

3
Education

Word Match

Andy VanDuyne

How good is your memory? "Word Match," a memory game for the 64, will test your children's ability to remember short words. Suitable for grades K through 6, it can be modified for more difficult levels.

Loosely adapted from the old TV show "Concentration," "Word Match" is designed to entertain and test the memory of one or two players. The object is to find and match pairs of words hidden behind rows of colored blocks.

Word Match is easy to learn. Players take turns selecting blocks, which disappear to reveal the words they conceal. An unsuccessful match means it's the next player's turn. Players who successfully match a pair of words gain another turn, and the matched blocks turn into the player's own color. To win the game, a player must match more pairs of hidden words than the opponent. The opponent, by the way, can be either another person or the computer itself.

Word Match is ideal for grade-school children because all the words are only three letters long. A total of 50 words are included in the program, in lines 32–34. You can customize the program with your own word list by amending those lines. It's best if you keep the number of words at 50. Just make sure that there are no spaces between the words (just as you see in lines 32–34), and that the lines do not exceed 80 characters. To make the game suitable for older children, you may want to include some unusual three-letter words and use the game for a vocabulary builder, as well as a memory game.

Matching the Words

Type the program in and save it. You'll find "The Automatic Proofreader" program in Appendix C an immense help in entering Word Match, for the Proofreader makes it almost impossible to enter a line incorrectly.

When you first run Word Match, a two-screen instruction display appears. After you've read the first screen, press the RETURN key to look at the second. (If you've played the game before and don't want to be bothered with the instructions, just hit the *N* key.) Then you'll be asked for the players' names. After the second name is entered, the screen clears, and a message reminds you that the computer is selecting the words.

Although Word Match was designed primarily for two players, one person can compete against the computer by typing *64* as a player's name when the program starts.

An interesting twist is to enter the computer's name for both players and then watch the machine play itself. The computer, however, is not as smart as you might think. It picks its blocks completely at random. A young child can have fun in this mode without becoming discouraged by an unbeatable opponent. Usually an out-of-memory error results after several rounds, but sometimes the computer actually beats itself.

If you make a mistake typing in the block numbers, just use the DEL key to erase your answer. Type in the number you really want and press RETURN. Notice, too, that the program does not accept numbers for blocks which have already been matched.

Word Match

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C.

```
2 POKE53280,6:POKE53281,1:PRINTCHR$(147)    :rem 67
4 POKE254,0                                 :rem 90
5 GOTO10                                   :rem 207
6 FORP=1TO2E3:NEXT:RETURN                  :rem 187
10 DIMW$(12),W1$(6),C%(2),SQ%(12),SH%(12)   :rem 41
12 FORN=1TO12:READSQ%(N):NEXT               :rem 67
20 O=54272:B=827                            :rem 76
22 S=54272:FORN=STO54295:POKEN,0:NEXT:POKEN,15
                                           :rem 120
24 POKES+5,15:POKES+6,255:POKES+2,0:POKES+3,8
                                           :rem 178
25 C%(1)=2:C%(2)=5                          :rem 87
29 IFPEEK(254)=0THENGOSUB601               :rem 190
30 GOSUB701:PRINT"{CLR}{3 DOWN}O.K., "N$(1)" AND "
   N$(2)"...."                             :rem 106
31 PRINT"{2 DOWN}PLEASE WAIT WHILE THE SCREEN IS S
   ET UP-"                                 :rem 186
32 W$="CARCATBOYHATHITTOPATEEATPITPATGOTHIMHERWHYH
   OWTINILLWHORUNYOUACEBEDINK              :rem 193
33 W$=W$+"AIMARTTOTTIEENDDOGPENWINNEWWONNOWPIGDADM
   OMOFFPALLAPEAREYETOECAPPAN               :rem 78
```

```
34 W$=W$+"NOTTONTENDAYBID"                          :rem 127
35 FORN=1TO50:POKEB+N,0:NEXT:FORN=1TO50             :rem 241
36 Z=INT(RND(1)*50)+1:IFPEEK(B+Z)<>0THEN36:rem 213
37 POKEB+Z,N:NEXT                                   :rem 83
38 FORN=1TO6:W1$(N)=MID$(W$,1+(PEEK(B+N)-1)*3,3):N
   EXT                                              :rem 16
39 C%(1)=2:C%(2)=5                                  :rem 92
40 FORN=1TO12:POKEB+N,0:NEXT                        :rem 6
41 FORN=1TO11STEP2                                  :rem 123
42 Z=INT(RND(1)*12)+1:Y=INT(RND(1)*12)+1            :rem 227
43 IFPEEK(B+Z)<>0ORPEEK(B+Y)<>0ORZ=YTHEN42:rem 138
44 POKEB+Z,N:POKEB+Y,N+1                            :rem 221
45 NEXT                                             :rem 168
47 FORN=1TO12:POKEB+N,INT((PEEK(B+N)-1)/2)+1:W$(N)
   =W1$(PEEK(B+N)):NEXT                             :rem 117
50 GOSUB500                                         :rem 122
70 D$="{HOME}{19 DOWN}":SP$="{39 SPACES}"           :rem 40
100 REM GAME                                        :rem 143
105 X=1                                             :rem 92
110 PRINTD$"{RED}WHICH BLOCKS, "N$(X)"?"            :rem 226
115 POKE53280,C%(X)                                 :rem 5
117 IFN$(X)="64"THENGOSUB1000                       :rem 158
120 PRINTD$"{DOWN}"SP$D$"{DOWN}{PUR}{RVS}BLOCK A?
    {BLK}{OFF}";:POKE198,0                          :rem 190
121 GOSUB901:I=VAL(AN$):GOSUB400:ON(I>12)+2GOTO120
    ,124                                            :rem 161
124 ON(PEEK(B+I)=0)+2GOTO120,135                    :rem 12
125 PRINTD$"{2 DOWN}"SP$D$"{2 DOWN}{BLU}{RVS}BLOCK
     B?{BLK}{OFF}";:POKE198,0                       :rem 105
126 GOSUB901:J=VAL(AN$):GOSUB400:ON(J>12)+2GOTO125
    ,129                                            :rem 178
129 ON(PEEK(B+J)=0)+2GOTO125,130                    :rem 18
130 IFI=JTHEN125                                    :rem 186
131 PRINTD$SP$SP$SP$SP$;:GOTO138                    :rem 240
135 FORN=1TO3:POKESQ%(I)+N+40+O,6:POKESQ%(I)+N+40,
    ASC(MID$(W$(I),N,1))-64:NEXT                    :rem 53
136 ON(N$(X)="64")+2GOTO1040,125                    :rem 173
138 FORN=1TO3:POKESQ%(J)+N+40+O,6:POKESQ%(J)+N+40,
    ASC(MID$(W$(J),N,1))-64                         :rem 194
139 NEXT                                            :rem 220
140 IFPEEK(B+I)=PEEK(B+J)THEN200                    :rem 123
150 PRINTD$SP$D$TAB(15)"{RED}{RVS}NO MATCH-{OFF}":
    POKES+4,65:FORN=1TO30:POKES+1,80                :rem 196
151 POKES+1,80-2*N:NEXT:POKES+1,0:POKES+4,64
                                                    :rem 126
152 GOSUB6:PRINTD$SP$                               :rem 68
153 I$=STR$(I):I$=RIGHT$(I$,LEN(I$)-1):J$=STR$(J):
    J$=RIGHT$(J$,LEN(J$)-1)                         :rem 112
154 POKESQ%(I)+41,32:POKESQ%(J)+41,32:POKESQ%(I)+4
    3,32:POKESQ%(J)+43,32                           :rem 26
```

```
155 FORN=1TOLEN(I$):POKESQ%(I)+41+N,ASC(MID$(I$,N,
    1)):POKESQ%(I)+41+N+O,4:NEXT              :rem 98
156 FORN=1TOLEN(J$):POKESQ%(J)+41+N,ASC(MID$(J$,N,
    1)):POKESQ%(J)+41+N+O,4:NEXT             :rem 103
160 IFX=1THENX=2:GOTO110                     :rem 231
162 X=1:GOTO110                              :rem 100
200 REM RIGHTANS                             :rem 214
205 PRINTD$SP$SP$D$"{15 SPACES}{BLK}{PUR}{RVS}MATC
    H!!!!{OFF}"                              :rem 135
207 FORN=1TO5:POKES+4,65:FORZ=40TO80:POKES+1,Z:NEX
    TZ,N                                     :rem 216
210 POKES+1,0:POKES+4,64                     :rem 126
211 IFX=1THENS1=S1+1                         :rem 185
212 IFX=2THENS2=S2+1                         :rem 189
215 GOSUB6                                    :rem 78
220 GOSUB802                                 :rem 174
235 PRINTD$SP$                                :rem 86
237 CR=CR+1:IFCR=6THEN300                    :rem 242
238 POKEB+I,0:POKEB+J,0                       :rem 90
240 GOTO110                                   :rem 97
300 FORN=1TO5:POKES+4,65:FORZ=80TO30STEP-1:POKES+1
    ,Z:NEXTZ,N                               :rem 107
302 POKES+1,0:POKES+4,64                     :rem 128
305 PRINTD$"THE GAME IS OVER-":GOSUB6        :rem 193
307 IFS1>S2THENPRINTD$SP$D$N$(1)" WINS!!!":rem 159
308 IFS2>S1THENPRINTD$SP$D$N$(2)" WINS!!!":rem 161
309 IFS2=S1THENPRINTD$SP$D$"IT'S A TIE!!!":rem 165
310 GOSUB6:PRINTD$"{DOWN}WANT ANOTHER?(Y/N)":POKE1
    98,0                                     :rem 230
311 GETA$:IFA$="N"THENPRINT"{CLR}{BLU}":POKEBK,27:
    END                                       :rem 37
312 IFA$="Y"THENRUN10                        :rem 233
314 GOTO311                                  :rem 102
400 POKES+4,33:POKES+1,50:FORP=1TO20:NEXT:POKES+1,
    0:POKES+4,32:RETURN                       :rem 71
500 REM DRAW SCREEN                          :rem 103
501 PRINT"{CLR}":FORN=1TO4:PRINTTAB(9)"{BLK} {RVS}
    {19 SPACES}"                             :rem 188
502 FORZ=1TO3                                 :rem 28
503 PRINTTAB(9)" {RVS} {OFF}{5 SPACES}{RVS} {OFF}
    {5 SPACES}{RVS} {OFF}{5 SPACES}{RVS} {OFF}":NE
    XTZ,N                                    :rem 167
504 PRINTTAB(9)" {RVS}{19 SPACES}":PRINT"{HOME}
    {2 DOWN}{PUR}"                           :rem 185
505 FORN=1TO9STEP3                           :rem 136
506 PRINTTAB(12)NSPC(3)N+1SPC(3)N+2           :rem 42
507 PRINT"{2 DOWN}":NEXT                       :rem 8
508 PRINT"{13 RIGHT}10{4 RIGHT}11{4 RIGHT}12"
                                             :rem 245
509 RETURN                                   :rem 126
```

```
600 REM INTRO                                          :rem 6
601 FORZ=1TO12:SH%(Z)=Ø:NEXT:FORZ=1TO12              :rem 198
602 X=INT(RND(1)*12)+1:IFSH%(X)<>ØTHEN6Ø2             :rem 91
603 SH%(X)=Z:NEXTZ                                   :rem 1Ø8
604 GOSUB5Ø1:POKES+4,65:FORZ=1TO11STEP2              :rem 19Ø
605 I=SH%(Z):J=SH%(Z+1)                               :rem 2Ø
606 X=1:Q=C%(X):C%(X)=VAL(MID$("25",INT(RND(1)*2)+
    1,1)):IFQ=C%(X)THEN6Ø6                              :rem 7
607 POKES+1,RND(1)*5Ø+1Ø:GOSUB8Ø2:POKES+1,Ø
                                                     :rem 24Ø
608 NEXT                                             :rem 221
609 I=1:J=12:C%(X)=1:GOSUB8Ø2                        :rem 135
619 PRINT"{HOME}{3 DOWN}"TAB(11)"{BLK}WORD":POKES+
    1,3Ø:FORP=1TO1ØØ:NEXT                            :rem 132
620 PRINT"{HOME}{15 DOWN}"TAB(23)"MATCH":POKES+1,2
    Ø                                                 :rem 95
621 FORP=1TO1ØØ:NEXT:POKES+1,Ø:POKES+4,64             :rem 16
622 GOSUB6:GOSUB6:POKE254,255                         :rem 12
623 PRINT"{CLR}{2 DOWN}WOULD YOU LIKE INSTRUCTIONS
    ? (Y/N)":POKE198,Ø                                :rem 8Ø
624 GETA$:IFA$="Y"THENGOSUB15Ø1:GOTO63Ø              :rem 1Ø7
625 IFA$="N"THEN63Ø                                   :rem 4Ø
626 GOTO624                                          :rem 115
630 RETURN                                           :rem 121
700 REM GET NAMES                                    :rem 2Ø7
701 DIMN$(2):PRINT"{BLU}{CLR}NAMES, PLEASE!"
                                                     :rem 159
702 PRINT"{HOME}{15 DOWN}TO PLAY AGAINST THE COMPU
    TER, ENTER"                                       :rem 86
704 PRINT" '64' AS A PLAYER."                        :rem 244
706 PRINT"{HOME}{DOWN}":FORN=1TO2:PRINT"{DOWN}PLAY
    ER"N;:INPUTN$(N):NEXT:RETURN                      :rem 36
800 REM PAINT SQUARES                                 :rem 28
802 Q=SQ%(I):R=SQ%(J)                                :rem 184
804 FORN=1TO3                                         :rem 21
806 FORW=QTOQ+4:POKEW+O,C%(X):POKEW,16Ø:NEXT:Q=Q+4
    Ø:NEXT                                            :rem 81
808 FORN=1TO3                                         :rem 25
810 FORW=RTOR+4:POKEW+O,C%(X):POKEW,16Ø:NEXT:R=R+4
    Ø:NEXT:RETURN                                    :rem 1Ø6
900 REM INPUT ROUTINE                                 :rem 51
901 POKE198,Ø:AN$=""                                  :rem 53
902 GETA$:IFA$=""THEN9Ø2                              :rem 89
903 IFA$=CHR$(13)THEN92Ø                              :rem 77
904 IFA$=CHR$(2Ø)ANDLEN(AN$)>ØTHENGOSUB931:rem 242
905 IFLEN(AN$)>1THEN9Ø2                               :rem 73
906 IFA$<"Ø"ORA$>"9"THEN9Ø2                          :rem 2Ø6
907 PRINTA$;:AN$=AN$+A$:GOTO9Ø2                       :rem 72
920 IFAN$=""THEN9Ø2                                   :rem 4Ø
922 RETURN                                           :rem 125
```

```
930 REM DELETE KEY                                    :rem 28
931 AN$=LEFT$(AN$,LEN(AN$)-1)                         :rem 77
933 PRINT"{LEFT} {LEFT}";                            :rem 229
939 RETURN                                           :rem 133
1000 REM 64 PLAYS                                    :rem 152
1005 I=INT(RND(1)*12)+1:ON(PEEK(B+I)=0)+2GOTO1005,
     135                                             :rem 185
1040 J=INT(RND(1)*12)+1:IFJ=ITHEN1040                :rem 100
1050 IFPEEK(B+J)=0THEN1040                           :rem 227
1060 PRINTD$SP$D$"64 PICKS"I"AND"J"{LEFT}."
                                                     :rem 204
1065 GOSUB6:GOTO138                                  :rem 145
1500 REM INSTRUCTIONS                                 :rem 95
1501 PRINTCHR$(14)CHR$(147)                          :rem 249
1502 PRINT"{BLK}{2 SPACES}WORDS WILL BE HIDDEN BEH
     IND BLOCKS"                                      :rem 11
1503 PRINT"ON THE SCREEN. ENTER THE BLOCK NUMBER,"
                                                      :rem 69
1504 PRINT"AND THE WORD WILL BE UNCOVERED. YOU"
                                                     :rem 160
1505 PRINT"MAY UNCOVER TWO WORDS DURING EACH TURN.
     "                                               :rem 127
1506 PRINT"{DOWN}{2 SPACES}IF THE TWO WORDS MATCH,
      THE BLOCKS"                                     :rem 95
1507 PRINT"WILL BE FILLED WITH YOUR COLOR, AND YOU
     "                                                :rem 19
1508 PRINT"HAVE ANOTHER TURN. IF THEY DON'T MATCH,
     "                                               :rem 137
1509 PRINT"THE WORDS ARE COVERED UP AGAIN AND THE"
                                                     :rem 169
1510 PRINT"OTHER PLAYER GETS A TURN."                :rem 210
1512 PRINT"{DOWN}{3 SPACES}THE GAME IS OVER WHEN A
     LL OF THE"                                      :rem 127
1513 PRINT"BLOCKS ARE COLORED IN. THE PLAYER WHO"
                                                      :rem 26
1514 PRINT"HAS FOUND THE MOST MATCHES IS THE
     {19 SPACES}{DOWN}** WINNER **"                  :rem 166
1515 GOSUB1600:                                       :rem 77
1516 PRINT"{CLR}{2 DOWN}{3 SPACES}TELL THE COMPUTE
     R WHICH BLOCK TO"                               :rem 213
1517 PRINT"UNCOVER BY TYPING A NUMBER (1-12) AND"
                                                      :rem 38
1518 PRINT"PRESSING RETURN. IF YOU MAKE A MISTAKE,
     "                                               :rem 190
1520 PRINT"YOU MAY USE THE 'DEL' KEY TO CHANGE"
                                                      :rem 32
1521 PRINT"YOUR ANSWER. PRESS THE {RVS}RETURN{OFF}
      KEY WHEN"                                      :rem 116
1522 PRINT"YOU ARE FINISHED WITH YOUR ANSWER."
                                                      :rem 67
```

```
1523 PRINT"{2 DOWN}{4 SPACES}IF YOU CAN'T FIND ANO
     THER PERSON"                              :rem 32
1524 PRINT"WITH WHOM TO PLAY THE GAME, YOU MAY"
                                              :rem 248
1525 PRINT"PLAY AGAINST THE COMPUTER. JUST ENTER"
                                              :rem 157
1526 PRINT"'64' AS ONE OF THE PLAYERS' NAMES."
                                               :rem 44
1527 PRINT"THE COMPUTER ISN'T VERY SMART, BUT YOU"
                                              :rem 182
1528 PRINT"CAN HAVE FUN PRACTICING."          :rem 146
1529 GOSUB1600:PRINTCHR$(147)CHR$(142):RETURN
                                              :rem 208
1600 PRINT"{3 DOWN}TOUCH {RVS}RETURN{OFF} TO CONTI
     NUE....":POKE198,0                       :rem 121
1602 GETA$:IFA$<>CHR$(13)THEN1602             :rem 100
1604 RETURN                                   :rem 171
2000 DATA1115,1121,1127,1275,1281,1287,1435,1441,1
     447,1595,1601,1607                        :rem 86
```

Connect the Dots

Janet Arnold

"Connect the Dots" is an entertaining graphics program for young children who can locate numbers and letters on the keyboard. You can even add new drawings of your own.

As teachers at a small private school, my husband and I saw many children anxious to get their hands on our computer whenever we brought it to class. Unlike many adults, who are hesitant to use it or even refuse to touch it altogether, the children jockeyed for their turn at even the dullest programs we loaded.

I wrote "Connect the Dots" to provide my own children and my preschool/kindergarten students with a game that could entertain while reinforcing their skills at the same time.

Making Dots into Pictures

Here's how it works. The child is given a four-item menu from which to choose the picture he or she wishes to draw. The greater the number of dots, the longer it takes to complete the picture.

A grid appears on the screen. Some of the squares contain markings. Tell the child to look for the solid dot, because that's what must be matched with the coordinates. When the prompt *Number?* appears at the top, show the child how to press the correct number coordinate and hit RETURN. Answering the next prompt, *Letter?*, will probably take longer unless the child is familiar with the keyboard.

A wrong number-letter combination is answered with a low "uh-oh" sound and the words *Try again.*

After a correct answer, the computer draws a line connecting the dots and plays an amusing sound effect. A short timing loop delays this just long enough for the child to look from the keyboard back to the screen to enjoy this reward.

The finished drawing is accompanied by a short tune and the remark, *Good job! Draw again?* Hitting a *Y* calls up the menu again. An *N* ends the program.

Working with Your Child

When introducing this activity to a child, a few additional explanations may be necessary. Be sure to explain the difference between the number 0—point out the slash—and the letters O and Q.

A tot whose visual discrimination is immature might reverse letters. Connect the Dots can give that child enjoyable practice in overcoming this. If you notice a child confusing 7 and L, for instance, ask, "Is that line walking on the ceiling or on the floor?"

Of course, preschoolers and some kindergartners who are still learning their numbers and letters will enjoy naming them aloud to you.

Children with short attention spans should try the pictures with fewer dots. Even then, be prepared to help them along or to complete it for them. This isn't necessarily bad, because the time spent with children at the computer can enrich your relationship and will tell them that their activities are important to you.

There's no time limit in Connect the Dots, so don't rush your child. This will be a welcome relief to the child who equates computers with tense, timed, shoot-or-be-shot action.

If some children's eyes have trouble following the grid from the dot to the coordinates, show them how to trace with their fingers directly on the screen.

Details of the Program

It's important to type this program exactly as shown.

The fourth selection on the menu is a heart inscribed with my children's names. Substitute your own message by changing lines 780–800.

Following is a line-by-line program description, giving the starting line number of each section:

Line	Function
100	Title and instructions
300	Menu
370	Draw grid
440	Search DATA for starting point of chosen picture

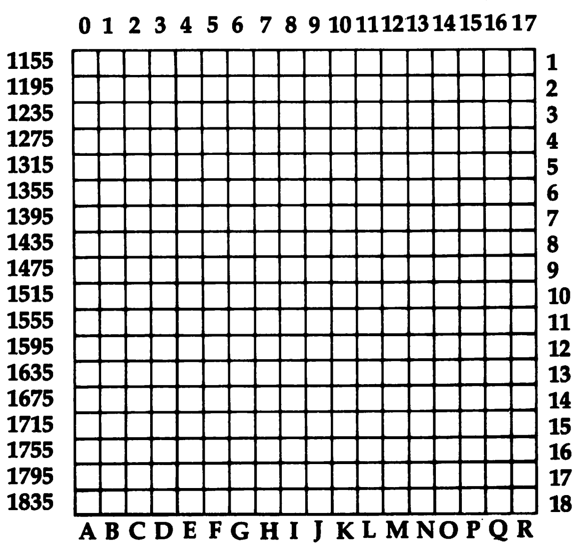
490	Read four pieces of DATA per dot and POKE dot
520	Ask for dot's coordinates
550	Response for wrong answer
610	Response for right answer
650	Set up butterfly
690	Set up mushroom
730	Set up dog
770	Set up heart
830	Response for completed picture
1000	DATA for butterfly
1090	DATA for mushroom
1140	DATA for horse
1200	DATA for heart

Designing Your Own Pictures

Part of the fun of this program is designing your own pictures. My five-year-old, Jonathan, contributed the mushroom found in Connect the Dots by coloring in squares of graph paper.

To substitute a picture of your own, design one using the accompanying grid. Remember that most of your design should consist of a continuous line as in dot-to-dot pictures. Anything else must be POKEd in when the picture is first set up.

Grid for Designing Pictures



For the purposes of this article, let's assume that you've drawn a clown to replace the dog in the listed program. Substitute the title CLOWN for HORSE in line 330. This changes the menu to read C–CLOWN.

Lines 730–750 POKE in the horse's tail and a starting square (SQ). Use these lines to POKE in your clown's nose, for example. (Hint: Since children are always asked the coordinates of a *solid* dot, use an open O or you will confuse them.)

To compute the screen memory location of the nose, add the four-digit row number to the left of the grid to the column number above the grid. This same number + CD is your color memory location.

POKE in your starting square—use screen code 160, a reversed space—and assign SQ the value of the screen memory location of that starting square.

Now just figure your DATA. The computer reads four pieces of data per dot: screen memory location (A), color of the line to be drawn (B), number-letter coordinates of the dot (E$), and the direction that the line will travel to reach the dot (S). Figure each as follows:

First, compute the screen memory location of the dot as explained earlier.

The second number is the color code of the line to be drawn. The color code is always the number of the color's computer key minus 1 (black=0, red=1, and so on). Appendix G in the *Commodore User's Guide,* the manual that came with your computer, lists these color values.

Third, look at your grid to find the number-letter coordinates of the dot. The number comes first and is found on the right side of the grid. Follow this with the letter. Do not separate the number and letter with a space.

The last number is a STEP value. This number tells the computer in which direction the line should be drawn. For instance, a line moving from left to right travels one space at a time, so its STEP value is 1. From right to left, the line moves backwards one space at a time, making its STEP value −1. A line traveling diagonally up to the left has a STEP value of −41 on the Commodore 64 since the computer skips back 41 spaces before POKEing the next square.

Use this diagram to figure STEP values.

STEP Values

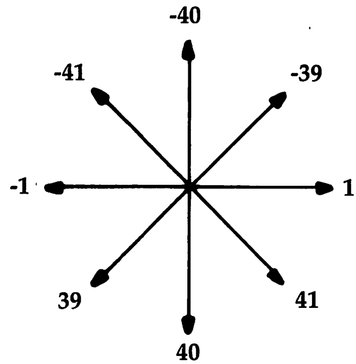


Figure each dot's DATA in the same manner. Separate each piece of DATA with a comma. You must insert your new DATA into the proper line numbers, so check the program explanation listed earlier. Since you are replacing the horse with your clown, your DATA will go in lines 1140–1180. Be sure to leave the first piece of DATA, C, in line 1140. This is the DATA that the computer searches for to set the DATA pointer. Notice that the last set of DATA for every drawing is 0, 0, 0, 0. Make sure this also ends any new drawings you may add.

Connect the Dots

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C.

```
100 REM TITLE PAGE                                    :rem 20
110 PRINT"{CLR}{7}":POKE53280,0:POKE53281,0:rem 31
120 PRINTSPC(10)"QQQ{3 SHIFT-SPACE}QQ
    {2 SHIFT-SPACE}QQQQQ{2 SHIFT-SPACE}QQ" :rem 40
130 PRINTSPC(10)"Q{SHIFT-SPACE} Q{SHIFT-SPACE}Q
    {2 SHIFT-SPACE}Q{2 SHIFT-SPACE} Q{SHIFT-SPACE}
     {SHIFT-SPACE}Q{2 SPACES}Q"                   :rem 180
140 PRINTSPC(10)"Q {SHIFT-SPACE}Q{SHIFT-SPACE}Q
    {2 SHIFT-SPACE}Q{2 SHIFT-SPACE} Q{SHIFT-SPACE}
     {2 SHIFT-SPACE}Q"                             :rem 132
150 PRINTSPC(10)"Q {SHIFT-SPACE}Q{SHIFT-SPACE}Q
    {2 SHIFT-SPACE}Q{2 SHIFT-SPACE} Q{SHIFT-SPACE}
    {2 SPACES}{2 SHIFT-SPACE}Q"                    :rem 133
160 PRINTSPC(10)"Q{SHIFT-SPACE} Q{SHIFT-SPACE}Q
    {2 SHIFT-SPACE}Q{2 SHIFT-SPACE} Q{SHIFT-SPACE}
    {2 SPACES}Q{2 SHIFT-SPACE}Q"                    :rem 87
```

```
170 PRINTSPC(10)"QQQ{2 SHIFT-SPACE} QQ
    {3 SHIFT-SPACE} Q {SHIFT-SPACE} {SHIFT-SPACE}Q
    Q"                                       :rem 233
180 L=1114:C=55386:CD=54272:WV=54276         :rem 220
190 A$="Z":POKE54296,15:POKE54277,22:POKE54278,165
    :GOSUB840                                :rem 114
200 :                                        :rem 204
210 REM INSTRUCTIONS                          :rem 44
220 PRINTSPC(13)"{2 DOWN}{WHT}INSTRUCTIONS:"
                                              :rem 22
230 PRINTSPC(9)"{DOWN}WHEN THE GRID APPEARS,":PRIN
    TSPC(10)"FIND THE SOLID DOT."            :rem 141
240 PRINTSPC(9)"{DOWN}TYPE THE NUMBER OF THE":PRIN
    TSPC(10)"ROW AND HIT RETURN."            :rem 171
250 PRINTSPC(6)"{DOWN}THEN TYPE THE LETTER OF THE"
                                             :rem 126
260 PRINTSPC(9)"COLUMN AND HIT RETURN."      :rem 109
270 PRINTSPC(10)"{2 DOWN}{RVS}HIT ANY KEY TO PLAY.
    ":POKE198,0                               :rem 90
280 GETS$:IFS$=""THEN280                     :rem 123
290 :                                        :rem 213
300 REM DRAW SELECTION                        :rem 75
310 POKE53281,6:PRINT"{CLR}"SPC(6)"{5 DOWN}⁅3⁆WH
    AT WOULD YOU LIKE TO DRAW?"               :rem 83
320 PRINTSPC(8)"⁅7⁆{3 DOWN}A - BUTTERFLY (22 DOT
    S)"                                      :rem 192
330 PRINTSPC(9)"{DOWN}B - MUSHROOM (12 DOTS)":PRIN
    TSPC(11)"{DOWN}C - HORSE (20 DOTS)"       :rem 82
340 PRINTSPC(10)"{DOWN}D - HEART (10 DOTS)":POKE19
    8,0                                       :rem 73
350 GETA$:IFA$<"A"ORA$>"D"THEN350             :rem 95
360 :                                        :rem 211
370 REM DRAW BOARD                            :rem 20
380 PRINT"{CLR}⁅3⁆"SPC(11)"{2 DOWN}ABCDEFGHIJKLM
    NOPQR{HOME}"                             :rem 210
390 FORRH=1TO18:FORT=1TO18:POKEL+T+RH*40,79:POKEC+
    T+RH*40,14:NEXT:NEXT                     :rem 170
400 PRINTSPC(11)"⁅3⁆{20 DOWN}ABCDEFGHIJKLMNOPQR"
                                              :rem 19
410 PRINT"{HOME}{3 DOWN}⁅7⁆";              :rem 129
420 FORI=1TO18:PRINTSPC(8)RIGHT$(STR$(I),2)SPC(19)
    "⁅G⁆"RIGHT$(STR$(I),2):NEXT           :rem 137
430 :                                        :rem 209
440 REM FIND DATA                            :rem 183
450 RESTORE                                  :rem 189
460 READB$:IFB$<>A$THEN460                   :rem 243
470 ONASC(A$)-64GOTO650,690,730,770          :rem 139
480 FORT=1TO500:NEXT                         :rem 246
490 READA,B,E$,S:IFA=0THEN830                :rem 189
```

```
500 POKEA,81:POKEA+CD,B                          :rem 100
510 PRINT"{HOME}{39 SPACES}"                     :rem 122
520 PRINT"{7}{HOME} (←) NUMBER";:GOSUB930:N$=IN
    $                                            :rem 195
530 PRINT"{HOME}"SPC(20)"(↑) LETTER";:GOSUB930:L$=
    IN$                                          :rem 11
540 IFE$=N$+L$THEN610                            :rem 161
550 PRINT"{HOME}{BLK}{15 SPACES}TRY AGAIN
    {10 SPACES}"                                 :rem 109
560 POKECD,48:POKECD+1,11:POKEWV,33:POKEWV,32
                                                 :rem 18
570 FORT=1TO400:NEXT:POKECD,195:POKECD+1,16:POKEWV
    ,33:POKEWV,32                                :rem 222
580 FORT=1TO400:NEXT                             :rem 246
590 FORT=1TO1200:NEXT:GOTO510                    :rem 47
600 :                                            :rem 208
610 FORT=1TO700:NEXT:FORT=1TO18:POKESQ,160:POKESQ+
    CD,B:IFSQ=ATHEN630                           :rem 146
620 SQ=SQ+S:NEXT                                 :rem 20
630 POKEWV,17:FORZ=9TO26:POKECD+1,Z:POKECD,0:NEXT:
    POKEWV,16:GOTO480                            :rem 84
640 :                                            :rem 212
650 POKE1242,77:POKE1242+CD,0:POKE1244,78:POKE1244
    +CD,0                                        :rem 126
660 POKE1283,160:POKE1283+CD,5                   :rem 166
670 SQ=1283:GOTO480                              :rem 91
680 :                                            :rem 216
690 POKE1563,160:POKE1563+CD,4:POKE1564,160:POKE15
    64+CD,4                                      :rem 241
700 POKE1717,160:POKE1717+CD,5                   :rem 165
710 SQ=1717:GOTO480                              :rem 88
720 :                                            :rem 211
730 POKE1436,74:POKE1436+CD,0:POKE1437,75:POKE1437
    +CD,0                                        :rem 137
740 POKE1397,85:POKE1397+CD,0                    :rem 130
750 POKE1208,160:POKE1208+CD,2:SQ=1208:GOTO480
                                                 :rem 146
760 :                                            :rem 215
770 PRINT"{HOME}{7 DOWN}"                        :rem 249
780 PRINTSPC(16)"{GRN}MATTHEW"                   :rem 70
790 PRINTSPC(16)"{2 DOWN}{7}JONATHAN"            :rem 30
800 PRINTSPC(17)"{2 DOWN}{3}EMILY"               :rem 64
810 POKE1283,160:POKE1283+CD,2:SQ=1283:GOTO480
                                                 :rem 152
820 :                                            :rem 212
830 PRINT"{HOME}{10 SPACES}GOOD JOB! DRAW AGAIN?
    {3 SPACES}"                                  :rem 113
840 READB$:IFB$<>"Z"THEN840                      :rem 48
850 READPL,PH,D:IFPL=-1ANDA$="Z"THENPOKEWV,0:RETUR
    N                                            :rem 29
```

```
860 IFPL=-1THENPOKEWV,Ø:GOTO89Ø                   :rem 223
870 POKECD,PL:POKECD+1,PH:POKEWV,33:FORT=1TOD*75:N
    EXT:POKEWV,32                                  :rem 85
88Ø GOTO85Ø                                       :rem 118
89Ø GETY$:IFY$<>"Y"ANDY$<>"N"THEN89Ø              :rem 135
9ØØ IFY$="Y"THEN31Ø                                :rem 66
91Ø :                                             :rem 212
92Ø PRINT"{CLR}";:END                              :rem 75
93Ø PRINT"? ";:IN$=""                              :rem 93
94Ø PRINT"{RVS} {OFF}{LEFT}";                     :rem 234
95Ø GETA$:IFA$=""THEN94Ø                           :rem 94
96Ø ZL=LEN(IN$):IFA$=CHR$(2Ø)ANDZLTHENPRINTA$;:IN$
    =LEFT$(IN$,ZL-1)                               :rem 3Ø
97Ø IFA$=CHR$(13)ANDZLTHENPRINT" ":RETURN          :rem 26
98Ø IF(A$<"Ø"ORA$>"R")OR(A$>"9"ANDA$<"A")ORLEN(IN$
    )=2THEN95Ø                                     :rem 67
99Ø PRINTA$;:IN$=IN$+A$:GOTO94Ø                    :rem 92
1ØØØ :                                            :rem 251
1Ø1Ø DATA A,14Ø3,5,7I,4Ø,1247,2,3M,-39            :rem 119
1Ø2Ø DATA 1249,2,3O,1,1331,2,5Q,41,1491,2,9Q,4Ø,15
     69,2,11O,39,1651,7,13Q,41                    :rem 236
1Ø3Ø DATA 1731,7,15Q,4Ø,177Ø,7,16P,39,1767,7
                                                  :rem 189
1Ø35 DATA 16M,-1,16Ø3,7,12I,-41,1759,7,16E,39
                                                  :rem 229
1Ø4Ø DATA 1756,7,16B,-1,1715,7,15A,-41,1635,7,13A,
     -4Ø,1557,7,11C,-39                            :rem 69
1Ø5Ø DATA 1475,2,9A,-41                           :rem 115
1Ø6Ø DATA 1315,2,5A,-4Ø,1237,2,3C,-39,1239,2,3E,1,
     14Ø3,2,7I,41,1683,5,14I,4Ø                   :rem 216
1Ø7Ø DATA Ø,Ø,Ø,Ø                                  :rem 38
1Ø8Ø :                                              :rem 3
1Ø9Ø DATA B,1722,5,15H,1,1562,4,11H,-4Ø,1559,4,11E
     ,-1,1519,4,1ØE,-4Ø                            :rem 75
11ØØ DATA 1441,4,8G,-39                           :rem 118
111Ø DATA 1446,4,8L,1,1528,4,1ØN,41,1568,4,11N,4Ø,
     1565,4,11K,-1,1725,4,15K,4Ø                   :rem 53
112Ø DATA 173Ø,5,15P,1,1722,5,15H,-1,Ø,Ø,Ø,Ø
                                                  :rem 118
113Ø :                                            :rem 255
114Ø DATA C,1364,2,6J,39,1359,2,6E,-1,1398,2,7D,39
     ,1598,2,12D,4Ø,168Ø,2,14F,41                 :rem 111
115Ø DATA 1681,2,14G,1,1641,2                     :rem 154
1155 DATA 13G,-4Ø,16ØØ,2,12F,-41,156Ø,2,11F,-4Ø,15
     64,2,11J,1                                   :rem 168
116Ø DATA 1687,2,14M,41,1688,2,14N,1,16Ø8,2
                                                  :rem 123
1165 DATA 12N,-4Ø,1567,2,11M,-41,14Ø7,2,7M,-4Ø
                                                   :rem 14
```

```
1170 DATA 1329,2,5O,-39,1331,0,5Q,1,1291,0,4Q,-40,
     1290,0,4P,-1,1208,2,2N,-41              :rem 244
1180 DATA 0,0,0,0                            :rem 40
1190 :                                        :rem 5
1200 DATA D,1160,2,1F,-41,1157,2,1C,-1,1235,2,3A,3
     9,1475,2,9A,40,1803,2,17I,41             :rem 56
1210 DATA 1491,2,9Q,-39,1251,2,3Q,-40,1169,2,1O,-4
     1,1166,2,1L,-1,1283,2,4I,39              :rem 47
1220 DATA 0,0,0,0                            :rem 35
1230 :                                        :rem 0
1240 DATA Z,195,16,3, 31,21,1, 30,25,2, 135,33,2
                                             :rem 169
1250 DATA 30,25,2, 31,21,2, 195,16,2, 31,21,2, 30,
     25,3, 31,21,1                           :rem 233
1260 DATA 195,16,2                           :rem 115
1270 DATA 143,12,2, 195,16,1                  :rem 36
1280 DATA 0,0,3, 195,16,1, -1,0,0            :rem 205
```

Word Scramble

Mike Salman

Match wits with an opponent in this game as you play against time. For two or more players.

"Word Scramble" is a bit different from other jumbling games you might have played. Instead of the computer giving you letters to unscramble, your opponent chooses the word you'll be trying to piece back together. Because the players select the words, the variety is almost limitless.

A Three-Minute Puzzle

As soon as you've typed in and saved Word Scramble (make sure you use "The Automatic Proofreader" program, found in Appendix C, to help you type in the game), you're ready to try to stump your opponent. Although the game is designed for two players, you can make up teams if there are more who want to play. The computer asks for the players' names and then tells player one to enter a word (maximum of ten letters). If you enter a word longer than ten letters, a message will remind you that it's not allowed. Just move the blinking square (the cursor) to the end of the word, and press the DEL key to erase the word. Then you can type in another, this one less than ten letters.

When the word has been scrambled, player two presses the space bar to see the jumbled letters. He or she has only three minutes to put the letter back into order.

At the top of the screen, you'll see a display of the elapsed time. Below the mixed-up letters, you should see a bar. That's where you'll type the first letter of the word. If you type the wrong letter, you hear a buzz. Type the right one and you hear a beep; the letter then appears on the screen.

A Ten-Point Penalty

If you find the word within the three-minute time limit and have made no wrong guesses, you're rewarded with 50 points. For every wrong guess that you make, you *lose* ten points. A scoreboard is displayed every second turn so you'll know when both players have played an equal number of turns. When you want to quit playing, just press the RUN/STOP and RESTORE keys at the same time.

Word Scramble

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C.

```
1 POKE53280,6:POKE53281,1                         :rem 141
5 SN=54272                                        :rem 23
6 POKESN+24,15:POKESN+5,17:POKESN+6,240:POKESN,100
                                                   :rem 27
10 PRINT"{CLR}":PRINT"{RED}{9 DOWN}{13 RIGHT}WORD
   {SPACE}SCRAMBLE"                               :rem 131
20 GOSUB1000:PRINT"{CLR}"                          :rem 65
25 PRINT"{RED}{2 DOWN}EACH PLAYER TAKES A TURN ENT
   ERING A{5 SPACES}COMMON ";                     :rem 247
30 PRINT"WORD (A MAXIMUM OF 10 LETTERS)." :rem 103
35 PRINT"{DOWN}THE COMPUTER WILL THEN SCRAMBLE THE
    WORD";                                        :rem 162
40 PRINT"AND PRINT IT."                            :rem 96
45 PRINT"{DOWN}YOU HAVE THREE MINUTES TO FIND IT."
                                                   :rem 152
50 PRINT"{DOWN}IF FOUND WITHIN THE ALLOTTED TIME,
   {SPACE}YOU"                                    :rem 183
55 PRINT"WILL BE GIVEN 50 POINTS."                 :rem 227
60 PRINT"{DOWN}FOR EVERY WRONG GUESS THAT YOU MAKE
   , YOUWILL LOSE 10 POINTS.{BLU}"                 :rem 57
65 PRINT"{3 DOWN}{7 RIGHT}{RVS}{PUR}PRESS SPACE BA
   R WHEN READY{OFF}"                             :rem 239
70 IFPEEK(197)<>60THEN70                          :rem 131
75 POKE198,0                                      :rem 153
80 PRINT"{CLR}{4 DOWN}{GRN}PLAYER # 1'S NAME{BLU}"
   :INPUTP$(0)                                    :rem 200
85 PRINT"{3 DOWN}{PUR}PLAYER # 2'S NAME{BLU}":INPU
   TP$(1)                                         :rem 169
90 PRINT"{CLR}{16 DOWN}{RED}";P$(C);", ENTER WORD
   {SPACE}TO BE SCRAMBLED:{OFF}{BLU}"              :rem 67
92 W$="":INPUTW$:IFW$=""THENPRINT"{UP}";:GOTO92
                                                   :rem 27
93 IFW$="QUIT" THEN 410                           :rem 254
94 V1=LEN(W$)                                     :rem 220
95 IFV1>10THENPRINT"{RVS}{GRN}NO MORE THAN 10 LETT
   ERS PLEASE{OFF}{BLU}":GOSUB 990:GOTO90 :rem 117
96 FOR K=1 TO V1:V2$=MID$(W$,K,1):V2=ASC(V2$)
                                                   :rem 205
97 IFV2<65ORV2>90THENPRINT"{RVS}{GRN}LETTERS ONLY
   {SPACE}PLEASE{OFF}{BLU}":GOSUB 990 :GOTO 90
                                                   :rem 54
```

```
98 NEXT                                          :rem 176
100 GOSUB200                                     :rem 163
110 GOSUB300                                     :rem 165
120 T(C)=T(C)+S(C)                               :rem 178
130 GOSUB400:FORI=1TO10:B$(I)="":NEXT            :rem 184
140 GOTO90                                        :rem 55
200 FORI=1TOLEN(W$)                              :rem 126
210 A$(I)=MID$(W$,I,1)                           :rem 107
220 NEXT                                         :rem 211
230 C$="":FORI=1TOLEN(W$)                        :rem 163
240 R=INT(RND(1)*LEN(W$)+1)                      :rem 248
250 IFB$(R)<>""THEN240                           :rem 178
260 B$(R)=A$(I)                                  :rem 221
270 NEXT                                         :rem 216
271 FORI=1TOLEN(W$):C$=C$+B$(I):NEXT             :rem 111
272 IFC$=W$ANDLEN(W$)<>1THENFORI=1TOLEN(W$):B$(I)=
    "":NEXT:GOTO230                              :rem 201
275 PRINT"{CLR}{5 DOWN}{8 SPACES}{RVS}{RED}WORD HA
    S BEEN SCRAMBLED.{OFF}{BLU}"                  :rem 35
280 POKE 198,0:PRINT"{6 DOWN}{7 SPACES}{GRN}PRESS
    {SPACE}SPACE BAR WHEN READY{BLU}"            :rem 234
285 IFPEEK(197)<>60THEN285                       :rem 243
290 PRINT"{CLR}{5 DOWN}{15 RIGHT}";               :rem 66
295 FORI=1TOLEN(W$):PRINT"{RED}";B$(I);:NEXT
                                                 :rem 162
298 POKE198,0:RETURN                             :rem 234
300 X=95:S(C)=50                                  :rem 89
310 TI$="000000"                                 :rem 246
320 PRINT:PRINT:PRINT:PRINT                      :rem 119
325 SC=1399:CC=SC+54272                            :rem 5
330 FORI=1TOLEN(W$)                              :rem 130
335 POKESC,99:POKECC,2                            :rem 75
340 GETC$                                        :rem 222
350 PRINT"{HOME}{RVS}{9 RIGHT}"MID$(TI$,4,1)"
    {OFF}MINUTES{2 SPACES}{RVS}"RIGHT$(TI$,2)"
    {OFF}SECONDS"                                :rem 100
355 IFTI$="000300"THENGOSUB500:GOTO390           :rem 228
360 IFC$=""THEN340                               :rem 214
365 PRINT"{4 DOWN}"                              :rem 179
370 IFC$=A$(I)THENPRINTTAB(X)A$(I);:BY=50:LN=50:GO
    SUB600:GOTO380                               :rem 141
375 IFS(C)<10THENGOSUB550:GOTO390                 :rem 10
378 IFC$<>A$(I)THENS(C)=S(C)-10:BY=20:LN=120:GOSUB
    600:GOTO335                                   :rem 79
380 X=X+1:SC=SC+1:CC=CC+1:NEXT                    :rem 59
390 RETURN                                       :rem 124
400 IFC<>1THENC=1:RETURN                          :rem 11
410 PRINT"{CLR}{5 DOWN}{17 RIGHT}{RED}{RVS}SCORES
    {OFF}{BLU}"                                  :rem 233
420 PRINT"{17 RIGHT}E6 T3"                        :rem 38
```

```
430 PRINT"{DOWN}{10 RIGHT}"P$(0);TAB(25);P$(1)
                                                :rem 139
440 PRINT"{9 RIGHT}"T(1);TAB(24);T(0)          :rem 29
445 PRINT "{9 DOWN}{13 RIGHT}PRESS {RVS}Q{OFF} TO
    {SPACE}QUIT"                                :rem 232
447 PRINT "{6 RIGHT}OR ANY OTHER KEY TO CONTINUE"
                                                :rem 26
450 C=0:GET R$:IF R$="" THEN 450               :rem 97
455 IF R$="Q" THEN END                         :rem 123
460 RETURN                                     :rem 122
500 PRINT"{CLR}{4 DOWN}{12 RIGHT}{RVS}{RED}YOUR TI
    ME IS UP{OFF}{BLU}"                         :rem 55
510 PRINT"{2 DOWN}{10 RIGHT}WORD WAS "W$".":S(C)=0
                                                :rem 77
520 FORT=1TO5000:NEXT:RETURN                   :rem 59
550 PRINT"{RVS}{RED}{2 DOWN}{9 RIGHT}YOU RAN OUT O
    F POINTS{OFF}{BLU}"                         :rem 185
560 PRINT"{2 DOWN}{PUR}{10 RIGHT}WORD WAS {BLU}"W$
    "."                                         :rem 127
570 FORT=1TO2000:NEXT                          :rem 35
580 RETURN                                     :rem 125
600 POKESN+1,BY:POKESN+4,33:FORQQ=1TOLN:NEXT:POKES
    N+4,32:RETURN                               :rem 127
990 FOR DELAY=1 TO 500:NEXT                    :rem 23
1000 FORBY=50TO20STEP-1:LN=20:GOSUB600:NEXT:FORI=1
     TO500:NEXT                                 :rem 73
1010 RETURN                                    :rem 162
```

Turtle Graphics Interpreter

Irwin Tillman

This comprehensive three-program package gives your 64 full turtle-graphics capabilities. It's an excellent learning tool for children, and it offers a new graphics capacity for all ages. For disk or tape users.

Turtle geometry is fast becoming the first exposure to computers for many children. Instead of printing their names on the screen, they are more likely drawing squares and triangles. While such facilities are generally found with specific languages (such as PILOT and Logo), the concept of turtle geometry is not unique to any single language. It can just as easily be used with BASIC, the language of your Commodore 64. One of the reasons for turtle graphics's popularity is that it's not only a natural introduction to computing, but also an excellent tool to teach thinking.

If you're not familiar with turtle graphics, the basic concept involves moving a turtle around the screen, leaving a trail as it goes. This is done through a series of English commands, such as FORWARD and RIGHT. Other commands control the color scheme, define loops, and allow you to assemble a series of commands into procedures.

Coordinating the Turtle Programs

"Turtle Graphics Interpreter" consists of three programs designed originally for use with a disk drive; if you are using a tape drive, be sure to read the appropriate section elsewhere in this article.

Program 1, "Interpreter," does most of the work. It accepts and executes the commands you enter. Program 2, "Turtle Data," POKEs in the shape tables for the turtle sprites and a number of machine language routines. Finally, Program 3, "Turtle Boot" runs the whole package.

If you use "The Automatic Proofreader" from Appendix C, typing in these programs will be much easier. Designed to insure error-free programs, the Proofreader makes it almost impossible to enter a program incorrectly. This is important, especially with Program 2. If you mistype that program, the machine language routines which are part of it may crash the computer when the Interpreter is run.

Type in each program separately, saving them all on the same disk (or tape—refer to the section later in this article for tape use instructions). Save all three programs before you try to run any of them. *This is important: When you save Programs 1 and 2, type SAVE "TURTLE GRAPHIC 1",8 and SAVE "TURTLE GRAPHIC 2",8.* The programs must be saved out under those filenames for Program 3, Turtle Boot, to properly access them. If you want to change the filenames, then make sure lines 150 and 170 in Program 3 reflect those changes.

One final note about entering the turtle graphics programs. When you type in Program 3, leave out the CHR$(31) in line 140 until you're sure everything is working right. This will make the operation of the Boot program visible. When you are sure that the Boot is loading and running Turtle Data and the Interpreter, reinsert the CHR$(31).

Once you have all three programs saved to tape or disk, load and run the Boot program to run the whole package.

Turtle Commands

The Interpreter recognizes 30 commands, some of which can be abbreviated. In addition, the CLR/HOME key will clear the text portion of the screen and home the cursor (regardless of whether the SHIFT key is pressed). Pressing the f1 key changes the border color; f3 alters the text-background color. In addition, trying to move from the text window into the hi-res screen will be treated as a CLR/HOME. The Interpreter's commands (possible abbreviations are in parentheses) are:

FORWARD x (can be abbreviated as **FD**). Moves the turtle a distance of x in the direction it is pointing. The value of x must be greater than zero. The turtle will normally leave a trail as it moves (see PENUP, PENDOWN, PENDRAW, and PENERASE). You cannot leave the screen.

RIGHT x (RT) and **LEFT x (LT).** Turns the turtle right (clockwise) or left (counterclockwise) x degrees (x is at least zero). Because there are only eight turtle sprites, the turtle will not always seem to be pointing in *exactly* the direction it should, but it will still draw and move properly.

SETHEADING x (SETH) and **PRINTHEADING.** Setting the heading to x will turn the turtle without changing its position. Headings range from 0 to 360. Straight up is 0°, and the values increase clockwise. PRINTHEADING returns the current value of the turtle's heading.

SETPOSITION x y (SETP) and **PRINTPOSITION.** Setting the position to *x y* moves the turtle without changing its heading. The value of x should be between −159 and 160, and y values range from −106 to 106. Do not separate the x and y values with a comma, only a space. They should not be enclosed in parentheses, either. Note that the range of y will change if you change the "crunch factor" (see the section "Crunching the Screen"). The turtle starts at (0,0), the center of the screen. PRINTPOSITION returns the values of *x y*.

PENERASE (PE) and **PENDRAW (PW).** These commands control whether the turtle will erase a trail or leave one. The program starts in draw mode.

PENDOWN (PD) and **PENUP (PU).** Normally the turtle's pen is down. PENUP raises it so the turtle cannot leave or erase a trail. You may still set draw or erase modes, but you will not see any effect until after you have lowered the pen and moved forward.

PENCOLOR x (PC), BACKGROUNDCOLOR x (BC), and **TURTLECOLOR x (TC).** Each of these changes the color to x, where x is between 0 and 15. The first two will also perform a CLR/HOME. (It's not a bug, it's a feature.) There can be only one pen color on the screen at any time, so executing the PENCOLOR command will recolor all the lines that have already been drawn on the screen. Try a number of combinations of background and pen colors. Because of the hardware problems in displaying isolated pixels on the screen, the same pen color will appear as different hues at different points on the screen. Experiment—you may like the effect, which is known as *artifacting*.

SHOWTURTLE (ST) and **HIDETURTLE (HT).** Hiding the turtle is useful when you want to view a finished design. These commands have no effect on the turtle's color, movement, position, and so on. SHOWTURTLE returns the turtle to the screen.

HOME. Moves the turtle to (0,0) and sets the heading to 0°.

CLEAN. Erases the hi-res screen. Note that pressing CLR/HOME will *not* disturb the hi-res drawings.

CLEARSCREEN (CS). Performs a CLEAN *and* a HOME.

These commands, as well as all others that the turtle graphics package supports, are listed in the quick reference chart which follows the program listings.

Combining Commands

The Intrepreter will accept lines of up to 78 characters (that would fill up two entire lines in the text display window), and you may include numerous commands on each line—just be sure to use spaces between commands (no commas or colons). Here's a simple demonstration to animate the turtle:

FORWARD 100 RIGHT 90 FORWARD 100 RIGHT 90 FORWARD 100 RIGHT 90 FORWARD 100

It could have been abbreviated as:

FD 100 RT 90 FD 100 RT 90 FD 100 RT 90 FD 100

These commands cause the turtle to draw a square. Because the Interpreter is in BASIC, the turtle won't move at breakneck speed. (If you are extremely ambitious, you could convert the plotting routine to machine language.)

If you're willing to give up a little more time in interpretive overhead, you can use the powerful REPEAT (RP) command. You could rewrite the commands to draw a square as:

REPEAT 4 [FORWARD 100 RIGHT 90]

or

REPEAT 4 [FD 100 RT 90]

The statements you want to be repeated should be enclosed in square brackets and preceded by REPEAT x, where x is the number of times they should be repeated. REPEATs may be nested to a depth of 255 (although procedure calls will decrease this, as detailed below). For example, try the following commands:

CS REPEAT 8 [REPEAT 4 [FORWARD 100 RIGHT 90] RIGHT 45]

Using Procedures

The full power of turtle graphics is realized with procedures. A procedure is like a program; it's just a series of commands given a specific name. That name is added to the commands that the Interpreter will recognize.

To make up a new procedure, use the DEFINE command. For example, type DEFINE BOX. You will be prompted with BOX?, after which you should type REPEAT 4 [FORWARD 100 RIGHT 90]. The Interpreter will respond with BOX DEFINED. From now on, whenever you type BOX (either from the keyboard or from within another procedure), the commands REPEAT 4 [FORWARD 100 RIGHT 90] will be executed. You could define the last design as 8BOXES, typing CS REPEAT 8 [BOX RIGHT 45] after the 8BOXES? prompt appears.

Each time you call a procedure counts as a level of nesting (just as a repeat loop does). One very important warning: Don't allow a procedure to call itself (or to call *another* procedure that may eventually call the first). This will result in a loop that you will have to break by pressing the STOP key. When you restart the program by typing RUN, you will lose your procedure definitions and any designs on the screen.

There are a number of commands which facilitate working with procedures. NAMES will print the names of all the current procedures (limit of 255). PRINTPROCEDURE x (PPROC) will print the commands associated with the procedure named x. ERASE x will erase procedure x, and RENAME x y will change the name of procedure x to y. ERASEALL will erase *all* the current procedure definitions.

Saving and Loading Procedures

Procedures may also be saved to and loaded from disk or tape. SAVE x will save *all* the current procedures (a "workspace") to a file named *x.TURTLE*; LOAD x will copy the procedures in *x.TURTLE* into memory. These will be *added* to those already defined, so you can merge workspaces. Files may be erased from the disk with SCRATCH x, which will erase *x.TURTLE*. While these commands are operating, the screen will seem to go awry; ignore this as it will be restored when the operations are complete.

QUIT will exit the program, but leave the machine in an unusual state. The screen will still be split, but this may be corrected with RUN/STOP–RESTORE. Since memory is reconfigured, you'll want to return it to its normal state. If you don't want to power off and back on again, type:

POKE 2048,0: POKE 44,8: NEW

Crunching the Screen

Because each brand of TV and computer monitor has a different vertical aspect ratio, you may notice that your squares aren't square, circles look like eggs, and so on. If so, type:

REPEAT 180 [FORWARD 2 RIGHT 2]

If your design isn't a circle, take a centimeter ruler and measure the diameter along the x and y axes. (These should be easy to identify; just slide the ruler along the screen until you get the maximum measurements in the horizontal and vertical directions.) Divide the x value by the y value. This is the "crunch factor." Change line 50 of Program 1 to set CR to this value. If you're using a Commodore color monitor (models 1701, 1702 or 1703), the value I've supplied in the program (.74) is appropriate. Note that changing this value changes the scaling on the y axis. The new limits will be ±79/CR.

For Tape Users

You can modify the package to use a tape drive with the following changes:

- Change the device number in lines 150 and 170 of Program 3 from 8 to 1.
- Change the word DISK to TAPE in line 80.
- Delete lines 7000–7100, 25000–25060, and line 1280 in Program 1.
- Change these lines in Program 1:

```
23010 GOSUB 5000:IF WD$<>""THEN23018
23014 ER=-1:PRINT"YOU MUST SUPPLY A NAME":RETURN
23018 OPEN2,1,0,WD$+".TURTLE"
23060 CLOSE2:RETURN
24010 GOSUB 5000:IF WD$<>""THEN24018
24014 ER=-1:PRINT"YOU MUST SUPPLY A NAME":RETURN
24018 OPEN2,1,1,WD$+".TURTLE"
24040 CLOSE2:RETURN
```

Program 3 should be saved first on the tape, followed by Program 2, and then Program 1. When Program 3 is loaded and run, it will then load and run the other two programs. For this autoload feature to work properly, you must save the programs with the names shows in lines 150 and 170—TURTLE GRAPHIC 2 for Program 2 and TURTLE GRAPHIC 1 for Program 1. Or you could change the names in those lines to match the names under which you saved the programs.

There is one additional requirement for the autoload feature to operate properly. You *must* leave the PLAY button depressed after Program 3 finishes loading. If you release the button, the PRESS PLAY message will be printed to the screen when Program 2 is loaded, which will prevent the loading of Program 1.

How It Works

Short of rewriting the Interpreter in machine language, there are still a number of modifications you may wish to make to customize the program. I've included these details to briefly give you an idea of how the package functions.

Program 3 reconfigures memory to start loading programs at $4000 (16384 in decimal), leaving locations $0800–$3FFF (2048–16383) free for turtle sprite data. The LOADs and RUNs are accomplished by printing the appropriate commands on the screen and filling the keyboard buffer with RETURNs.

Program 2 POKEs in the 512 bytes of sprite data below $1000 (4096), and then puts a number of machine language routines in memory beginning at $C000 (49152). The first routine is an interrupt-driven split-screen routine. It also takes care of checking for the f1, f3, and CLR/HOME keys, and keeps text from scrolling onto the hi-res screen. This routine is initialized with SYS 49322. To clean the hi-res screen, use SYS 49295. SYS 49235 will clean under the hi-res screen (1024–1823) and erase the text screen (1824–2023). The hi-res bitmap is stored beginning at 8192.

Here are the important sections of the Interpreter (Program 1):

10–170: Initialization. Frequently used variables and constants are created first to improve speed. Here are most of the variables's functions:

PE	−1 = penup, 0 = pendown
DR	−1 = pendraw, 0 = penerase
C	conversion from degrees to radians
SC	screen base
BL	bytes per hi-res screen line
BB	bytes per hi-res screen block
MX	MSB (Most Significant Byte) of sprite 0 x location
PX	LSB (Least Significant Byte) of sprite 0 x location
PY	sprite 0 y location
BG	used for sprite x seam
CR	screen crunch factor
MA	mask
BA	base in computer
C1–C7	constants used in determining sprite position
SP	sprite image number (0–7)
H	heading
CI	degrees in circle
XH,XL	x hi/lo values
YH,YL	y hi/lo values
IX,IY	initial x,y coordinates in FORWARD command
X,Y	current coordinates
SS	sprite spacing (45°)
HA	one-half
FF	used as a mask
PC	procedure counter
DH	delta heading
K,QQ,ZZ	temporary numeric storage
T$,ZZ$	temporary string storage
SE	sprite enable
PT	sprite 0 pointer
D	distance traveled
ER	−1 = error, 0 = OK
BY	byte to be POKEd
RO,CO	row, column for upper-left corner of sprite
XS,YS	coordinates for turtle sprite
WD$	current word
NU	numeric input value
PN	procedure number temp
MD$	disk read/write mode
NP	number of procedures in disk file

200–620: The parser routine is the most complicated part of the program. NE keeps track of the nesting level. The command line typed at the keyboard is assigned to ST$(0). This serves as a permanent copy of the command line. ST(0) is an index into this string (how much has been processed). These are copied into IN$ and IN, which is what we actually work from. Commands are read off (and removed) from the left end of IN$ and executed in lines 1000–1300; IN and ST(0) are constantly updated.

Whenever a repeat command is found, the nesting level is incremented, the repetition factor is put in RP(NE), and the contents of the loop are put in a new command line, ST$(NE). The parser then executes ST$(NE) as described. When we reach the end of a command line, we "pop" up by decrementing NE and continuing where we left off in the previous command line. Advanced programmers may recognize this as a stack used to simulate recursion.

Procedures are implemented in the same way. Whenever a procedure name is encountered, we drop down a nesting level, and treat the procedure's commands as the contents of a repeat loop with a repetition factor of 1.

1000–1300: Identifies and executes commands. If you choose to permanently change the name (or abbreviation) of a command, do it here. This section also clears the error flag to 0 (false) before each command. Any command that fails will set the error flag to −1 (true). The parser keeps track of the flag, and aborts all pending commands when the flag is set true. The individual commands all have good diagnostics; and you may assume that your commands have been successfully executed if no message to the contrary is printed.

2000–8000: These subroutines are used by the Interpreter in executing various commands.

9000–22000: Each of these subroutines corresponds to a single command; consulting the variable list should help clarify them.

Sample Designs

Here are some simple designs to get you started. The names of the procedures are in boldface:

RECTANGLE: RP 2 [FD 80 RT 90 FD 30 RT 90]
HEXAGON: RP 6 [FD 100 RT 60]
PENTAGON: RP 5 [FD 100 LT 72]
PENTAGRAM: RP 5 [FD 161.8 LT 144]
TWOPENTAS: SETP −60 −80 SETH 90 PENTAGON LT 36 PENTAGRAM
ARROW: RECTANGLE LT 90 FD 15 LT 135 RP 2 [FD 42.4 LT 90] LT 45 FD 15 PE FD 28 PW

HONEYCOMB: SETP −30 30 SETH 330 RP 6 [RP 6 [FD 25 RT 60] RT 120 PU FD 25 LT 60 PD]

Program 1. Interpreter

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C, to enter the following three programs.

```
10 REM TURTLE GRAPHICS INTERPRETER            :rem 202
30 IF PEEK(49152)<>173 THEN PRINT CHR$(150) "TURTL
   E DATA DID NOT LOAD": END                   :rem 87
40 X=0: Y=0: IX=0: IY=0: D=0: NU=0: BY=0: BI=0: XH
   =160: XL=-159: C=↑/180                     :rem 121
50 CR=.74: YH=INT(79/CR): YL=-YH: BA=2: BB=8: BL=3
   20: SC=8192: PE=0: DR=-1                   :rem 195
60 MA=7: H=0: PX=53248: BB=8: BL=320: SC=8192: PE=
   0: DR=-1: MA=7: H=0: PX=53248               :rem 33
70 PY=53249: BG=256: RO=0: CO=0: XS=0: YS=0: SP=0:
    PT=2040: SE=53269: HA=.5                  :rem 189
80 C1=12: C2=40: C3=50: C4=28: C5=24: C6=3: C7=5:
   {SPACE}CI=360: MX=53264: PC=0               :rem 10
90 FF=255: SS=45: SB=56: YM=79                 :rem 88
100 DIM ST$(255),ST(255),RP(255),PR$(255),PN$(255)
                                               :rem 88
110 DEF FNR(X)=INT((X+.005)*100)/100          :rem 123
120 REM INITIALIZE SCREEN AND TURTLE          :rem 220
130 GOSUB 3000: POKE 2, 110: POKE 53277, 0: POKE 5
    3271, 0: POKE 53287,0                     :rem 146
140 SYS 49295: SYS 49235: SYS 49322: POKE SE, 1: P
    OKE 53280,2: POKE53281,11                  :rem 63
150 PRINT CHR$(129) "TURTLE GRAPHICS INTERPRETER"
                                              :rem 218
170 PRINT CHR$(30)                            :rem 218
200 REM MAIN LOOP - GET A LINE OF COMMANDS AND PRO
    CESS IT                                   :rem 193
210 ST$(0)="": INPUT ST$(0)                   :rem 118
220 NE=0: ST(0)=0: RP(0)=0: ER=0              :rem 107
230 IF ST$(0)="" THEN 210                     :rem 179
240 REM COPY UNEXECUTED PART OF CURRENT COMMAND ST
    RING (NESTING LEVEL = NE)                  :rem 37
250 REM INTO IN$ TO BE PROCESSED               :rem 66
260 IN$=RIGHT$(ST$(NE), LEN(ST$(NE))-ST(NE)): IN=0
                                               :rem 51
270 GOSUB 5000{2 SPACES}FILL WD$ WITH NEXT WORD FR
    OM IN$                                    :rem 106
280 IF WD$<>"" THEN 350                       :rem 109
290 REM IN$ IS EMPTY; WE ARE DONE WITH ALL COMMAND
    S IF NESTING LEVEL IS 0                   :rem 140
300 IF NE=0 THEN 200                          :rem 227
310 REM WE HAVE COMPLETED A REPETITION OF THE CURR
    ENT COMMAND STRING ST$(NE)                 :rem 55
```

```
320 REM IF NEEDED, REPEAT.{2 SPACES}ELSE, POP NEST
    ING LEVEL                              :rem 156
330 RP(NE)=RP(NE)-1: IF RP(NE)>0 THEN ST(NE)=0: GO
    TO 240                                 :rem 42
340 NE=NE-1: GOTO 240                      :rem 97
350 IF (WD$="REPEAT")OR(WD$="RP") THEN 440 :rem 20
360 REM CHECK IF COMMAND IS A PROCEDURE NAME
                                           :rem 16
370 GOSUB 6000: IF PN=0 THEN 410           :rem 120
380 REM STUFF IN$ WITH PROC STRING AS IF IT WERE A
     REPEAT LOOP                           :rem 56
390 IN$= "[" + PR$(PN) + "]" + RIGHT$(IN$, LEN(IN$
    )-IN): IN=0: NU=1                      :rem 28
400 ST(NE)=ST(NE)-LEN(PR$(PN))-2: GOTO 480:rem 103
410 REM IDENTIFY AND EXECUTE WD$ AS A COMMAND
                                           :rem 78
420 GOSUB 1000: IF ER THEN 200             :rem 248
430 GOTO 270: REM WE ARE DONE CURRENT COMMAND
                                           :rem 67
440 REM GET REPETITION FACTOR FOR REPEAT LOOP
                                           :rem 0
450 GOSUB 4000: IN$=RIGHT$(IN$, LEN(IN$)-IN): IN=0
                                           :rem 214
460 IF (NOT ER)AND(NU>0)AND(INT(NU)=NU) THEN 480
                                           :rem 229
470 PRINT "I CAN'T REPEAT SOMETHING " WD$ " TIMES"
     :IN$="": GOTO 200                     :rem 113
480 REM PUSH THE COMMAND STRING STACK (INCREMENT N
    ESTING LEVEL)                          :rem 115
490 NE=NE+1: IF NE=256 THEN PRINT "NESTING TOO DEE
    P": GOTO 200                           :rem 191
495 RP(NE)=NU: ST(NE)=1: K=0               :rem 45
500 REM FILL ST$(NE) WITH CONTENTS OF REPEAT BRACK
    ETS                                    :rem 158
510 ST$(NE)="": QQ=0: K=0                  :rem 1
520 T$=MID$(IN$, ST(NE), 1)                :rem 106
530 IF T$="]" THEN K=K-1                   :rem 221
540 IF K>0 THEN ST$(NE)=ST$(NE)+T$         :rem 78
550 IF T$="[" THEN K=K+1: QQ=-1            :rem 82
560 IF K<=0 THEN 600                       :rem 227
570 ST(NE)=ST(NE)+1                        :rem 75
580 IF ST(NE)<=LEN(IN$) THEN 520           :rem 225
590 PRINT "MISMATCHED BRACKETS IN REPEAT": IN$="":
     GOTO 200                              :rem 112
600 IF (K<0) OR ((K=0)AND(NOTQQ)) THEN 590:rem 172
610 ST(NE-1)=ST(NE)+ST(NE-1): ST(NE)=0     :rem 142
620 GOTO 240: REM EXECUTE THE NEW COMMAND STRING
                                           :rem 57
1000 REM IDENTIFY AND EXECUTE COMMAND      :rem 230
1005 ER=0                                  :rem 202
```

```
1010 IF (WD$="FORWARD")OR(WD$="FD") THEN GOSUB 900
     Ø: RETURN                                  :rem 69
1020 IF (WD$="RIGHT")OR(WD$="RT") THEN GOSUB 10000
     : RETURN                                   :rem 243
1030 IF (WD$="LEFT")OR(WD$="LT") THEN GOSUB 11000:
      RETURN                                    :rem 156
1040 IF (WD$="PENUP")OR(WD$="PU") THEN PE=-1: RETU
     RN                                         :rem 189
1050 IF (WD$="PENDOWN")OR(WD$="PD") THEN PE=0: RET
     URN                                        :rem 18
1060 IF WD$="HOME" THEN GOSUB 12000: RETURN
                                                :rem 123
1070 IF WD$="CLEAN" THEN SYS 49295: RETURN :rem 79
1080 IF (WD$="CLEARSCREEN")OR(WD$="CS") THEN GOSUB
      12000: SYS 49295: RETURN                  :rem 218
1090 IF (WD$="SETHEADING")OR(WD$="SETH") THEN GOSU
     B 13000: RETURN                            :rem 233
1100 IF (WD$="SETPOSITION")OR(WD$="SETP") THEN GOS
     UB 14000: RETURN                           :rem 111
1110 IF (WD$="PENERASE")OR(WD$="PE") THEN DR=0: RE
     TURN                                       :rem 73
1120 IF (WD$="PENDRAW")OR(WD$="PW") THEN DR=-1: RE
     TURN                                       :rem 72
1130 IF (WD$="ST")OR(WD$="SHOWTURTLE") THEN POKE S
     E, 1: RETURN                               :rem 76
1140 IF (WD$="HIDETURTLE")OR(WD$="HT") THEN POKE S
     E, 0: RETURN                               :rem 26
1150 IF (WD$="PENCOLOR")OR(WD$="PC") THEN GOSUB 15
     000: RETURN                                :rem 205
1160 IF (WD$="BACKGROUNDCOLOR")OR(WD$="BC") THEN G
     OSUB 16000: RETURN                         :rem 190
1170 IF (WD$="TURTLECOLOR")OR(WD$="TC") THEN GOSUB
      17000: RETURN                             :rem 210
1180 IF WD$="PRINTHEADING" THEN PRINT FNR(H): RETU
     RN                                         :rem 107
1190 IF WD$="PRINTPOSITION" THEN PRINT "(" FNR(X)
     {SPACE}"," FNR(Y) ")": RETURN              :rem 218
1200 IF WD$="DEFINE" THEN GOSUB 18000: RETURN
                                                :rem 255
1210 IF WD$="NAMES" THEN GOSUB 19000: RETURN
                                                :rem 202
1220 IF (WD$="PRINTPROCEDURE")OR(WD$="PPROC") THEN
      GOSUB 20000: RETURN                       :rem 140
1230 IF WD$="ERASE" THEN GOSUB 21000: RETURN
                                                :rem 193
1240 IF WD$="ERASEALL" THEN PC=0: PRINT "ALL PROCE
     DURES ERASED": RETURN                      :rem 188
1250 IF WD$="RENAME" THEN GOSUB 22000: RETURN
                                                :rem 12
```

```
1260 IF WD$="LOAD" THEN GOSUB 23000: RETURN
                                        :rem 118
1270 IF WD$="SAVE" THEN GOSUB 24000: RETURN
                                        :rem 135
1280 IF WD$="SCRATCH" THEN GOSUB 25000: RETURN
                                        :rem 98
1290 IF WD$="QUIT" THEN PRINT "BYE": END :rem 207
1300 ER=-1: PRINT "I DON'T UNDERSTAND " WD$: RETUR
     N                                  :rem 119
2000 REM MOVE TURTLE                    :rem 189
2010 RO=YM-(Y*CR): CO=X-XL              :rem 15
2020 IF (SP/BA)=INT(SP/BA) THEN XS=CO+C1: YS=RO+C2
     : GOTO 2200                        :rem 170
2030 XS=CO: IF SP>C6 THEN XS=XS+C5      :rem 199
2050 IF (SP=C6)OR(SP=C7) THEN YS=RO+C4: GOTO 2200
                                        :rem 222
2060 YS=RO+C3                           :rem 243
2200 IF XS<BG THEN POKE PX, XS: POKE MX, 0: GOTO 2
     220                                :rem 67
2210 POKE PX, XS-BG: POKE MX, 1         :rem 148
2220 POKE PY, YS                        :rem 118
2230 RETURN                             :rem 167
3000 REM CHANGE HEADING                 :rem 61
3010 H=H+DH                             :rem 72
3020 IF H>=CI THEN H=H-CI: GOTO 3020    :rem 144
3030 IF H<0 THEN H=H+CI: GOTO 3030      :rem 245
3040 SP=(INT(H/SS+HA)) AND MA:          :rem 160
3050 QQ=PEEK(SE): POKE SE, 0: POKE PT, SB+SP: GOSU
     B 2000                             :rem 42
3065 POKE SE, QQ                        :rem 99
3070 RETURN                             :rem 170
4000 REM NUMERIC INPUT                  :rem 75
4010 REM GETS NEXT WORD FROM IN$ AS A NUMBER (NU).
     {2 SPACES}CHECKS FOR ERROR         :rem 40
4020 GOSUB 5000: ER=0: NU=0: IF WD$="" THEN ER=-1:
      RETURN                            :rem 23
4030 FOR K= 1 TO LEN(WD$): T$=MID$(WD$, K, 1)
                                        :rem 202
4040 IF ((T$<"0")OR(T$>"9")) AND (T$<>"-")AND(T$<>
     "+")AND(T$<>".") THEN ER=-1        :rem 59
4050 NEXT: NU=VAL(WD$): RETURN          :rem 47
5000 REM FILL WD$ WITH NEXT WORD FROM IN$  :rem 53
5010 WD$="": IF IN$="" THEN 5070         :rem 6
5020 IN$=RIGHT$(IN$, LEN(IN$)-IN): IN=0  :rem 134
5030 ST(NE)=ST(NE)+1: IN=IN+1           :rem 120
5040 IF IN>LEN(IN$) THEN IN=IN-1: ST(NE)=ST(NE)-1:
      GOTO 5070                         :rem 58
5050 IF MID$(IN$, IN, 1)<>" " THEN WD$=WD$ + MID$(
     IN$, IN, 1): GOTO 5030             :rem 187
5060 IF (WD$="")AND(IN$<>"") THEN 5020  :rem 126
```

```
5070 RETURN                                       :rem 172
6000 REM IDENTIFY PROCEDURE                       :rem 175
6010 REM RETURNS INDEX (PN) OF PROCNAME IN WD$; 0
     {SPACE}IF NOT A PROCNAME                      :rem 6
6020 K=0: PN=0                                    :rem 197
6030 K=K+1: IF K>PC THEN RETURN                   :rem 236
6040 IF WD$<>PN$(K) THEN 6030                     :rem 232
6050 PN=K: RETURN                                 :rem 11
7000 REM OPEN DISK FILE                           :rem 40
7010 ER=0: GOSUB 5000: IF WD$<>"" THEN 7030
                                                  :rem 138
7020 ER=-1: PRINT "YOU MUST SUPPLY A FILENAME": RE
     TURN                                         :rem 213
7030 OPEN 15,8,15                                 :rem 88
7040 OPEN 2,8,2, "0:" + WD$ + ".TURTLE,S," + MD$:
     {SPACE}INPUT#15, QQ,T$,K,ZZ                  :rem 217
7050 IF (QQ=26)AND(MD$="W") THEN PRINT "WRITE-PROT
     ECTED DISK": ER=-1: RETURN                   :rem 183
7060 IF (QQ=67)AND(MD$="W")AND(K=36) THEN PRINT "D
     ISK IS FULL.": ER=-1: RETURN                 :rem 109
7070 IF (QQ=63)AND(MD$="W") THEN PRINT "FILENAME I
     S USED": ER=-1: RETURN                       :rem 59
7080 IF (QQ=62)AND(MD$="R") THEN PRINT "NO SUCH FI
     LE ON DISK": ER=-1: RETURN                   :rem 224
7090 IF QQ>19 THEN PRINT "I'M HAVING TROUBLE WITH
     {SPACE}THE DISK": ER=-1                      :rem 244
7100 RETURN                                       :rem 168
8000 REM GET VALID COLOR NUMBER                   :rem 68
8010 GOSUB 4000 NUMERIC INPUT                     :rem 176
8020 IF ER OR (NU>15)OR(NU<0) THEN ER=-1          :rem 139
8030 RETURN                                       :rem 171
9000 REM FORWARD COMMAND                          :rem 193
9010 GOSUB 4000: IF ER OR (NU<=0) THEN PRINT "I CA
     N'T GO FORWARD " WD$: RETURN                 :rem 198
9020 IX=X: IY=Y: FOR D= 0 TO NU: X=FNR(D*SIN(H*C)+
     IX): Y=FNR(D*COS(H*C)+IY)                    :rem 232
9030 IF X>XH THEN X=XH                            :rem 245
9040 IF X<XL THEN X=XL                            :rem 252
9050 IF Y>YH THEN Y=YH                            :rem 251
9060 IF Y<YL THEN Y=YL                            :rem 2
9070 IF PE THEN 9120                              :rem 239
9080 BY=SC + BL*INT((YM-(Y*CR))/BB) +BB*INT((X-XL)
     /BB) + ((YM-(Y*CR)) AND MA)                  :rem 74
9090 BI=MA - ((X-XL) AND MA)                      :rem 129
9100 IF DR THEN POKE BY, PEEK(BY) OR BA↑BI: GOTO 9
     120                                          :rem 113
9110 POKE BY, PEEK(BY) AND (FF-BA↑BI)             :rem 27
9120 GOSUB 2000: NEXT: RETURN                     :rem 161
10000 REM RIGHT COMMAND                           :rem 82
```

```
10010 GOSUB 4000: IF ER OR (NU<0) THEN PRINT "I CA
      N'T TURN RIGHT " WD$: RETURN             :rem 205
10020 DH=NU: GOSUB 3000: RETURN                :rem 246
11000 REM LEFT COMMAND                           :rem 0
11010 GOSUB 4000: IF ER OR (NU<0) THEN PRINT "I CA
      N'T GO LEFT " WD$: RETURN                :rem 200
11020 DH=-NU: GOSUB 3000: RETURN                :rem 36
12000 REM HOME COMMAND                         :rem 255
12010 X=0: Y=0: H=0: DH=0: GOSUB 3000: RETURN
                                               :rem 114
13000 REM SETHEADING COMMAND                   :rem 179
13010 GOSUB 4000: IF (NOT ER)AND(H<=360) THEN 1303
      0                                        :rem 127
13020 ER=-1: PRINT "I CAN'T SET A HEADING OF " WD$
      : RETURN                                  :rem 84
13030 H=NU: DH=0: GOSUB 3000: RETURN           :rem 233
14000 REM SETPOSITION COMMAND                   :rem 57
14010 GOSUB 4000: IF (NOT ER)AND(NU>=XL)AND(NU<=XH
      ) THEN 14030                             :rem 201
14020 ER=-1: PRINT "I CAN'T SET AN X-VALUE OF "WD$
      : RETURN                                 :rem 181
14030 QQ=NU: GOSUB 4000                        :rem 248
14040 IF (NOT ER)AND(NU>=YL)AND(NU<=YH) THEN X=QQ:
       Y=NU: GOSUB 2000: RETURN                :rem 152
14050 ER=-1: PRINT "I CAN'T SET A Y-VALUE OF "WD$:
       RETURN                                  :rem 107
15000 REM PENCOLOR COMMAND                      :rem 59
15010 GOSUB 8000: IF ER THEN PRINT WD$ " IS NOT A
      {SPACE}PENCOLOR": RETURN                 :rem 168
15020 POKE 2, (PEEK(2)AND15)+16*NU: SYS 49235: RET
      URN                                      :rem 112
16000 REM BACKGROUNDCOLOR COMMAND               :rem 57
16010 GOSUB 8000: IF ER THEN PRINT WD$ " IS NOT A
      {SPACE}BACKGROUNDCOLOR": RETURN          :rem 166
16020 POKE 2, (PEEK(2)AND240)+NU: SYS 49235: RETUR
      N                                         :rem 16
17000 REM TURTLECOLOR COMMAND                   :rem 58
17020 GOSUB 8000: IF ER THEN PRINT WD$ " IS NOT A
      {SPACE}TURTLECOLOR": RETURN              :rem 168
17030 POKE 53287, NU: RETURN                    :rem 28
18000 REM DEFINE NEW PROCEDURE                  :rem 27
18010 GOSUB 5000:IF WD$<>"" THEN 18030         :rem 176
18020 PRINT "I NEED A PROCEDURE NAME": ER=-1: RETU
      RN                                       :rem 194
18030 IF PC=FF THEN PRINT"I CAN'T REMEMBER ANY MOR
      E PROCEDURES": ER=-1: RETURN             :rem 105
18040 GOSUB 6000: IF PN<>0 THEN PRINT WD$ " ALREAD
      Y EXISTS": ER=-1: RETURN                 :rem 123
18050 PC=PC+1: PN$(PC)=WD$: PRINT WD$;: INPUT PR$(
      PC)                                      :rem 206
```

```
18060 PRINT WD$ " IS NOW DEFINED": RETURN  :rem 40
19000 REM PRINTNAMES COMMAND               :rem 222
19010 PRINT "NUMBER OF PROCEDURES:" PC     :rem 243
19020 IF PC=0 THEN RETURN                  :rem 154
19030 FOR K= 1 TO PC: PRINT PN$(K): NEXT: RETURN
                                           :rem 139
20000 REM PRINTPROCEDURE COMMAND            :rem 11
20010 GOSUB 5000: IF WD$<>"" THEN 20030    :rem 162
20020 ER=-1: PRINT "I NEED A PROCEDURE NAME": RETU
      RN                                   :rem 187
20030 GOSUB 6000: IF PN<>0 THEN PRINT PR$(PN): RET
      URN                                  :rem 215
20040 ER=-1: PRINT "THERE IS NO PROCEDURE " WD$: R
      ETURN                                :rem 102
21000 REM ERASE COMMAND                     :rem 70
21010 GOSUB 5000: IF WD$<>"" THEN 21030    :rem 164
21020 ER=-1: PRINT "I NEED A PROCEDURE NAME": RETU
      RN                                   :rem 188
21030 GOSUB6000: IF PN<>0 THEN 21050       :rem 116
21040 ER=-1: PRINT "THERE IS NO PROCEDURE " WD$: R
      ETURN                                :rem 103
21050 PR$(PN)=PR$(PC): PN$(PN)=PN$(PC): PC=PC-1:PR
      INT WD$ " IS ERASED": RETURN         :rem 145
22000 REM RENAME COMMAND                   :rem 143
22010 GOSUB 5000: IF WD$<>"" THEN 22030    :rem 166
22020 ER=-1: PRINT "I NEED TO KNOW THE OLD NAME":
      {SPACE}RETURN                        :rem 117
22030 GOSUB 6000                            :rem 61
22040 IF PN=0 THEN PRINT "PROCEDURE " WD$ " DOESN'
      T EXIST": ER=-1: RETURN               :rem 69
22050 QQ=PN                                :rem 118
22060 GOSUB 5000: IF WD$<>"" THEN22080     :rem 176
22070 PRINT "I NEED TO KNOW THE NEW NAME": ER=-1:
      {SPACE}RETURN                        :rem 133
22080 GOSUB 6000                            :rem 66
22090 IF PN<>0 THEN PRINT "YOU HAVE ALREADY USED T
      HAT NAME": ER=-1: RETURN               :rem 0
22100 PN$(QQ)=WD$: PRINT "RENAMING OK": RETURN
                                           :rem 182
23000 REM LOAD COMMAND                     :rem 248
23010 MD$="R": GOSUB 7000: IF ER THEN 23060
                                           :rem 137
23020 INPUT#2, NP                          :rem 166
23030 IF (NP+PC)>FF THEN PRINT "TOO MANY PROCEDURE
      S": ER=-1: GOTO 23060                :rem 251
23040 FOR K= 1 TO NP: INPUT#2, PN$(PC+K), PR$(PC+K
      ): NEXT: PC=PC+NP                    :rem 108
23050 PRINT NP "PROCEDURES LOADED"          :rem 14
23060 CLOSE 2: CLOSE 15: RETURN            :rem 211
24000 REM SAVE COMMAND                       :rem 8
```

```
24010 MD$="W": GOSUB 7000: IF ER THEN 24040
                                                :rem 142
24020 PRINT#2, PC: FOR K= 1 TO PC: PRINT#2, PN$(K)
      : PRINT#2, PR$(K): NEXT                   :rem 114
24030 PRINT PC "PROCEDURES SAVED"               :rem 204
24040 CLOSE 2: CLOSE 15: RETURN                 :rem 210
25000 REM SCRATCHFILE COMMAND                     :rem 2
25010 ER=0: GOSUB 5000: IF WD$<>"" THEN 25030
                                                :rem 234
25020 PRINT "YOU MUST SUPPLY A FILENAME": ER=-1: R
      ETURN                                       :rem 5
25030 OPEN 15,8,15                              :rem 136
25040 PRINT#15, "S0:" + WD$ + ".TURTLE": INPUT#15,
       QQ,T$,ZZ,ZZ                               :rem 42
25050 IF (QQ>19)AND(QQ<>62) THEN PRINT "I'M HAVING
       TROUBLE WITH THE DISK":ER=-1              :rem 25
25060 CLOSE 15: RETURN                          :rem 243
```

Program 2. Turtle Data

```
100 REM TURTLE DATA, FOR USE WITH TURTLE GRAPHICS
    {SPACE}INTERPRETER. USE TURTLE BOOT          :rem 95
110 REM TO LOAD.                                :rem 103
120 FOR K= 3584 TO 4095: READ J: POKE K, J: NEXT
                                                :rem 142
200 REM HEADING 0                               :rem 150
210 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,0,0,0,24,0,0                 :rem 239
220 DATA 60,0,0,126,0,0,255,0,1,255,128,3,255,192,
    7,255,224,15,255,240,31,255                 :rem 171
230 DATA 248,0,0,0,0,0,0,0                      :rem 209
240 REM HEADING 45                              :rem 211
250 DATA 0,63,255,0,31,255,0,15,255,0,7,255,0,3,25
    5,0,1,255,0,0,127,0,0,63,0                   :rem 90
260 DATA 0,31,0,0,7,0,0,3,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0                   :rem 252
270 DATA 0,0,0,0,0                               :rem 83
280 REM HEADING 90                              :rem 215
290 DATA 0,0,0,0,0,0,224,0,0,240,0,0,248,0,0,254,0
    ,0,255,0,0,255,128,0,255,192                :rem 163
300 DATA 0,255,224,0,255,224,0,255,192,0,255,128,0
    ,255,0,0,254,0,0,248,0,0,240                :rem 199
310 DATA 0,0,224,0,0,0,0,0,0,0,0,0,0,0,0         :rem 78
320 REM HEADING 135                               :rem 2
330 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,0,0,0,3,0,0                   :rem 27
340 DATA 7,0,0,31,0,0,63,0,0,127,0,1,255,0,3,255,0
    ,7,255,0,15,255,0,31,255,0                   :rem 76
```

```
350 DATA 63,255,0                                :rem 63
360 REM HEADING 180                              :rem 6
370 DATA 0,0,0,31,255,248,15,255,240,7,255,224,3,2
    55,192,1,255,128,0,255,0,0                   :rem 128
380 DATA 126,0,0,60,0,0,24,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,0,0,0,0,0,0,0                 :rem 58
390 DATA 0,0,0,0,0,0,0,0,0                       :rem 198
400 REM HEADING 235                              :rem 2
410 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,0,0,192,0,0                   :rem 203
420 DATA 224,0,0,248,0,0,252,0,0,254,0,0,255,128,0
    ,255,192,0,255,224,0,255,240                 :rem 199
430 DATA 0,255,248,0,255,252,0,0                 :rem 20
440 REM HEADING 270                              :rem 5
450 DATA 0,0,0,0,0,0,0,0,14,0,0,30,0,0,62,0,0,254,
    0,1,254,0,3,254,0,7,254,0,15                 :rem 128
460 DATA 254,0,15,254,0,7,254,0,3,254,0,1,254,0,0,
    254,0,0,62,0,0,30,0,0,14,0,0                 :rem 159
470 DATA 0,0,0,0,0,0,0,0                         :rem 105
480 REM HEADING 315                              :rem 9
490 DATA 255,252,0,255,248,0,255,240,0,255,224,0,2
    55,192,0,255,128,0,254,0,0                   :rem 134
500 DATA 252,0,0,248,0,0,224,0,0,192,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,0,0,0,0,0,0,0                 :rem 82
510 DATA 0,0,0,0,0,0,0,0,0,0,0,0                 :rem 212
600 REM SPLITSCREEN ROUTINE                      :rem 236
610 FOR K= 49152 TO 49349: READ J: POKE K, J: NEXT
                                                 :rem 254
620 DATA 173,25,208,141,25,208,41,1,208,3,76,188,2
    54,173,18,208,16,18,169,21                   :rem 166
630 DATA 141,24,208,169,27,141,17,208,169,1,141,18
    ,208,76,188,254,169,25,141                   :rem 178
640 DATA 24,208,169,59,141,17,208,169,209,141,18,2
    08,24,165,214,105,236,16                     :rem 75
650 DATA 3,32,83,192,165,197,201,4,208,3,238,32,20
    8,201,5,208,3,238,33,208,32                  :rem 200
660 DATA 132,192,76,49,234,165,2,162,0,157,0,4,232
    ,208,250,157,0,5,232,208,250                 :rem 245
670 DATA 157,0,6,232,208,250,162,31,157,0,7,202,16
    ,250,169,32,162,201,157,31,7                 :rem 238
680 DATA 202,208,250,24,160,0,162,20,32,240,255,96
    ,162,39,165,2,157,248,6,202                  :rem 199
690 DATA 16,250,96,24,169,32,133,252,169,0,133,251
    ,168,145,251,200,208,251,230                 :rem 2
700 DATA 252,165,252,201,64,208,1,96,152,240,239,1
    20,169,127,141,13,220,169,1                  :rem 198
710 DATA 141,26,208,169,192,141,21,3,169,0,141,20,
    3,169,1,141,18,208,88,96                     :rem 58
```

Program 3. Turtle Boot

```
10 REM TURTLE BOOT                                  :rem 89
20 POKE 53281, 6                                   :rem 246
30 PRINT CHR$(147); CHR$(154) TAB(10) "TURTLE GRAP
   HICS BOOT": PRINT: PRINT                        :rem 197
40 PRINT "THIS PROGRAM WILL LOAD AND RUN THE"
                                                   :rem 134
50 PRINT "TURTLE DATA AND INTERPRETER PROGRAMS.":
   {SPACE}PRINT                                    :rem 183
60 PRINT "WHILE THEY ARE LOADING, THE SCREEN WILL"
                                                   :rem 197
70 PRINT "BLANK.": PRINT                           :rem 149
80 PRINT "DO NOT REMOVE THE DISK UNTIL THE" :rem 4
90 PRINT "INTERPRETER PROMPTS YOU FOR YOUR FIRST"
                                                   :rem 126
100 PRINT "COMMAND.": PRINT: PRINT: POKE 198, 0
                                                   :rem 132
110 PRINT "PRESS " CHR$(18) "SPACE" CHR$(146) " WH
    EN READY"                                       :rem 51
120 GETA$: IF A$="" THEN 120                        :rem 73
130 Q$=CHR$(34): D$=CHR$(17)                       :rem 152
140 PRINT CHR$(147); CHR$(31); D$; D$; D$ "POKE 16
    384, 0: POKE 44, 64: NEW"                       :rem 74
150 PRINT D$; D$ "LOAD" Q$ "TURTLE GRAPHIC 2" Q$ "
    ,8"                                            :rem 120
160 PRINT D$; D$; D$; D$; D$ "RUN"                  :rem 81
170 PRINT D$; D$ "LOAD" Q$ "TURTLE GRAPHIC 1" Q$ "
    ,8"                                            :rem 121
180 PRINT D$; D$; D$; D$; D$ "RUN" CHR$(19):rem 15
190 FOR K= 1 TO 7: POKE 630+K, 13: NEXT: POKE 198,
    7                                                :rem 3
```

Turtle Graphics Commands Quick Reference Chart

Command	Description
FORWARD x (FD)	Moves turtle forward
RIGHT x (RT)	Turns turtle clockwise
LEFT x (LT)	Turns turtle counterclockwise
SETHEADING x (SETH)	Turns turtle without changing position
PRINTHEADING	Returns current turtle heading
SETPOSITION x y (SETP)	Moves turtle without changing heading
PRINTPOSITION	Returns current turtle coordinates
PENERASE (PE)	Erase a trail
PENDRAW (PW)	Draw a trail
PENDOWN (PD)	Pen is down

PENUP (PU)	Pen is up—turtle cannot erase or draw
PENCOLOR x (PC)	Changes trail color
BACKGROUNDCOLOR x (BC)	Changes hi-res background color
TURTLECOLOR x (TC)	Changes turtle color
SHOWTURTLE (ST)	Shows the turtle again after it's been hidden
HIDETURTLE (HT)	Makes the turtle invisible
HOME	Moves turtle to 0,0 and sets heading to 0 degrees
CLEAN	Erases the hi-res screen
CLEARSCREEN	Performs a CLEAN and a HOME
REPEAT x [ ] (RP)	Repeats a command
DEFINE x	Define a procedure
NAMES	Prints all the names of current procedures
PRINTPROCEDUREx (PPROC)	Prints the commands in procedure x
ERASE x	Erases procedure x
RENAME x y	Renames procedure x with new name y
ERASEALL	Erases all current procedures
SAVE x	Saves all current procedures as filename *x.TURTLE*
LOAD x	Loads into memory procedures from filename *x.TURTLE*
SCRATCH x	Erases filename *x.TURTLE* from disk
QUIT	Exit the program

These are not commands, but keys you can press for the following results:

CLR/HOME	Clears the text display window
f1	Changes screen border color
f3	Changes text background color

4
Sound and Graphics

Programming 64 Sound

John Michael Lane

This in-depth look at sound for the 64 provides you with practical methods for controlling the 64's SID chip from BASIC. Not only does it discuss sound and music in general, but it also examines some techniques for programming more complicated music.

Sight and sound are two essential components of successful computer games. Though the methods used to produce visual images differ from one computer to another, it's not too hard to produce an image that looks something like what you want. When designing space games, it's really easy, because just about anything can look like a spaceship.

Producing sound, however, can be quite a different matter. How can you produce the sound of a laser gun when dealing with such unfamiliar concepts as frequency, waveforms, and envelopes? (Actually lasers don't make any noise, but you know the sound I mean.)

Without a pretty expensive test setup, it can seem impossible to produce exactly the sound you're looking for. The only recourse is trial and error. Still, if you understand a little about the physics of sound and how it relates to the sound generator you're using, you can produce creditable results.

Real Sound

Sound is produced when physical objects vibrate. Vibrations are then set in motion and travel through the air as sound waves to our ears. Sound, in its purest form, has only two physical attributes, *frequency* and *amplitude*. Frequency, the number of vibrations per sound, is usually measured in cycles per sound, or *hertz*. The higher the frequency or *pitch* of the sound, the higher a note sounds to our ears.

We've probably never heard a tone that consisted purely of one frequency. Physical objects create vibrations at frequencies which are multiples of a fundamental frequency. The presence and quantity of these overtones determine the tonal quality, the *color* or *timbre,* of the sound. It's this tonal quality that determines whether a noise we hear sounds like a banjo or a drum (although there are other factors which we'll get to in a minute).

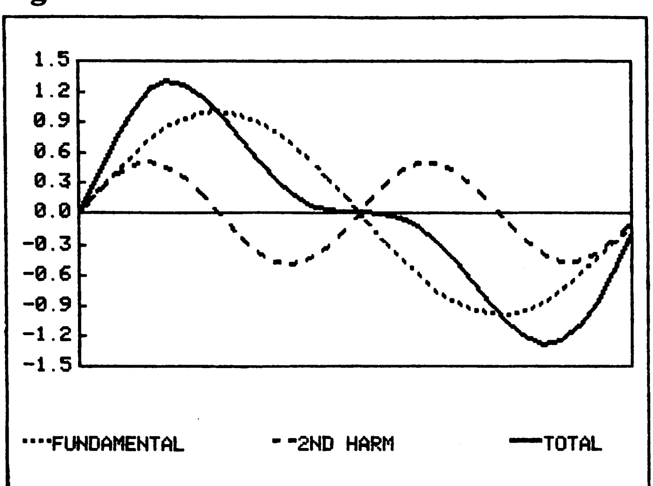
Different instruments and objects produce these overtones in varying amounts. Some produce overtones strong in even multiples of the fundamental frequency. Others produce tones rich in the odd multiples. There really is no limit to the variety of tonal qualities that exist in the real world.

On some organs, and on some music synthesizers, you can specify the exact amount of each overtone you want included in each sound. On the synthesizer included in the Commodore 64, this is handled through the different types of waveforms that can be selected. But how does a waveform relate to tonal quality?

Waveforms

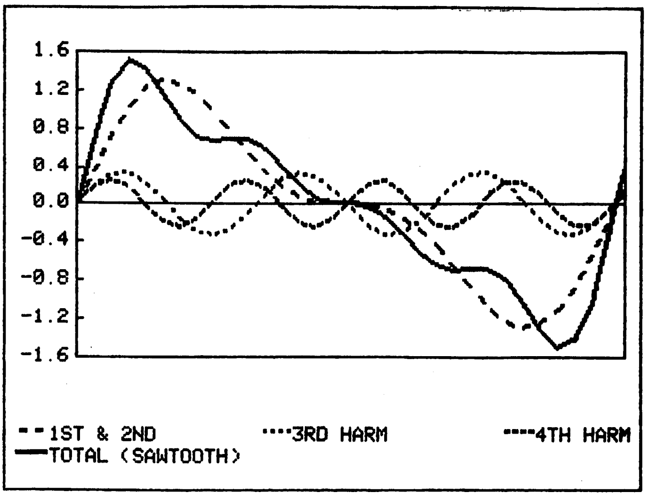
Figure 1 shows a sine wave at the fundamental frequency (all pure tones are sine waves) and at the first overtone or second harmonic. Notice that when we add the two waveforms together, the result no longer exactly resembles a sine wave.

Figure 1. Fundamental and Sound Harmonics Combined



In Figure 2, we've continued adding sine waves of higher harmonics. You can see that the resulting total waveshape is beginning to resemble a sawtooth, one of the waveforms available from the Commodore 64's Sound Interface Device (SID). If we kept adding the higher harmonics until we reached infinity, we would have a perfect sawtooth.

Figure 2. Adding Third and Fourth Harmonics Creates Sawtooth

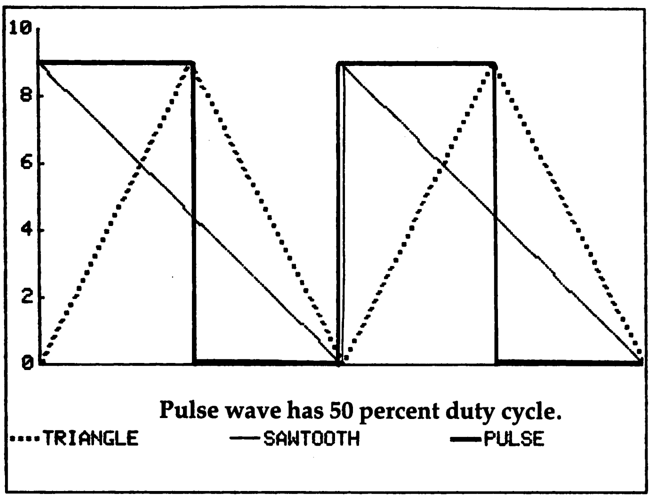


So the shape of the wave actually defines the harmonic content of the sound. Since all pure tones are sine waves, the shape of the wave generated by a sound synthesizer is actually assembled from sine waves that are multiples of the fundamental frequency.

The Commodore 64's SID has a choice of three basic waveforms and *white noise,* which is a collection of random frequencies. The three waveforms are a triangular wave, a pulse wave, and a sawtooth wave. The pulse wave has a variable pulse width, or *duty cycle,* which allows you additional freedom to vary the color of the sound produced. None of these waveshapes corresponds exactly to the sound produced by any instrument. It is also impossible to duplicate the complex harmonics of a real instrument simply by choosing one of these three waveforms. They do, nevertheless, give you the flexibility to produce a wide variety of color content, and you can get close to the particular sound you're seeking.

The harmonic content of the triangular wave diminishes very quickly, and the color of the wave consists almost entirely of the fundamental frequency. The sawtooth wave is the richest in terms of harmonics, and the pulse wave falls in between. However, since the pulse width of the pulse wave can be varied, it can also contain a great variety of harmonic content. Figure 3 illustrates the three different waveforms available through your 64's SID chip.

Figure 3. Waveform Shapes



Sound Envelopes

Earlier we said that sound consists of two qualities, frequency and amplitude. We've discussed primary frequency and how harmonic overtones are defined by the shape of the wave, but what about amplitude, or loudness?

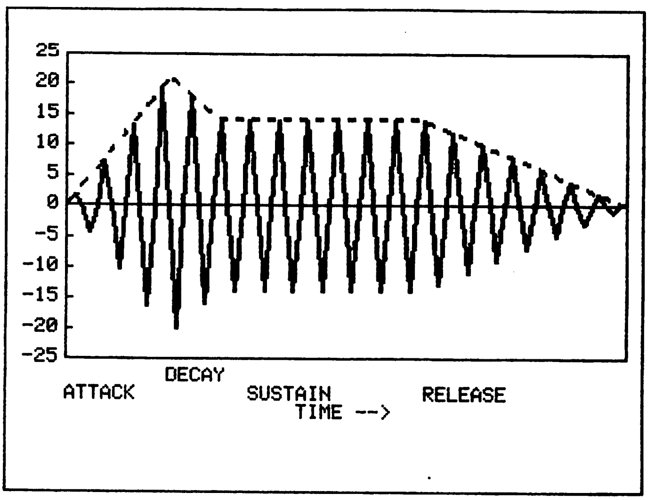
We don't mean how loud the sound is simply in the sense of volume, but rather how quickly the sound rises to its full strength and how quickly it dies down again to silence.

If you play an organ, you know that the sound of a note almost immediately reaches its full strength after you press the key and just as quickly dies down when you release the key. To our ears, it's just about instantaneous.

This is quite different from plucking a guitar string, where the sound quickly (but not quite instantaneously) reaches its full height and then slowly dies down, so that the tone continues several seconds after the note was struck. Violins, xylophones, banjos, and woodwinds are all different in the way that the sound rises, is sustained, and then dies down. Generally, these qualities are referred to as the *envelope* of the sound.

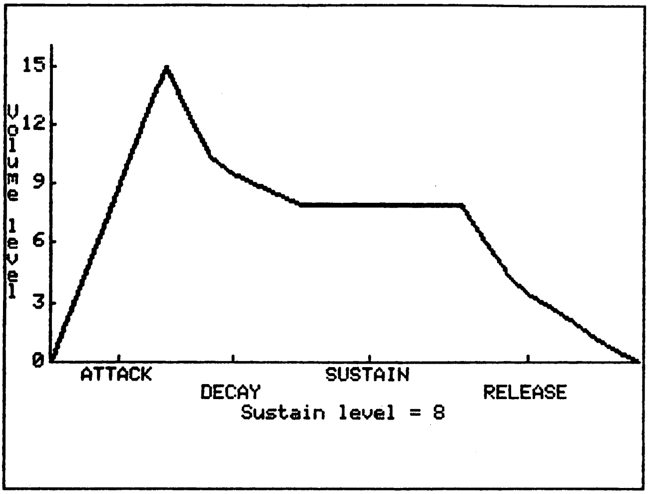
If you look at Figure 4, you'll see how a sound looks if you could feed it into an oscilloscope. We can see the shape of the wave. The shape of the envelope defines the characteristics of a sound in a manner very similar to the way that harmonic content defines a sound.

Figure 4. The Envelope Defines the Height of Individual Waveforms



The Commodore 64 uses a four-part sound envelope (see Figure 5). The first phase, called the *attack*, is the length of time it takes for the sound to reach its full volume. The second phase is the *decay*. During this phase, the sound decreases from the peak achieved during the attack phase to the level set for the sustain phase. During the third or *sustain* phase, the volume remains constant. In the final phase, the *release*, the volume decreases to zero.

Figure 5. Attack/Decay/Sustain/Release (ADSR) Envelope



Not all sounds have this four-part volume envelope. Some have only an attack and release phase, and some (like the organ) have only the sustain phase. We can achieve all these on the Commodore 64 simply by setting the other phases to zero.

The Commodore's SID allows us to set the attack, decay, and release phases to any one of 15 values or to zero. The times that correspond to the 15 values can be seen in Table 1. The times vary from milliseconds to seconds. Note that the table does not include times for the sustain phase. The SID chip allows you to set a sustain volume level, but you must control the length of the sustain by opening and closing a *gate*. That gate is bit 0 of the fourth register in the SID chip. We'll cover this in greater detail later.

To turn the sound *on* in the SID chip, you must open the *gate*. As soon as the gate is opened, the sound level begins to rise at a rate determined by the attack. Once the peak level is reached, the sound begins to decline to the level set for the sustain. The rate at which it declines is defined by the decay.

However, if the sustain level is set at 15 (the highest choice), the decay phase is essentially meaningless because the sustain level and the peak of the attack phase are the same. Thus the decay phase has nowhere to decay to.

Table 1. ADSR Envelope Values and Times

VALUE	ATTACK RATE	DECAY RATE	RELEASE RATE
0	2 ms	6 ms	6 ms
1	8 ms	24 ms	24 ms
2	16 ms	48 ms	48 ms
3	24 ms	72 ms	72 ms
4	38 ms	114 ms	114 ms
5	56 ms	168 ms	168 ms
6	68 ms	204 ms	204 ms
7	80 ms	240 ms	240 ms
8	100 ms	.3 sec	.3 sec
9	.25 sec	.75 sec	.75 sec
10	.5 sec	1.5 sec	1.5 sec
11	.8 sec	2.4 sec	2.4 sec
12	1 sec	3 sec	3 sec
13	3 sec	9 sec	9 sec
14	5 sec	15 sec	15 sec
15	8 sec	24 sec	24 sec

Once the decay phase is complete, the sustain cycle will continue as long as the gate is open. When the gate is closed, the release phase begins and the volume falls from the sustain level to zero. So, how long is the sustain phase?

Obviously, the sustain phase lasts as long as the time that the gate is open, minus the time required for the attack and decay phases. If you close the gate too soon, you may have no sustain phase at all. If you close it really early, you'll cut short the decay or the attack and decay phases as well. Figure 6 shows several combinations of attack, decay, and release values and how they interact with the gate to produce the sound envelope.

Figure 6. Standard Four-Part Envelope

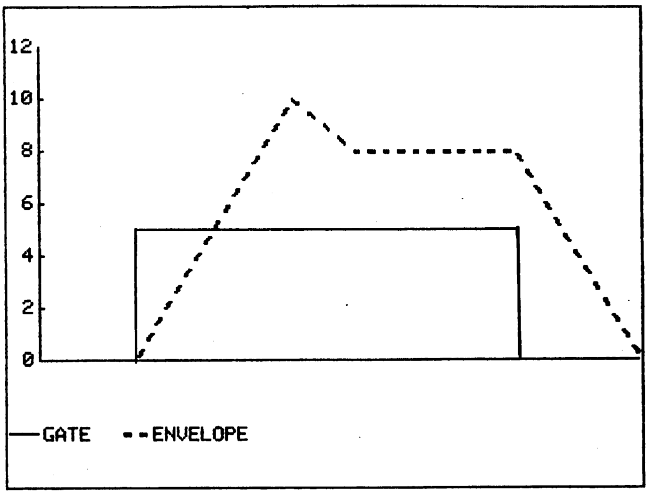


Figure 6a. Organlike Envelope

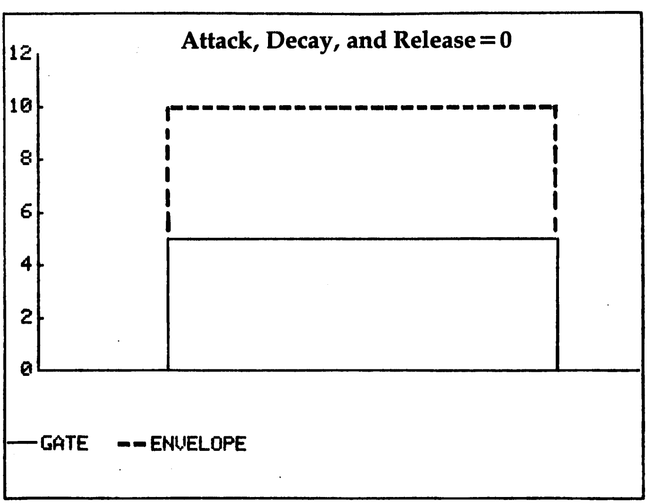


Figure 6b. Pianolike Envelope

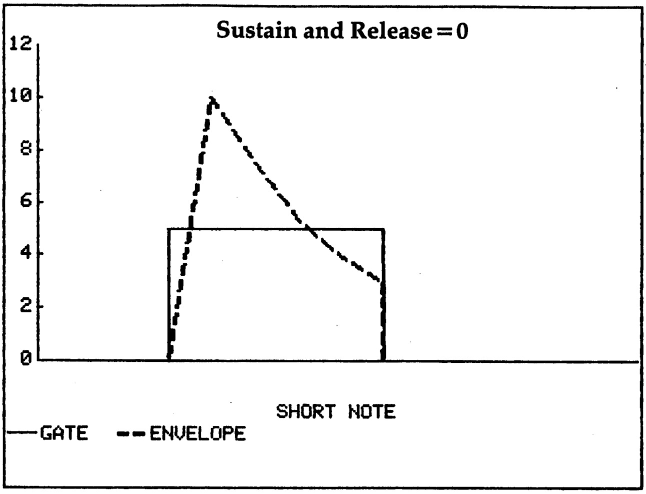
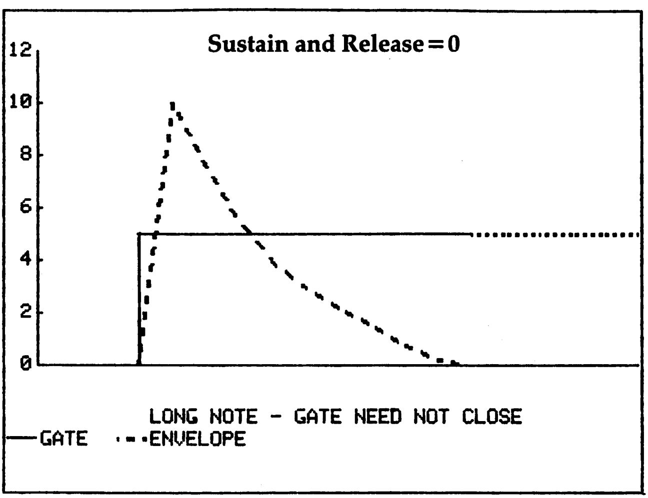


Figure 6c. Pianolike Envelope



Programming Sound

The SID is really a quite amazing chip. It uses just 29 programmable registers, and with those (you won't even use them all) you can produce a great variety of sounds.

For our purposes, we'll consider only the first 21 registers in the SID chip. We'll also briefly consider the twenty-fifth register, which sets the volume (no volume, no sound).

The first 21 registers break down into three groups of seven. That's because the SID has three voices, and the seven-register groups perform almost the same function for all three voices. That makes it far easier—all you have to learn is how to program seven registers.

Table 2 gives the functions of the seven register groups. Registers 0 and 1 hold the frequency. Register 0 contains the least significant byte, and register 1 the most significant byte. With two registers you can store only numbers less than 65512. That sounds pretty high, but the frequency contained in the two registers relates to the internal oscillator (clock) of the Commodore 64 and does not translate to the frequency we are familiar with in terms of cycles per second (hertz). To translate into hertz, you must multiply the frequency contained in the two registers by .059605. This means that the highest frequency the SID can produce is 3904 hertz. The frequency can go as low as zero, but the sound system in your TV set probably won't reproduce a frequency of less than 50 hertz (or 840 to the SID).

The easy way to load the frequency into the two registers is to use this program segment:

```
100 S=54272:REM (STARTING ADDRESS OF SID CHIP)
110 F0=FR/.059605:REM FR=FREQUENCY IN CYCLES/SECON
    D
120 F2=INT(F0/256):F1=F0-256*F2
130 POKE S,F1:POKE S+1,F2
```

If you already know the frequency in terms of the SID chip, you can omit line 110.

The next two registers (2 and 3) contain the pulse width of the rectangular pulse wave. This value is a 12-bit number with the eight least significant bits stored in register 2, and the four most significant stored in bits 3–0 of register 3. The four remaining bits of register 3 are not used. If you are using something other than a pulse wave, you don't have to worry about doing anything with these two registers.

The pulse width can take a value from 0 to 4095, which corresponds to a range of 0–100 percent for the duty cycle. A value of 2048 implies a 50 percent duty cycle and generates a square wave. If these two registers are set to zero and the pulse wave is selected, no sound will be produced.

The following program segment can be used to set the pulse width:

```
140 P0=DC*4095/100:REM DC=DUTY CYCLE IN %
150 P2=INT(P0/256):P1=P0-256*P2
160 POKE S+2,P1:POKE S+3,P2
```

A duty cycle of 10 percent will sound exactly the same as a duty cycle of 90 percent. For some advanced applications, the two may sound different, but for a solitary pulse wave, there will be no difference.

Table 2. Map of Sound Interface Device (SID) Registers

```
ADDRESS  REG #   BIT 7   BIT 6   BIT 5   BIT 4   BIT 3   BIT 2   BIT 1   BIT 0
================================================================================
                         VOICE ONE
FREQUENCY REGISTERS
54272     0      [<------        FREQUENCY    LOW ORDER BYTE            ------>]
54273     1      [<------      FREQUENCY    HIGH ORDER BYTE             ------>]
PULSE WIDTH REGISTERS
54274     2      [<------      PULSE WIDTH    LOW ORDER BYTE            ------>]
54275     3      [<------BITS 7-4 NOT USED---->][ HIGHEST 4 BITS OF PULSE WIDTH]
CONTROL REGISTER
54276     4      [NOISE][PULSE ][SAWTH.][TRIANG][ TEST ][ RING ][ SYNC ][ GATE ]
ATTACK/DECAY REGISTER
54277     5      [<----   ATTACK VALUE    ---->][<---     DECAY VALUE       --->]
SUSTAIN/RELEASE REGISTER
54278     6      [<----   SUSTAIN LEVEL   ---->][<---    RELEASE VALUE      --->]

                         VOICE TWO
FREQUENCY REGISTERS
54279     7      [<------        FREQUENCY    LOW ORDER BYTE            ------>]
54280     8      [<------      FREQUENCY    HIGH ORDER BYTE             ------>]
PULSE WIDTH REGISTERS
54281     9      [<------      PULSE WIDTH    LOW ORDER BYTE            ------>]
54282    10      [<------BITS 7-4 NOT USED---->][ HIGHEST 4 BITS OF PULSE WIDTH]
CONTROL REGISTER
54283    11      [NOISE][PULSE ][SAWTH.][TRIANG][ TEST ][ RING ][ SYNC ][ GATE ]
ATTACK/DECAY REGISTER
54284    12      [<----   ATTACK VALUE    ---->][<---     DECAY VALUE       --->]
SUSTAIN/RELEASE REGISTER
54285    13      [<----   SUSTAIN LEVEL   ---->][<---    RELEASE VALUE      --->]

                         VOICE THREE
FREQUENCY REGISTERS
54286    14      [<------        FREQUENCY    LOW ORDER BYTE            ------>]
54287    15      [<------      FREQUENCY    HIGH ORDER BYTE             ------>]
PULSE WIDTH REGISTERS
54288    16      [<------      PULSE WIDTH    LOW ORDER BYTE            ------>]
54289    17      [<------BITS 7-4 NOT USED---->][ HIGHEST 4 BITS OF PULSE WIDTH]
CONTROL REGISTER
54290    18      [NOISE][PULSE ][SAWTH.][TRIANG][ TEST ][ RING ][ SYNC ][ GATE ]
ATTACK/DECAY REGISTER
54291    19      [<----   ATTACK VALUE    ---->][<---     DECAY VALUE       --->]
SUSTAIN/RELEASE REGISTER
54292    20      [<----   SUSTAIN LEVEL   ---->][<---    RELEASE VALUE      --->]

VOLUME REGISTER
54296    24      [-NOT COVERED IN THIS ARTICLE-][<----  VOLUME CONTROL     ---->]
```

Now that we've covered the general aspects of sound and music programming on the 64, let's look at some more complicated techniques.

The Control Register

The control register (register 4 in Table 2) is the most complex register in the chip. *Each* of the eight bits in this register has a has a different function. Dealing with individual bits within a one-byte register is often a problem for BASIC programmers. One very easy way to approach the problem is to use the following:

```
170 B(0)=1
180 B(1)=0
190 B(2)=1
200 B(3)=0
210 B(4)=0
220 B(5)=0
230 B(6)=0
240 B(7)=1
250 FOR I=0 TO 7
260 Q=Q+B(I)*2↑I
270 NEXT I:POKE S+4,Q
```

This is not a very efficient way of programming, but by defining the bits we want (that is, B(I) where I = the bit number) in terms of a 1 and those we don't want in terms of a 0, this will work. It will be somewhat slow and cannot be used in a loop that must execute quickly, which is usually the case when doing musical programming.

A quicker method is to think of the bits in terms of their values in an eight-bit binary number. Bit 0 has a value of 1, bit 1 is 2, bit 2 is 4, bit 3 is 8, and so on, until bit 7 equals 128. In the lines above, we set bits 0, 2, and 7 *on*; to use the more efficient technique of bit values, we can simply add their values: 1+4+128=133. POKEing 133 into the register then sets those three bits. It's much simpler, but requires you to add up the bit values *before* writing the program.

The first bit of the control register, bit 0, acts as the gate to turn the sound on and off. Remember that when the gate is opened (when bit 0 is set to 1), the attack phase of the volume envelope begins. When the gate is closed (bit 0 is set to 0), the release phase of the envelope is triggered. If the gate is closed prematurely, the sustain, decay, and even a portion of the attack phase may be omitted. Opening and closing the gate is actually very easy. Just remember that POKEing an odd value in register 4 turns the gate *on* and that POKEing an even value turns the gate *off*.

Watch the Timing

Be careful of turning the gate off by POKEing zero into the register. That will also clear the waveform bits (which we'll discuss in a second) and will result in your envelope having no release phase.

The next bit, bit 1, is the *sync* bit. If this bit is on, the output from voice 1 will be synchronized with the output from voice 3. *Sync* in this case means that the output of voice 1 will be replaced with a logical AND of the output of voice 1 and voice 3. Another way to think of it is that voice 1 is turned on and off with the frequency of voice 3. In order for this bit to have any effect, voice 3 must be set to a frequency less than voice 1. The best way to understand this effect is to listen to it. Program 4, "Laser," contains a demonstration using the sync bit. When using sync, the lower frequency will predominate. The effect works best when the lower frequency is 10–50 percent of the higher.

The sync bit has a slightly different effect in the other two voices. In voice 2 it produces a sync of voice 2 with voice 1, and in voice 3 it produces a sync of voice 3 with voice 2.

The next bit, bit 2, is the *ring modulation* bit. When this bit is set on, it produces nonharmonic overtones that sound like a bell. In order for this effect to take place, the triangular waveform must be selected for voice 1, and voice 3 must have a frequency other than zero.

Ring modulation in the other voices works like the sync bit; that is, for voice 2 to be ring modulated, voice 1 must have a nonzero frequency. For voice 3, voice 2 must be nonzero. In all cases the triangular waveform must be selected for the affected voice.

Bit 3 in the control register is the test bit. Setting the test bit to one will turn off the sound generator. This technique will generally be used only by machine language programmers.

Bits 4–7 are the waveform bits. Turning on bit 4 will select the triangular waveform; bit 5 will select the sawtooth; bit 6 the pulse; and bit 7 white noise (like the hissing sound you hear between stations on a radio).

At this point you must be asking yourself "What happens if more than one bit is selected?" The answer is that the two (or more) waveforms will be ANDed together (a logical AND will be done on the waveforms). Commodore cautions that selecting more than one waveform while using the white noise waveform could cause the oscillator to go silent, so don't combine waveforms using the white noise waveform. Even while avoiding the white noise waveform, it's still possible to generate four more waveform shapes using combinations of the sawtooth, triangular, and rectangular pulse waveforms. However, the volume declines significantly when combining waveforms.

Register 5 contains the attack and decay values for voice 1's sound envelope. (Registers 12 and 19 serve the same function for voices 2 and 3 respectively.) The four-bit attack value is held in bits 7–4. The four-bit decay value is held in bits 3–0. The values can be loaded like this:

```
300 A=13:D=5:REM ATTACK=13,DECAY=5
310 POKE S+5,16*A+D
```

Register 6 contains the sustain level and the release value for voice 1. (Again, registers 13 and 20 are used for voices 2 and 3.) As above, the sustain level is held in bits 7–4, and the release value in bits 3–0. Program them in the following manner:

```
320 SU=13:R=4:REM SUSTAIN=13,RELEASE=4
330 POKE S+6,16*SU+R
```

We've covered the seven register groups and shown how to load them. Program 1, "Twiddle," allows you to explore all possible combinations using these seven registers. The program lets you set and change any of the values and then listen to an eight-note scale governed by those values. If you sit down and play with the program for a couple of hours, you'll get a good understanding of how changing the SID chip parameters affects a sound. The program is also useful for demonstrating how to play a tune within a BASIC program. (Note that pressing almost any key not displayed on the screen will play the sound scale you've set up.)

From Sound to Music

To play actual music, you generally write a program which will load all the parameters *except* the waveform and the frequency. At this point you select the note to be played and POKE the appropriate values into the frequency register. Then you POKE the waveform value plus one (16+1=17 for triangular, 33 for sawtooth, 65 for pulse, and 129 for white noise) into register 4 (the control register). Adding a 1 causes the gate bit (bit 0) to be turned on, and the tone begins. The program waits a certain period of time and then POKEs the waveform value (16, 32, 64, or 128) into the register. By POKEing an even number into the register we turn the gate off, and the note begins its release phase and gradually dies out (according to the release value that you've set).

A simple way to time the note is to use a delay loop. An empty loop (like the one below) will execute 1000 cycles in just about one second.

```
400 FOR I=1 TO 1000:NEXT I
```

Therefore, each cycle is just about 1/1000 second (or a millisecond). To turn the note on and off, the program line will look like this:

```
400 POKE S+4,17:FOR I=1 TO 250:NEXT:POKE S+4,16
```

The above program line will play a note for about one quarter of a second.

This technique works well for a single voice, but it may not work at all for more than one voice. The problem is that while the computer is timing the duration of one note, it cannot be separately timing voices 2 and 3. We could fill the empty loop with timing routines for voices 2 and 3, but that would change the execution time for the loop and throw the timing off.

A second technique is to use the internal timer of the Commodore 64 through the use of variable TI. The variable TI is updated automatically on the Commodore 64 and increases by a value of one every 1/60 second. We can use this timer to time the duration of our notes:

```
500 T0=TI:REM INTIALIZE THE VARIABLE "T0"
510 T0=T0+D:REM INCREASE "T0" BY DURATION OF THE F
    IRST NOTE - D
520 IF T0<=TI THEN GOSUB 1100:REM CHECK IF THE TIM
    E IS UP
525 REM IF SO SUBROUTINE 1100 WILL CHANGE NOTES
530 GOTO 520:REM IF NOT CHECK TIME AGAIN
```

The key to using this routine is to make sure that the subroutine executes quickly, at least while using multiple voices. Program 2, "Tune," illustrates this technique using all three voices. But this method isn't problem-free. We want to reproduce the rhythm of the original tune as accurately as possible. It's physically impossible to change the frequency of all three voices at once. Using BASIC, it's somewhat difficult to change all three voices in less than 1/6 second. For that reason, we split all the frequencies into the higher-and lower-order bytes before the tune begins. We can then change the frequency of all three voices in about 1/10 second. For most tunes that will be satisfactory. However, for a fast tempo, you might have to omit the second or third voice in order to maintain the rapid changes of the first voice.

Sound Effects

Let's briefly explore sound effects: the noise of a firing laser, or an explosion, siren, or any other sound we need. How can we do it?

Unfortunately, there's no direct way. The best approach is trial and error. Listen to the sound carefully. Most sounds in nature cannot be duplicated simply by selecting the right waveform and envelope. Generally, the frequency is also actively changing during the sound's life. While you listen to (or think about) the sound you want, consider what's happening to the frequency. Is it rising or falling? How quickly?

Also consider the volume. Many volume envelopes cannot be duplicated using the attack/decay/sustain/release envelope. You'll often have to change the volume level through program control, using the volume register (register 24).

Programs 3 and 4, "Blast-off" and "Laser," illustrate one approach. In Blast-off, both the frequency and volume are modulated by the program. Laser demonstrates the sync feature and modulates the frequency to produce the laser sound. Both programs were written after much trial and error.

Many authors, when converting programs to the 64, simply drop the sound effects or stop at a sound which is only vaguely like the one they want. Be persistent; the 64 can accurately produce almost any sound. As you gain experience, you'll find that the trial and error phase will decrease significantly.

Twiddle (Program 1) illustrates the basic methods of loading the SID registers and lets you experiment by changing the waveform and ADSR envelope while listening to the musical scale.

Tune (Program 2) uses the three voices to play an English folk tune. Don't be discouraged by the long list of DATA statements. Voice 1 repeats the same statements four times, and there is considerable repetition in voices 2 and 3. Once you've typed in the first few DATA statements, you can simply change the line numbers with the screen editor to produce the remainder.

Tune can be used to produce any melody by changing the values in the DATA statements. Each note is represented by a pair of values. The first represents the duration of the note (in sixtieths of a second). A value of 30–40 is appropriate for a quarter note. The second value is the frequency of the note. Appendix E in the *Commodore 64 Programmer's Reference Guide* offers a simple frequency table. Below are the values for the 12-semitone scale starting at middle C.

C—4291	C#—4547
D—4817	D#—5103
E—5407	
F—5728	F#—6069
G—6430	G#—6812
A—7217	A#—7647
B—8101	

Notes for other octaves can be calculated by doubling or halving these values, depending upon whether you're going one octave up (doubling) or one octave down (halving).

It's useful to convert one measure of music to one DATA statement if you can. This makes it easier to match the voices.

Voice 1 is the sound of a flute, voice 2 is a mandolin, and voice 3 is a guitar.

Blast-off and Laser are supposed to produce the sound of their titles. They're pretty straightforward.

Program 1. Twiddle

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C, when you type in the following four programs.

```
5 S=54272                                        :rem 201
7 DIM A(15),D(15)                                 :rem 48
10 FORL=STOS+24:POKEL,0:NEXT                      :rem 53
15 GOSUB 1000                                    :rem 167
17 GOSUB 1100                                    :rem 170
18 GOSUB 1200                                    :rem 172
20 PRINT"{CLR}";TAB(5);"TOUCH W FOR WAVEFORM"
                                                   :rem 5
30 PRINT TAB(5)"TOUCH A FOR ATTACK RATE"          :rem 32
40 PRINT TAB(5)"TOUCH S FOR SUSTAIN LEVEL":rem 238
45 PRINT TAB(5)"TOUCH T FOR SUSTAIN TIME" :rem 171
50 PRINT TAB(5)"TOUCH R FOR RELEASE"              :rem 80
60 PRINT TAB(5)"TOUCH D FOR DECAY"               :rem 168
70 PRINT TAB(5)"TOUCH P FOR PULSE WIDTH"          :rem 88
72 PRINT TAB(5)"TOUCH B TO SET DEAD TIME"         :rem 40
75 PRINT TAB(5)"TOUCH + OR - FOR FREQUENCY CHANGE"
                                                  :rem 85
```

```
80 GET A$:IF A$=""THEN80                          :rem 243
82 IF A$="W"THEN 200                              :rem 247
84 IF A$="A" THEN 250                             :rem 232
86 IF A$="S" THEN 300                             :rem 248
88 IF A$="R" THEN 350                             :rem 254
90 IF A$="D" THEN 400                             :rem 229
92 IF A$="P" THEN 450                             :rem 248
94 IF A$="T" THEN 500                             :rem 250
96 IF A$="+" THEN GOSUB 1400                      :rem 131
97 IF A$="B" THEN 550                             :rem 240
98 IF A$="-" THEN GOSUB 1450                      :rem 140
100 REM                                           :rem 117
105 POKE S+24,15                                  :rem 59
110 POKE S+5,16*A+D                               :rem 225
120 POKE S+6,16*SL+R                              :rem 79
130 POKE S+3,INT(P/256)                           :rem 248
140 POKE S+2,P-256*INT(P/256)                     :rem 60
150 FOR I=1 TO 8                                  :rem 15
160 IFINT(F(I))<=65536THENPOKE S+1,INT(F(I)/256)
                                                  :rem 229
170 POKE S,F(I)-256*INT(F(I)/256)                 :rem 2
180 IFINT(F(I))<=65536THENPOKE S+4,2↑(W+3)+1
                                                  :rem 244
185 FORJ=1TOT:NEXT                                :rem 173
187 POKE S+4,2↑(W+3)                              :rem 67
188 FORJ=1TOB:NEXT                                :rem 158
190 NEXT I:GOTO 20                                :rem 247
200 PRINT"WAVEFORM IS";" - ";W                    :rem 164
202 PRINT"1=TRIANGLE"                             :rem 41
204 PRINT"2=SAWTOOTH"                             :rem 79
206 PRINT"3=PULSE"                                :rem 98
208 PRINT"4=NOISE"                                :rem 90
210 INPUT"ENTER WAVEFORM (1-4)";W                 :rem 193
215 IFW<1 ORW>4THEN210                            :rem 23
220 GOTO 100                                      :rem 94
250 PRINT"ATTACK RATE IS";A                       :rem 100
260 INPUT"ENTER ATTACK RATE (0-15)";A             :rem 94
265 IFA<0ORA>15THEN260                            :rem 38
270 GOTO 100                                      :rem 99
300 PRINT"SUSTAIN LEVEL IS";SL                    :rem 121
310 INPUT"ENTER SUSTAIN LEVEL (0-15)";SL          :rem 115
315 IFSL<0ORSL>15THEN310                          :rem 218
320 GOTO 100                                      :rem 95
350 PRINT"RELEASE RATE IS";R                      :rem 191
360 INPUT"ENTER RELEASE RATE (0-15)";R            :rem 185
365 IFR<0ORR>15THEN360                            :rem 74
370 GOTO 100                                      :rem 100
400 PRINT"DECAY RATE IS";D                        :rem 18
410 INPUT"ENTER DECAY RATE (0-15)";D              :rem 12
415 IFD<0ORD>15THEN410                            :rem 38
```

```
420 GOTO 100                                      :rem 96
450 PRINT"PULSE WIDTH IS";100*P/4095              :rem 86
460 INPUT"ENTER PULSE WIDTH (0-100)";P           :rem 191
465 IFP<0ORP>100THEN460                          :rem 115
470 P=P*4095/100                                  :rem 52
480 GOTO 100                                     :rem 102
500 PRINT"SUSTAIN TIME IS";T;"MILLISECONDS"
                                                 :rem 236
510 PRINT"MINIMUM TIME FOR ATTACK/DECAY CYCLE IS:"
                                                  :rem 44
515 PRINT A(A)+D(D);"MILLISECONDS"                 :rem 4
520 INPUT"ENTER TIME IN MILLISECONDS";T          :rem 196
530 GOTO 100                                      :rem 98
550 PRINT"DEAD TIME IS";B;"MILLISECONDS"         :rem 198
560 INPUT"INPUT DEAD TIME IN MILLISECONDS";B
                                                 :rem 214
570 GOTO 100                                     :rem 102
1000 W=1:A=8:D=6:R=9:SL=12:P=2000:T=302          :rem 203
1010 RETURN                                      :rem 162
1100 FORI=1TO8:READF(I):NEXT                     :rem 234
1110 DATA 4291,4817,5407,5728,6430,7217,8101,8538
                                                 :rem 155
1120 RETURN                                      :rem 164
1200 FOR I=0TO15:READ A(I):D(I)=3*A(I):NEXT
                                                 :rem 160
1210 DATA 2,8,16,24,38,56,68,80,100,250,500,800,10
     00,3000,5000,7000                           :rem 186
1220 RETURN                                      :rem 165
1400 FOR I=1TO 8:F(I)=F(I)*2:NEXT:RETURN         :rem 100
1450 FOR I=1TO8:F(I)=F(I)/2:NEXT:RETURN          :rem 110
2000 T0=TI                                        :rem 32
2010 FOR I=1TO1000:NEXT                           :rem 62
2020 PRINT TI-T0                                 :rem 159
```

Program 2. Tune

```
5 DIM D(3,200),F(3,200),G(3,200)                 :rem 254
10 S=54272                                       :rem 245
20 FORI=0TO24:POKES+I,0:NEXT                      :rem 13
30 FORI=1TO3                                     :rem 215
40 J=1                                            :rem 28
50 READ D(I,J),F(I,J):REM GET FREQ & DURATION
                                                  :rem 15
55 G(I,J)=INT(F(I,J)/256):F(I,J)=F(I,J)-256*G(I,J)
                                                 :rem 202
60 IF F(I,J)=0 AND D(I,J)=0 THEN 90              :rem 228
70 J=J+1:GOTO 50                                 :rem 108
90 PRINT "VOICE";I;" ";J;" NOTES"                 :rem 64
100 NEXT I                                        :rem 25
110 POKES+24,15                                   :rem 55
```

```
200 REM SET VOICE ONE                              :rem 186
210 W1=16:REM TRIANGLE WAVEFORM                    :rem 154
220 POKES+5,6*16+0:REM ATTACK=6,DECAY=0             :rem 12
230 POKES+6,10*16+0:REM SUSTAIN=10,RELEASE=0
                                                   :rem 110
300 REM SET VOICE TWO                              :rem 211
310 W2=32:REM SAWTOOTH WAVEFORM                    :rem 189
320 POKES+12,0*16+9:REM ATTACK=0,DECAY=9            :rem 65
330 POKES+13,00*16+0:REM SUSTAIN=00,RELEASE=00
                                                   :rem 203
400 REM SET VOICE THREE                             :rem 82
410 W3=64:REM RECTANGULAR WAVE                      :rem 79
420 POKES+17,3:REM DUTY CYCLE 20%                  :rem 101
430 POKES+19,3*16+10:REM ATTACK=3,DECAY=10:rem 160
440 POKES+20,0*16+0:REM SUSTAIN=0:RELEASE=O
                                                   :rem 104
500 J=0:K=0:L=0:T1=TI:T2=T1:T3=T1                  :rem 207
600 IF T1=<TI THEN GOSUB 1100                       :rem 49
610 IF T2=<TI THEN GOSUB 1200                       :rem 52
620 IF T3=<TI THEN GOSUB 1300                       :rem 55
630 GOTO 600                                       :rem 104
1000 ON I GOTO 1100,1200,1300                      :rem 129
1100 J=J+1:T1=T1+D(1,J)                            :rem 215
1115 IFD(1,J)=0 THEN POKES+4,W1:POKES+11,W2:POKES+
     18,W3:END                                     :rem 217
1117 POKES+4,W1                                     :rem 95
1120 POKES,F(1,J):POKES+1,G(1,J)                    :rem 51
1140 POKES+4,W1+1:RETURN                           :rem 209
1200 K=K+1:T2=T2+D(2,K)                            :rem 222
1210 POKE S+11,W2                                  :rem 136
1220 POKE S+7,F(2,K):POKES+8,G(2,K)                :rem 161
1240 POKES+11,W2+1:RETURN                            :rem 1
1300 L=L+1:T3=T3+D(3,L)                            :rem 229
1310 POKES+18,W3                                   :rem 145
1320 POKES+14,F(3,L):POKES+15,G(3,L)                 :rem 2
1340 POKES+18,W3+1:RETURN                           :rem 10
2000 REM NOTES FOR VOICE ONE                       :rem 110
2010 DATA 30,4051                                   :rem 54
2020 DATA 30,5407,30,4051,30,6069,30,4051 :rem 215
2030 DATA 30,6430,30,6069,30,5407,30,4050 :rem 218
2040 DATA 30,5407,30,4050,30,6069,30,4050 :rem 215
2050 DATA30,6430,30,7217,30,8101,30,4050  :rem 210
2060 DATA30,5407,30,4050,30,6069,30,4050  :rem 217
2070 DATA30,6430,30,6069,30,5407,30,4050  :rem 222
2080 DATA30,5407,30,4050,30,6069,30,4817  :rem 230
2090 DATA60,5407,30,5407,30,4050           :rem 86
2120 DATA 30,5407,30,4051,30,6069,30,4051 :rem 216
2130 DATA 30,6430,30,6069,30,5407,30,4050 :rem 219
2140 DATA 30,5407,30,4050,30,6069,30,4050 :rem 216
2150 DATA30,6430,30,7217,30,8101,30,4050  :rem 211
```

```
2160 DATA30,5407,30,4050,30,6069,30,4050   :rem 218
2170 DATA30,6430,30,6069,30,5407,30,4050   :rem 223
2180 DATA30,5407,30,4050,30,6069,30,4817   :rem 231
2190 DATA120,5407                          :rem 117
2220 DATA 30,5407,30,4051,30,6069,30,4051 :rem 217
2230 DATA 30,6430,30,6069,30,5407,30,4050 :rem 220
2240 DATA 30,5407,30,4050,30,6069,30,4050 :rem 217
2250 DATA30,6430,30,7217,30,8101,30,4050   :rem 212
2260 DATA30,5407,30,4050,30,6069,30,4050   :rem 219
2270 DATA30,6430,30,6069,30,5407,30,4050   :rem 224
2280 DATA30,5407,30,4050,30,6069,30,4817   :rem 232
2290 DATA120,5407                          :rem 118
2320 DATA 30,5407,30,4051,30,6069,30,4051 :rem 218
2330 DATA 30,6430,30,6069,30,5407,30,4050 :rem 221
2340 DATA 30,5407,30,4050,30,6069,30,4050 :rem 218
2350 DATA30,6430,30,7217,30,8101,30,4050   :rem 213
2360 DATA30,5407,30,4050,30,6069,30,4050   :rem 220
2370 DATA30,6430,30,6069,30,5407,30,4050   :rem 225
2380 DATA30,5407,30,4050,30,6069,30,4817   :rem 233
2390 DATA120,5407                          :rem 119
2900 DATA 0,0                              :rem 113
3000 REM NOTES FOR VOICE TWO               :rem 135
3010 DATA990,0                             :rem 220
3020 DATA60,2703,60,2408                   :rem 201
3030 DATA30,2145,30,2025,60,2145            :rem 73
3040 DATA60,2025,60,1804                   :rem 199
3050 DATA30,1607,30,1517,60,1351            :rem 80
3060 DATA60,2703,60,2408                   :rem 205
3070 DATA30,2145,30,2025,60,2145            :rem 77
3080 DATA60,2025,60,1804                   :rem 203
3090 DATA30,1607,30,1517,60,1351            :rem 84
3120 DATA60,2703,60,2408                   :rem 202
3130 DATA30,2145,30,2025,60,2145            :rem 74
3140 DATA60,2025,60,1804                   :rem 200
3150 DATA30,1607,30,1517,60,1351            :rem 81
3160 DATA60,2703,60,2408                   :rem 206
3170 DATA30,2145,30,2025,60,2145            :rem 78
3180 DATA60,2025,60,1804                   :rem 204
3190 DATA30,1607,30,1517,60,1351            :rem 85
3220 DATA60,2703,60,2408                   :rem 203
3230 DATA30,2145,30,2025,60,2145            :rem 75
3240 DATA60,2025,60,1804                   :rem 201
3250 DATA30,1607,30,1517,60,1351            :rem 82
3260 DATA60,2703,60,2408                   :rem 207
3270 DATA30,2145,30,2025,60,2145            :rem 79
3280 DATA60,2025,60,1804                   :rem 205
3290 DATA30,1607,30,1517,60,1351            :rem 86
3900 DATA 0,0                              :rem 114
4000 REM NOTES FOR VOICE THREE               :rem 6
4010 DATA1950,0                             :rem 10
```

```
4020 DATA 60,2703,60,2408                          :rem 202
4030 DATA 30,2703,15,2703,15,2703,60,2025 :rem 215
4040 DATA 30,2703,30,2703,30,3034,30,3034 :rem 206
4050 DATA 15,3215,15,3215,15,3215,15,3215,60,3034
                                               :rem 99
4060 DATA 45,4050,15,3608,45,4050,15,3608 :rem 234
4070 DATA 45,4050,15,3608,15,4050,15,3608,15,3215,
     15,3034                                    :rem 249
4080 DATA 60,2703,60,2408                          :rem 208
4090 DATA 30,2703,15,2703,15,2703,60,2025 :rem 221
4100 DATA 30,2703,30,2703,30,3034,30,3034 :rem 203
4110 DATA 15,3215,15,3215,15,3215,15,3215,60,3034
                                               :rem 96
4120 DATA 45,4050,15,3608,45,4050,15,3608 :rem 231
4130 DATA 45,4050,15,3608,15,4050,15,3608,15,3215,
     15,3034                                    :rem 246
4140 DATA 60,2703,60,2408                          :rem 205
4150 DATA 30,2703,15,2703,15,2703,60,2025 :rem 218
4160 DATA 60,4050,60,4050                          :rem 199
4170 DATA 30,4050,15,4050,15,4050,60,4050 :rem 211
4900 DATA 800,0,0,0                                :rem 147
```

Program 3. Blast-off

```
10 S=54272                                         :rem 245
20 FOR I=STOS+24:POKEI,0:NEXT                      :rem 48
30 POKES+24,15                                     :rem 8
40 FR=0500                                         :rem 254
50 A=0:D=0:SS=15:R=0                               :rem 122
60 W=128:P=1024                                    :rem 35
70 POKES+1,INT(FR/256)                             :rem 17
80 POKES,FR-256*INT(FR/256)                        :rem 66
90 POKES+3,INT(P/256)                              :rem 205
100 POKES+2,P-256*INT(P/256)                       :rem 56
110 POKES+5,16*A+D                                 :rem 225
120 POKES+6,16*SS+R                                :rem 86
200 POKES+4,W+1:REM TURN SOUND ON                  :rem 223
210 FORI=200TO1 STEP-1                             :rem 0
220 FR=FR+100:REM INCREASE FREQUENCY               :rem 215
222 IF I< 45 THEN POKES+24,I/3:REM NEAR THE END TU
    RN DOWN THE VOLUME                             :rem 98
225 F2=INT(FR/256):F1=FR-256*F2                    :rem 224
230 POKES,F1:POKES+1,F2                            :rem 118
240 NEXT I                                         :rem 30
250 POKES+4,W:REM TURN SOUND OFF                   :rem 198
```

Program 4. Laser

```
10 S=54272                                      :rem 245
20 FOR I=STOS+24:POKEI,0:NEXT                    :rem 48
30 POKES+24,143:REM VOLUME AT 15/TURN OFF VOICE TH
   REE                                          :rem 108
40 F50000                                        :rem 46
50 A=0:D=8:SS=15:R=08                           :rem 186
60 W=064:P=1024                                  :rem 34
70 POKES+1,INT(FR/256)                           :rem 17
80 POKES,FR-256*INT(FR/256)                      :rem 66
90 POKES+3,INT(P/256)                           :rem 205
100 POKES+2,P-256*INT(P/256)                     :rem 56
110 POKES+5,16*A+D                              :rem 225
120 POKES+6,16*SS+R                              :rem 86
130 POKES+15,75                                  :rem 63
155 POKES+4,W+3:REM USING W+3 TURNS ON{2 SPACES}GA
    TE AND SYNC                                  :rem 32
160 FORI=1TO25                                   :rem 63
170 POKES+15,120-4*I:REM{2 SPACES}DECREASE FREQ VO
    ICE THREE                                   :rem 180
180 NEXT I                                       :rem 33
185 POKES+4,W                                     :rem 2
```

Sound Sculptor

Todd Touris

With formatted screens and a joystick-controlled pointer, "Sound Sculptor" gives you the ability to quickly and easily create your own music and save your creation.

"Sound Sculptor" uses several graphics screens to take the tedium out of creating data for your music or sound programs. It's not difficult to use and therefore needs little explanation; a basic understanding of the SID chip would probably be helpful, however. "Programming 64 Sound," an article elsewhere in this book, is a good source of information.

Automatic LOAD

Because there are two programs which make up Sound Sculptor, and because the first program automatically loads the second, you need to take some care as you type them in. Make sure you use "The Automatic Proofreader," found in Appendix C, as you enter Sound Sculptor. The Proofreader will help immensely in insuring error-free copies of both programs.

If you're using tape to store Sound Sculptor, put both Program 1 and Program 2 on the same tape. Type in Program 1 first, then save it. Next type in Program 2, saving it on the tape immediately following Program 1. Program 1 will automatically load Program 2.

Much the same process must be used if you have a disk drive. Both programs should be typed in and saved to the same disk. Make sure you save Program 2 with SAVE"2",8. That's the filename Program 1 will look for.

Run Program 1, press T or D (tape or disk), and wait patiently. You should be presented with a main menu. Press the f1 function key. (Don't worry about loading a file right now.) You'll then be asked to choose a sound between 0 and 1250. Enter a number and press RETURN.

You will see a menu which allows you to set one of three voices, work on the filter settings, clear the sound, choose a new sound, change joystick speed, or quit. If you don't clear the sound, the settings will be random and probably won't produce any sound at all. Use the keyboard to make your selection and plug a joystick into port 2.

Set the Volume First

Before you jump to the voice settings, make sure to go to the filter display and set the volume control, or you won't be able to hear anything. To change the various settings, you simply move the sprite arrow over the appropriate display and press the fire button. When a word or character is in *reverse display,* it means that the particular setting is on, or if the display is a scale (+ signs), it shows what value that setting contains.

Select a waveform. There are four available: sawtooth, pulse, triangle, and noise. Although you can set more than one at a time, it's not recommended. (See "Programming 64 Sound" for a good reason.) If you set the noise waveform while another is on, the voice must be cleared to produce any sound. Before you select the noise waveform, then, make sure all the others are turned off.

If you choose a pulse waveform, you should also set the pulse width. This adjustment changes for every pixel the arrow passes, not just the + symbols. If you've set the volume and the attack, decay, sustain, and release values already, you can hear a slight difference in the background sound if you turn your monitor's volume up to high.

Set an ADSR (attack, decay, sustain, and release) envelope by selecting values. If you want to hear the sound while you're experimenting, set the sustain to anything but the leftmost +, then hit the appropriate function key (see below for triggering the voices). If you have the frequency and waveform set, you should hear a steady tone. Change the note, octave, or waveform and listen to the difference.

Choose the frequency by setting the octave and the note.

Synchronization and ring modulation are rather complex, but they can create some interesting sounds. Experimenting with them is probably the best way to hear how they affect sounds. There are just a couple of things to keep in mind. First, the voice that's using synchronization or ring modulation must be set to the triangle waveform. Second, make sure you set the frequency for the voice that's indicated in the bottom box on the display. (If you're using voice 1, for instance, you need to set the frequency of voice 3.)

Filters

The filter display can be accessed from the main menu. (You can return from any display to the main menu just by pressing the space bar.) Once you see the filter setting screen, you can choose which type of filter to set (high pass, band pass, or low pass), the voice to be filtered (E stands for external, used if you're routing sounds to an external speaker), and the cutoff frequency. As with the pulse width, the cutoff frequency changes at each pixel, not just each + symbol.

Resonance will make the frequencies around the filter cutoff area louder. The very bottom box on the screen, *Voice 3 Output*, will shut off voice 3 if it's set (shown by reverse video). If in normal text, voice 3 is not affected. It's a good idea to shut off voice 3 when using synchronization or ring modulation with voice 1, since it will cut down on any extra noise.

Playing Sounds

To trigger the voices, you must use the function keys (f1 for voice one, f3 for voice two, f5 for voice three, and f7 for all voices). If the voice is off, it should go through attack and decay, and then remain at the sustain level; when the key is pressed again, the sound should be released and fall to zero volume. When pressing the function keys or switching a setting, you must be careful. The program is very fast and the keys are very responsive; sometimes the voice or setting can be triggered twice, so hit the keys sharply.

When you are finished experimenting with the various settings, press the space bar to return to the selection menu. You can continue working on more sounds, or you can press f8 to quit. When you quit, you will get another menu with three options.

Saving Sounds

The first option is to save a series of sounds on tape or disk as a file (depending on your earlier selection). You need to provide the beginning and ending sounds (separating the numbers with a comma) and then a filename. Make sure you have a disk in the drive or a tape in the Datassette.

Later, you can load these sounds back into the computer by pressing f3 at the beginning of the program instead of going right to the design/review routine. This feature allows you to build a library of various sounds.

Your second choice is to create DATA statements of your sound or sounds. After pressing the f3 key, you need to respond with the beginning and ending number(s) for the sound(s) you want to make DATA statements for. As soon as you press the RETURN key, the DATA statements appear. Hit RETURN several times (usually just once or twice more), and you'll see only the DATA statements on the screen. In fact, if you type LIST, the DATA statement lines will be the only ones in memory. If you want, you can save just the DATA statements as another program file, ready for appending to or merging with another program later.

With the program below, you can use these DATA statements to incorporate complex and fast sound effects into your BASIC programs.

```
1000 FORL=0 TO 42:READ DA:POKE828+L,DA:NEXTL
1010 DATA 166,2,165,251,133,253,165,252,133,254,22
     4,0,240,16,169,25,24,101
1020 DATA 253,133,253,169,0,101,254,133,254,202,20
     8,240,160,0,177,253
1030 DATA 153,0,212,200,192,26,208,246,96
```

This is a machine language routine that's POKEd into the cassette buffer (starting at location 828), but it's relocatable and can be put anywhere in free memory. To use it, you must POKE the values from the DATA statements created by Sound Sculptor into any free memory. For example, you could put one sound's data into the block of free memory beginning at 49152 with:

```
10 FORL=0 TO 24:READ SND:POKE 49152+L,SND:NEXTL
```

If you have more sounds, POKE the DATA into memory immediately following the first. Each sound created by the Sculptor includes 25 valid numbers (that's why the FOR-NEXT loop above reads FOR L=0 TO 24). The last DATA statement (no matter how many sounds you create DATA statements for) will have extra values. These will do no harm as long as you read only 25 values for each sound.

Next, POKE the starting address of the sounds into locations 251 and 252. For the example above, this would be accomplished by:

```
20 POKE252,49152/256:POKE251,49152-256*PEEK(252)
```

Now you should have a short program which reads the values from the DATA statements and POKEs them into memory. Run it and the sound's values are stored.

A Fast Sound Switch

This has to be done only once. Whenever you wish to call upon a certain sound, just POKE the sound number into location 2. For example, POKE 2,0 selects the first sound in memory. POKE 2,1 would call the second sound. Follow this with a SYS 828 (or to whatever memory location you have relocated the machine language routine), and you now have your sound in the SID chip. By doing this, you can switch various sounds in and out of the SID at lightning speed.

You need to turn on the voice you're using, of course. You can do this with a line which includes:

POKE 54272+4,PEEK(*n*)OR1

where *n* is the first location of that sound. It would be 49152 if that's where you earlier POKEd the sound's DATA values.

Turning off the sound can be done by:

POKE 54272+4,PEEK(*n*)AND254

The same process applies to turning on voices 2 and 3, except you'd use 54283 and 54290 respectively instead of 54272. It's a good idea to turn off the voice, then turn it back on, between calling different sounds.

If you POKEd two sounds' values into memory, starting at location 49152, for example, the routine to call those sounds might look like this.

```
10 POKE 2,0
20 SYS 828
30 POKE 54272+4,PEEK(49152+4)OR 1
40 FOR T=0TO1000:NEXT
50 POKE 54272+4,PEEK(49152+4)AND 254
60 POKE 2,1
70 SYS 828
80 POKE 54272+4,PEEK(49177+4)OR 1
90 FOR T=0TO1000:NEXT
100 POKE 54272+4,PEEK(49177+4)AND 254
```

Create two sounds of your own with the Sculptor and form the DATA statements. POKE those into memory as described earlier, then type in and run the routine above. You should hear your two sounds, one after the other. (Notice that the second sound turns on voice 1 by PEEKing 49177+4. That's the location of the control register for the second sound. You get that location by adding 25 to the first address used to store sound data, in this case 49152. Each additional sound can turn the voice on and off by PEEKing the location 25 higher than the previous sound.)

Tape users: Program 1 automatically loads Program 2. It's recommended that you save both on the same tape, Program 2 last.
Disk users: Save Program 2 as "2". Make sure both programs are on the same disk.

Program 1. Sound Sculptor—ML Loader

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C, to enter the following two programs.

```
50 POKE53281,11:POKE646,1:POKE53280,11      :rem 132
80 PRINT"{CLR}":PRINT:PRINT"{6 SPACES}PLEASE WAIT
   {SPACE}ONE MOMENT..."                     :rem 201
200 REM ML PROGRAM POKER                     :rem 168
210 FORL=49152TO50702                        :rem 169
220 READDA:POKEL,DA:NEXT                      :rem 20
240 PRINT"{CLR}{13 RIGHT}{11 DOWN}{RVS}T{OFF}APE O
    R {RVS}D{OFF}ISK"                        :rem 108
250 GETT$:IFT$=""THEN250                     :rem 119
255 IFT$<>"D"ANDT$<>"T"THEN250               :rem 200
260 IFT$="D"THEN380                           :rem 46
300 POKE631,76:POKE632,207:POKE633,13:POKE198,3
                                             :rem 189
350 FORT=1TO1000:NEXT:GOTO1000                :rem 82
380 POKE50660,8:POKE50662,8                  :rem 255
400 POKE631,76:POKE632,207:POKE633,34:POKE634,50:P
    OKE635,34:POKE636,44                      :rem 36
405 POKE637,56                               :rem 255
410 POKE638,58:POKE639,13:POKE198,9            :rem 9
1000 REM ML DATA                              :rem 88
49152 DATA 32, 140, 197, 160, 0, 177         :rem 252
49158 DATA 78, 153, 0, 212, 200, 192         :rem 252
49164 DATA 25, 208, 246, 32, 92, 194           :rem 9
49170 DATA 165, 197, 201, 60, 240, 23         :rem 44
49176 DATA 169, 16, 45, 0, 220, 208          :rem 207
49182 DATA 225, 165, 2, 240, 6, 32           :rem 150
49188 DATA 86, 192, 76, 0, 192, 32           :rem 172
49194 DATA 48, 192, 76, 0, 192, 96           :rem 177
49200 DATA 162, 21, 189, 24, 197, 202         :rem 46
49206 DATA 205, 1, 208, 48, 8, 189           :rem 163
49212 DATA 24, 197, 205, 1, 208, 48          :rem 205
49218 DATA 4, 202, 16, 236, 96, 189          :rem 217
49224 DATA 48, 197, 133, 75, 232, 189         :rem 68
```

```
49230 DATA 48, 197, 133, 76, 108, 75          :rem 14
49236 DATA 0, 234, 162, 15, 189, 72           :rem 209
49242 DATA 197, 202, 205, 1, 208, 48          :rem 254
49248 DATA 8, 189, 72, 197, 205, 1            :rem 173
49254 DATA 208, 48, 4, 202, 16, 236           :rem 206
49260 DATA 96, 189, 88, 197, 133, 75          :rem 32
49266 DATA 232, 189, 88, 197, 133, 76         :rem 79
49272 DATA 108, 75, 0, 234, 96, 24            :rem 161
49278 DATA 173, 0, 208, 233, 142, 144         :rem 49
49284 DATA 247, 74, 74, 74, 74, 133           :rem 225
49290 DATA 77, 234, 32, 175, 192, 76          :rem 16
49296 DATA 96, 196, 96, 234, 234, 24          :rem 26
49302 DATA 173, 0, 208, 233, 74, 144          :rem 249
49308 DATA 245, 41, 240, 160, 3, 81           :rem 199
49314 DATA 253, 41, 240, 81, 253, 145         :rem 45
49320 DATA 253, 32, 175, 192, 76, 140         :rem 50
49326 DATA 196, 160, 3, 177, 253, 74          :rem 12
49332 DATA 74, 74, 74, 10, 170, 160           :rem 204
49338 DATA 0, 189, 0, 197, 145, 253           :rem 216
49344 DATA 232, 200, 189, 0, 197, 145         :rem 52
49350 DATA 253, 24, 169, 8, 229, 77           :rem 222
49356 DATA 234, 170, 240, 15, 177, 253        :rem 103
49362 DATA 74, 145, 253, 136, 177, 253        :rem 112
49368 DATA 106, 145, 253, 200, 202, 208       :rem 143
49374 DATA 241, 96, 24, 173, 0, 208           :rem 210
49380 DATA 233, 144, 144, 8, 169, 128         :rem 59
49386 DATA 32, 32, 193, 76, 198, 195          :rem 25
49392 DATA 169, 64, 32, 32, 193, 76           :rem 223
49398 DATA 181, 195, 24, 173, 0, 208          :rem 11
49404 DATA 233, 144, 176, 8, 169, 32          :rem 7
49410 DATA 32, 32, 193, 76, 215, 195          :rem 3
49416 DATA 169, 16, 32, 32, 193, 76           :rem 217
49422 DATA 232, 195, 169, 4, 32, 32           :rem 207
49428 DATA 193, 76, 249, 195, 169, 2          :rem 27
49434 DATA 32, 32, 193, 76, 10, 196           :rem 211
49440 DATA 160, 4, 81, 253, 145, 253          :rem 253
49446 DATA 96, 234, 234, 24, 173, 0           :rem 211
49452 DATA 208, 233, 133, 144, 245, 170       :rem 147
49458 DATA 169, 0, 160, 2, 145, 253           :rem 209
49464 DATA 200, 177, 253, 41, 240, 72         :rem 48
49470 DATA 145, 253, 138, 162, 5, 136         :rem 54
49476 DATA 10, 145, 253, 200, 177, 253        :rem 100
49482 DATA 42, 145, 253, 136, 177, 253        :rem 110
49488 DATA 202, 208, 241, 200, 177, 253       :rem 151
49494 DATA 41, 15, 145, 253, 104, 24          :rem 254
49500 DATA 113, 253, 145, 253, 76, 193        :rem 101
49506 DATA 194, 96, 234, 160, 5, 32           :rem 213
49512 DATA 133, 193, 76, 208, 194, 160        :rem 107
49518 DATA 5, 32, 138, 193, 76, 228           :rem 221
49524 DATA 194, 160, 6, 32, 133, 193          :rem 2
```

```
49530 DATA 76, 246, 194, 160, 6, 32          :rem 212
49536 DATA 138, 193, 76, 10, 195, 162         :rem 64
49542 DATA 240, 76, 140, 193, 162, 15         :rem 50
49548 DATA 134, 251, 24, 173, 0, 208          :rem 255
49554 DATA 233, 133, 144, 205, 74, 74         :rem 54
49560 DATA 74, 166, 251, 16, 4, 10            :rem 154
49566 DATA 10, 10, 10, 81, 253, 37            :rem 148
49572 DATA 251, 81, 253, 145, 253, 96         :rem 63
49578 DATA 234, 234, 234, 234, 96, 234        :rem 115
49584 DATA 234, 234, 234, 234, 160, 1         :rem 48
49590 DATA 24, 173, 0, 208, 233, 133          :rem 251
49596 DATA 48, 240, 10, 145, 253, 76          :rem 11
49602 DATA 28, 195, 234, 24, 173, 0           :rem 206
49608 DATA 208, 233, 133, 144, 225, 74        :rem 103
49614 DATA 74, 74, 74, 74, 162, 1             :rem 116
49620 DATA 168, 240, 6, 138, 10, 136          :rem 253
49626 DATA 208, 252, 170, 138, 160, 2         :rem 49
49632 DATA 81, 253, 145, 253, 76, 86          :rem 16
49638 DATA 195, 234, 234, 234, 96, 173        :rem 120
49644 DATA 0, 208, 233, 133, 144, 248         :rem 49
49650 DATA 10, 160, 2, 81, 253, 41            :rem 144
49656 DATA 240, 81, 253, 145, 253, 76         :rem 62
49662 DATA 48, 195, 234, 234, 234, 234        :rem 112
49668 DATA 234, 81, 253, 41, 15, 81           :rem 215
49674 DATA 253, 145, 253, 96, 173, 0          :rem 12
49680 DATA 208, 233, 133, 144, 248, 74        :rem 108
49686 DATA 74, 74, 160, 3, 234, 32            :rem 165
49692 DATA 5, 194, 76, 68, 195, 169           :rem 241
49698 DATA 64, 32, 65, 194, 76, 44            :rem 183
49704 DATA 196, 169, 32, 32, 65, 194          :rem 17
49710 DATA 76, 61, 196, 169, 16, 32           :rem 219
49716 DATA 65, 194, 76, 78, 196, 169          :rem 38
49722 DATA 128, 32, 65, 194, 76, 27           :rem 221
49728 DATA 196, 160, 3, 81, 253, 145          :rem 11
49734 DATA 253, 96, 24, 169, 128, 113         :rem 64
49740 DATA 251, 145, 251, 136, 208, 246       :rem 152
49746 DATA 173, 0, 220, 41, 16, 240           :rem 197
49752 DATA 249, 96, 96, 234, 32, 228          :rem 24
49758 DATA 255, 201, 133, 48, 247, 201        :rem 105
49764 DATA 137, 16, 243, 201, 133, 208        :rem 99
49770 DATA 4, 32, 137, 194, 96, 201           :rem 213
49776 DATA 134, 208, 4, 32, 149, 194          :rem 14
49782 DATA 96, 201, 135, 208, 4, 32           :rem 210
49788 DATA 161, 194, 96, 32, 137, 194         :rem 77
49794 DATA 32, 149, 194, 32, 161, 194         :rem 67
49800 DATA 96, 169, 1, 160, 4, 81             :rem 111
49806 DATA 78, 145, 78, 141, 4, 212           :rem 216
49812 DATA 96, 169, 1, 160, 11, 81            :rem 160
49818 DATA 78, 145, 78, 141, 11, 212          :rem 9
49824 DATA 96, 169, 1, 160, 18, 81            :rem 170
```

```
49830 DATA 78, 145, 78, 141, 18, 212          :rem 10
49836 DATA 96, 41, 15, 170, 160, 16           :rem 212
49842 DATA 169, 43, 145, 251, 136, 208        :rem 110
49848 DATA 251, 232, 138, 168, 169, 171       :rem 170
49854 DATA 145, 251, 96, 169, 5, 133          :rem 17
49860 DATA 251, 169, 7, 133, 252, 160         :rem 55
49866 DATA 3, 177, 253, 76, 173, 194          :rem 25
49872 DATA 169, 117, 133, 251, 169, 5         :rem 65
49878 DATA 133, 252, 160, 5, 177, 253         :rem 63
49884 DATA 74, 74, 74, 74, 170, 76            :rem 184
49890 DATA 176, 194, 169, 157, 133, 251       :rem 172
49896 DATA 169, 5, 133, 252, 160, 5           :rem 219
49902 DATA 177, 253, 41, 15, 170, 76          :rem 7
49908 DATA 176, 194, 169, 197, 133, 251       :rem 176
49914 DATA 169, 5, 133, 252, 160, 6           :rem 211
49920 DATA 177, 253, 74, 74, 74, 74           :rem 227
49926 DATA 170, 76, 176, 194, 169, 237        :rem 129
49932 DATA 133, 251, 169, 5, 133, 252         :rem 53
49938 DATA 160, 6, 177, 253, 41, 15           :rem 216
49944 DATA 170, 76, 176, 194, 169, 173        :rem 128
49950 DATA 133, 251, 169, 4, 133, 252         :rem 52
49956 DATA 160, 1, 177, 253, 74, 74           :rem 222
49962 DATA 74, 74, 170, 76, 176, 194          :rem 27
49968 DATA 169, 237, 133, 251, 169, 5         :rem 74
49974 DATA 133, 252, 160, 2, 177, 253         :rem 57
49980 DATA 74, 74, 74, 74, 170, 76            :rem 181
49986 DATA 176, 194, 169, 181, 133, 251       :rem 175
49992 DATA 169, 6, 133, 252, 160, 3           :rem 215
49998 DATA 177, 253, 41, 15, 170, 76          :rem 22
50004 DATA 176, 194, 169, 78, 133, 251        :rem 105
50010 DATA 169, 5, 133, 252, 169, 1           :rem 194
50016 DATA 162, 1, 160, 2, 72, 49             :rem 91
50022 DATA 253, 240, 31, 138, 168, 177        :rem 93
50028 DATA 251, 201, 127, 16, 7, 169          :rem 248
50034 DATA 128, 24, 113, 251, 145, 251        :rem 82
50040 DATA 232, 232, 232, 232, 104, 10        :rem 65
50046 DATA 224, 17, 240, 3, 76, 98            :rem 156
50052 DATA 195, 76, 82, 194, 138, 168         :rem 65
50058 DATA 177, 251, 201, 127, 48, 232        :rem 95
50064 DATA 169, 128, 24, 113, 251, 145        :rem 93
50070 DATA 251, 76, 120, 195, 49, 253         :rem 48
50076 DATA 240, 11, 138, 168, 177, 251        :rem 98
50082 DATA 201, 127, 16, 14, 76, 80           :rem 195
50088 DATA 194, 138, 168, 177, 251, 201       :rem 158
50094 DATA 127, 48, 3, 76, 72, 194            :rem 165
50100 DATA 96, 169, 170, 133, 251, 169        :rem 98
50106 DATA 6, 133, 252, 162, 6, 169           :rem 199
50112 DATA 64, 160, 4, 76, 154, 195           :rem 202
50118 DATA 169, 189, 133, 251, 169, 6         :rem 60
50124 DATA 133, 252, 162, 5, 169, 128         :rem 43
```

```
50130 DATA 160, 4, 76, 154, 195, 169          :rem 0
50136 DATA 90, 133, 251, 169, 6, 133          :rem 250
50142 DATA 252, 162, 6, 169, 32, 160          :rem 246
50148 DATA 4, 76, 154, 195, 169, 109          :rem 12
50154 DATA 133, 251, 169, 6, 133, 252         :rem 42
50160 DATA 162, 6, 169, 16, 160, 4            :rem 147
50166 DATA 76, 154, 195, 169, 153, 133        :rem 110
50172 DATA 251, 169, 7, 133, 252, 162         :rem 45
50178 DATA 15, 169, 4, 160, 4, 76             :rem 109
50184 DATA 154, 195, 169, 113, 133, 251       :rem 149
50190 DATA 169, 7, 133, 252, 162, 15          :rem 251
50196 DATA 169, 2, 160, 4, 76, 154            :rem 159
50202 DATA 195, 169, 45, 133, 251, 169        :rem 102
50208 DATA 7, 133, 252, 162, 15, 169          :rem 251
50214 DATA 128, 160, 3, 76, 154, 195          :rem 253
50220 DATA 169, 201, 133, 251, 169, 4         :rem 37
50226 DATA 133, 252, 162, 9, 169, 64          :rem 1
50232 DATA 160, 3, 76, 154, 195, 169          :rem 2
50238 DATA 25, 133, 251, 169, 5, 133          :rem 250
50244 DATA 252, 162, 9, 169, 32, 160          :rem 252
50250 DATA 3, 76, 154, 195, 169, 105          :rem 1
50256 DATA 133, 251, 169, 5, 133, 252         :rem 44
50262 DATA 162, 9, 169, 16, 160, 3            :rem 152
50268 DATA 76, 154, 195, 234, 162, 49         :rem 64
50274 DATA 160, 0, 169, 95, 133, 251          :rem 252
50280 DATA 169, 4, 133, 252, 138, 145         :rem 47
50286 DATA 251, 200, 200, 232, 192, 16        :rem 81
50292 DATA 208, 246, 160, 1, 177, 253         :rem 47
50298 DATA 162, 255, 232, 74, 208, 252        :rem 104
50304 DATA 138, 133, 77, 10, 168, 169         :rem 51
50310 DATA 128, 113, 251, 145, 251, 96        :rem 88
50316 DATA 162, 0, 160, 0, 169, 167           :rem 195
50322 DATA 133, 251, 169, 4, 133, 252         :rem 37
50328 DATA 189, 112, 197, 145, 251, 200       :rem 147
50334 DATA 200, 232, 192, 24, 208, 244        :rem 84
50340 DATA 160, 3, 177, 253, 74, 74           :rem 203
50346 DATA 74, 74, 10, 168, 24, 169           :rem 212
50352 DATA 128, 113, 251, 145, 251, 96        :rem 94
50358 DATA 234, 234, 32, 96, 196, 32          :rem 6
50364 DATA 140, 196, 32, 193, 194, 32         :rem 50
50370 DATA 198, 195, 32, 181, 195, 32         :rem 57
50376 DATA 215, 195, 32, 232, 195, 32         :rem 50
50382 DATA 249, 195, 32, 10, 196, 32          :rem 1
50388 DATA 208, 194, 32, 228, 194, 32         :rem 58
50394 DATA 246, 194, 32, 10, 195, 76          :rem 7
50400 DATA 0, 192, 234, 234, 234, 32          :rem 235
50406 DATA 28, 195, 32, 48, 195, 32           :rem 211
50412 DATA 86, 195, 32, 27, 196, 32           :rem 210
50418 DATA 44, 196, 32, 61, 196, 32           :rem 209
50424 DATA 78, 196, 32, 68, 195, 76           :rem 227
```

```
300 PRINT" B{8 SPACES}BWAVEFORMB{10 SPACES}B
                                          :rem 177
310 PRINT" B OLOLOL JCCCCCCCCK{2 SPACES}";:rem 224
315 PRINT"NOISE{3 SPACES}B{10 SPACES}B{28 SPACES}B
    "                                      :rem 46
320 PRINT" BPULSE WIDTH +++++++++++++++B :rem 129
330 PRINT" JCCCCCCCCCCCCCCCCCCCCCCCCCCCK  :rem 46
340 PRINT" UCCCCCCCCCCCCCCCCCCCCCCCCCCCI  :rem 56
350 PRINT" BSYNCHRONIZATION{3 SPACES}USE VOICE B
                                          :rem 192
360 PRINT" BRING MODULATION{6 SPACES}#"SR".
    {2 SPACES}B                            :rem 50
370 PRINT" JCCCCCCCCCCCCCCCCCCCCCCCCCCCK";
                                          :rem 143
380 PRINT"{HOME}"                         :rem 127
390 A=(V-1)*7:S=S+A:POKE254,S/256:POKE253,S-256*PE
    EK(254)                               :rem 223
400 SYSVCH                                :rem 116
410 GOTO910                               :rem 104
420 PRINT"{CLR}";:POKE2,255               :rem 152
430 REM FILTER DISPLAY                     :rem 87
440 PRINT"{RVS}CCCCCCCCCCCCFILTER SETTINGSCCCCCCCC
    CCCCC{OFF}"                            :rem 79
450 PRINT"UCCCCCCCCCCCIUCCCCCCCCCCCCCCCCI  :rem 21
460 PRINT"BFILTER TYPEBBCUTOFF FREQUENCYB :rem 210
470 PRINT"B{11 SPACES}BB+++++++++++++++B    :rem 2
480 PRINT"B HIGH PASS BJCCCCCCCCCCCCCCCCK :rem 235
490 PRINT"B{11 SPACES}BUCCCCCCCCCCCCCCCCI :rem 158
500 PRINT"B BAND PASS BBVOICES{2 SPACES}FILTEREDB
                                          :rem 176
510 PRINT"B{11 SPACES}BB 1{3 SPACES}2{3 SPACES}3
    {3 SPACES}E{2 SPACES}B                 :rem 40
520 PRINT"B LOW{2 SPACES}PASS BJCCCCCCCCCCCCCCCCK
                                          :rem 184
530 PRINT"JCCCCCCCCCCKUCCCCCCCCCCCCCCCCI   :rem 11
540 PRINT"{13 SPACES}B{3 SPACES}RESONANCE
    {4 SPACES}B                           :rem 106
550 PRINT"{13 SPACES}B+++++++++++++++B    :rem 125
560 PRINT"{13 SPACES}JCCCCCCCCCCCCCCCCK    :rem 15
570 PRINT"{RVS}CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
    CCCCC{OFF}";                          :rem 196
580 PRINT"{13 SPACES}UCCCCCCCCCCCCCCCCI    :rem 26
590 PRINT"{13 SPACES}B{3 SHIFT-SPACE}
    {SHIFT-SPACE}VOLUME{5 SHIFT-SPACE}B    :rem 73
600 PRINT"{13 SPACES}B+++++++++++++++B    :rem 121
610 PRINT"{13 SPACES}JCCCCCCCCCCCCCCCCK    :rem 11
620 PRINT"{13 SPACES}UCCCCCCCCCCCCCCCCI    :rem 21
630 PRINT"{13 SPACES}B{RVS}VOICE #3 OUTPUT{OFF} B
                                           :rem 45
640 PRINT"{13 SPACES}JCCCCCCCCCCCCCCCCK    :rem 14
```

```
650 S=S+21:POKE254,S/256:POKE253,S-256*PEEK(254):S
    YSFCH:GOTO910                              :rem 249
660 REM INITIALIZATION                         :rem 168
670 SS=9758:POKE78,30:POKE79,38:SN=0:VCH=50360:FCH
    =50405:POKE53236,10                        :rem 17
680 POKE53248,24:POKE53249,50:POKE51,29:POKE52,38:
    POKE55,29:POKE56,38                          :rem 8
690 PRINT"{CLR}"                                 :rem 3
700 PRINT"{11 DOWN}"TAB(7)"WELCOME TO SOUND SCULPT
    OR"                                        :rem 122
710 FORL=1TO2000:NEXT                           :rem 23
720 PRINT"{CLR}"                               :rem 253
730 PRINT"{3 DOWN}"TAB(15)"{RVS}MAIN MENU{OFF}"
                                               :rem 110
740 PRINT"{2 DOWN}"TAB(14)"CHOOSE ONE:"         :rem 60
750 PRINT"{2 DOWN}"TAB(7)"{RVS}F1{OFF} DESIGN/REVI
    EW SOUNDS"                                 :rem 228
760 PRINT:PRINTTAB(7)"{RVS}F3{OFF} LOAD SOUND FILE
     "                                         :rem 122
770 GETA$:IFA$<"{F1}"ORA$>"{F3}"THEN770        :rem 241
780 ONASC(A$)-132GOTO860,1340                   :rem 87
790 REM JOYSTICK SPEED                         :rem 101
800 PRINT"{CLR}{12 DOWN}{3 SPACES}SELECT A SPEED B
    ETWEEN 0 AND 15."                          :rem 219
810 PRINT"{4 SPACES}0 - SLOWEST{6 SPACES}15 - FAST
    EST"                                       :rem 165
820 INPUTPS                                    :rem 205
830 IFPS<0ORPS>15THENPRINT"NUMBER NOT ACCEPTABLE":
    GOTO830                                    :rem 173
840 POKE53236,16-PS:GOTO910                     :rem 62
850 REM SOUND DESIGN/REVIEW                    :rem 197
860 PRINT"{CLR}"                                 :rem 2
870 PRINT"{11 DOWN} WHICH SOUND DO YOU WISH TO WOR
    K ON?"                                     :rem 180
880 PRINT"{2 SPACES}(NUMBER BETWEEN 0 & 1250 PLEAS
    E)                                          :rem 75
890 INPUTSN                                    :rem 210
900 IFSN<0ORSN>1250THENPRINT"NUMBER NOT ACCEPTABLE
    ":GOTO890                                   :rem 15
910 POKE53269,0:PRINT"{CLR}{RVS}SOUND #";SN"{OFF}
    {HOME}{3 DOWN}"TAB(15)"CHOOSE ONE:"         :rem 49
920 S=SS+SN*25                                  :rem 46
930 POKE79,S/256:POKE78,S-256*PEEK(79)         :rem 183
940 PRINT:PRINTTAB(8)"{RVS}1{OFF} - DISPLAY VOICE
    {SPACE}#1"                                 :rem 119
950 PRINT:PRINTTAB(8)"{RVS}2{OFF} - DISPLAY VOICE
    {SPACE}#2"                                 :rem 122
960 PRINT:PRINTTAB(8)"{RVS}3{OFF} - DISPLAY VOICE
    {SPACE}#3"                                 :rem 125
```

```
970 PRINT:PRINTTAB(8)"{RVS}4{OFF} - DISPLAY FILTER
     SETTINGS"                                :rem 234
980 PRINT:PRINTTAB(8)"{RVS}5{OFF} - CLEAR SOUND"
                                              :rem 143
990 PRINT:PRINTTAB(8)"{RVS}6{OFF} - NEW SOUND NUMB
    ER"                                       :rem 221
1000 PRINT:PRINTTAB(8)"{RVS}7{OFF} - CHANGE JOYSTI
     CK SPEED"                                 :rem 72
1010 PRINT:PRINTTAB(8)"{RVS}8{OFF} - QUIT"   :rem 6
1020 GETC$:IFC$<"1"ORC$>"8"THEN1020         :rem 159
1030 ONVAL(C$)GOTO1040,1050,1060,1070,1080,860 ,80
     0,1100                                    :rem 68
1040 V=1:SR=3:POKE53269,1:GOTO140           :rem 175
1050 V=2:SR=1:POKE53269,1:GOTO140           :rem 175
1060 V=3:SR=2:POKE53269,1:GOTO140           :rem 178
1070 POKE53269,1:GOTO420                    :rem 102
1080 FORL=0TO24:POKES+L,0:NEXT:GOTO910      :rem 135
1090 REM QUIT                               :rem 241
1100 PRINT"{CLR}{7 DOWN}"                   :rem 157
1110 PRINT TAB(14)"CHOOSE ONE:"             :rem 254
1120 PRINT:PRINTTAB(6)"{RVS}F1{OFF} - SAVE SOUND F
     ILE "                                    :rem 218
1130 PRINT:PRINTTAB(6)"{RVS}F3{OFF} - CONVERT TO D
     ATA STATEMENTS"                          :rem 235
1140 PRINT:PRINTTAB(6)"{RVS}F5{OFF} - END":rem 223
1150 GETA$:IFA$<"{F1}"ORA$>"{F5}"THEN1150   :rem 68
1160 ONASC(A$)-132GOTO1220 ,1170,1420       :rem 155
1170 PRINT"{CLR}{8 DOWN}"                   :rem 181
1180 PRINT"{2 SPACES}ENTER SOUNDS YOU WANT TO CONV
     ERT"                                     :rem 240
1190 PRINT"{6 SPACES}(START,END)";          :rem 185
1200 ER=1:GOTO110                           :rem 205
1210 REM SAVE SOUNDS ROUTINE                :rem 217
1220 PRINT"{CLR}{8 DOWN}"                   :rem 177
1230 PRINT"{2 SPACES}ENTER SOUNDS YOU WISH TO SAVE
     "                                        :rem 251
1240 PRINT"{6 SPACES}(START,END)";          :rem 181
1250 INPUTB,E:IFB<0ORE>1250ORB>ETHENPRINT"BAD INPU
     T":GOTO1250                              :rem 102
1260 S=B*25+9758:F=9758+E*25+25              :rem 97
1270 POKE79,S/256:POKE78,S-256*PEEK(79):POKE254,F/
     256:POKE253,F-256*PEEK(254)              :rem 161
1280 INPUT"{3 SPACES}WHAT DO YOU WISH TO NAME THE
     {SPACE}FILE";NM$:IFNM$=""THEN1280         :rem 96
1290 T=LEN(NM$):POKE2,T                     :rem 103
1300 FORJ=1TOT:POKE50944-J+T,ASC(RIGHT$(NM$,J)):NE
     XTJ                                      :rem 254
1310 SYS50659:SYS50692                        :rem 12
```

```
1320 PRINT:PRINTNM$" FILE HAS BEEN SAVED":PRINT"TH
     ANKYOU":END                               :rem 53
1330 REM LOAD ROUTINE                          :rem 241
1340 IFPEEK(50660)=1THENPRINT"{CLR}":POKE2,0:SYS50
     659:SYS50682:GOTO860                      :rem 87
1360 INPUT"{8 SPACES}FILENAME";NM$:T=LEN(NM$):POKE
     2,T:IFT=0THEN1360                         :rem 49
1370 FORJ=1TOT:POKE50944-J+T,ASC(RIGHT$(NM$,J)):NE
     XTJ                                       :rem 5
1380 PRINT"{CLR}":SYS50659:SYS50682            :rem 176
1390 IFST=66THENPRINT"{7 RIGHT}FILE NOT FOUND":GOT
     O1350                                     :rem 64
1400 GOTO860                                   :rem 156
1410 REM END                                   :rem 129
1420 PRINT"{CLR}THANKYOU":END                  :rem 175
1430 POKE2040,11:FORL=0TO24:READSP:POKE704+L,SP:NE
     XTL:POKE53287,7                           :rem 11
1440 FORL=25TO63:POKE704+L,0:NEXTL:GOTO670 :rem 88
1450 DATA48,0,0,56,0,0,60,0,0,62,0,0,45,0,0,36,0,0
     ,4,0,0,2,0,0,2                            :rem 19
```

64 Hi-Res Graphics Editor

Gregg Peele

Just as a word processor allows you to expand your writing skills by giving you power to manipulate text freely, "Hi-Res Graphics Editor" allows you to easily draw, erase, and edit images on the 64's hi-res screen. Once you have finished your drawing, you can even send the results to your 1525 printer. Joystick needed.

Creating, changing, even saving intricate drawings on your Commodore 64's hi-res screen is simple with the Editor. Using a joystick and sprites, parts of pictures can be imprinted onto a sprite and planted on another area of the screen. You can even enlarge the sprite to full-screen size to edit it more precisely.

Type It In with MLX

"Hi-Res Graphics Editor" is in two parts. (Three, if you use the optional automatic load routine. See the next section, "Autoload," for details.) First you must type in Program 1, "Machine Language for Hi-Res Graphics Editor," using the MLX program found in Appendix D. MLX makes it simple to enter machine language programs, and almost guarantees that you'll have a working copy of the Editor the first time you type it in. Once you've typed in, saved, and then loaded MLX, it will ask you for two numbers, or addresses. You should respond with:

Starting address: 49152
Ending address: 51557

You don't have to type in Program 1 all in one sitting. Read Appendix D for details on how to save and later return to a partially completed machine language program.

Save Program 1 to tape or disk. Turn your computer off, then on again, to reset it.

Now type in Program 2, the BASIC part of Hi-Res Graphics Editor. You'll find "The Automatic Proofreader," Appendix C, a great aid in entering any BASIC program, including this one. Make sure you've got a copy of the Proofreader on tape or disk, then type in Program 2. Save it to disk or tape. If you're using a Datassette, it's important that Program 2 is saved on the same tape as Program 1; it should immediately follow the machine language portion. If you have a disk drive, just make sure both programs are on the same disk.

To run the Editor, first load Program 1 with this format:

LOAD"*filename*",8,1 (for disk)
LOAD"*filename*",1,1 (for tape)

Now enter this line and press RETURN:

POKE 642,128: POKE 44,128: POKE 32768,0: NEW

This moves BASIC to a safe place in memory—leaving plenty of room for hi-res screens. *You must type this line each time before you load Program 2.*

Next, load the BASIC program—Program 2. Type RUN, press RETURN, and you are in the Editor.

Autoload

If you want to eliminate some of the steps in loading and running the Editor, you can use this short program to automatically load the two parts of the Editor.

```
10 IF FL=0 THEN FL=1:LOAD"HIRES/ML",8,1
20 PRINT"{CLR}{2 DOWN}POKE642,128:POKE44,128:POKE3
   2768,0:NEW"
30 PRINT"{3 DOWN}LOAD"CHR$(34)"HIRES/BAS"CHR$(34)"
   ,8"
40 PRINT"{HOME}";
50 POKE 198,6:POKE 631,13:POKE 632,13:POKE 633,13
60 POKE 634,82:POKE 635,213:POKE 636,13
```

The program assumes you have used the filenames HIRES/ML for the machine language portion and HIRES/BAS for the BASIC part. Change these names in lines 10 and 30 above to match the names you used. To use the program with tape, change the 8 to a 1 in lines 10 and 30.

All you have to do is load and run this short routine, and the rest is done for you. If you are using tape, save this routine *before* you save Programs 1 and 2.

Set the Joystick Speed

The first prompt in Hi-Res Graphics Editor is for joystick speed. Enter a number from 1 to 10 (10 is fastest). The lower the number, the more control you have over drawing. You can experiment with these numbers to find the best speed for your purposes.

Next, the screen clears and a rectangle appears in the center. This is the sprite cursor. Press the letter *D* and the box will change into an arrow. You are now in Draw Mode. With a joystick in port 2, you can move this arrow around the screen. (A trackball will also work with the Editor. In fact, it seems to give you even finer drawing and movement control.)

Pressing the fire button draws on the screen. If what you have drawn is invisible, press *B* to change the background color and *F* to change the foreground color. Keep pressing these keys to step through the sequence of all possible colors.

Erasing with the Arrow

If you wish to erase what you've drawn, engage the SHIFT LOCK key on the keyboard. Then hold down the fire button and use the joystick to point the arrow at any pixel you want to erase. To start over with a clean slate, just press the f1 key. This clears the screen.

Sprite Mode can be accessed by pressing the *A* (Add), *S* (Stamp), *C* (Copy), or *E* (Erase) key. Let's explore the most interesting of these, hitting the letter C.

Using the joystick, move the rectangle around the screen until it's superimposed on part of your original drawing. (If you've cleared the screen, you can return to Draw Mode by pressing D). Press the fire button, and the contents of the screen under the sprite will be copied onto the sprite.

You can enter Add Mode at any time by pressing A. (In fact, you're automatically in Add Mode as soon as you copy onto a sprite.) In this mode, you can move your sprite around the screen and plant the image anywhere you like. (You *add* the image of the sprite to the images already on the screen.) If you hold the button down while you have the sprite, the sprite's image becomes a wide brush, which you can use for calligraphy and to create other interesting effects.

A Graphic Stamp

Stamp Mode replaces the contents of the screen with the contents of the sprite. If you put the rectangle over a filled-in area, for example, and your sprite is mostly empty, it will erase much of what's beneath the sprite.

If you make a mistake in your drawing, use E, Erase Mode. This mode transforms the sprite cursor into a giant eraser which clears any pixels it passes over.

A Sprite Editor

You can create your own sprites by enlarging the sprite to full-screen proportions. Hold down the f7 key briefly. The screen will clear and an enlarged image of the sprite will appear in the upper left corner of the screen. To edit this sprite, press the fire button of the joystick as you move the cursor in this area. Erasing is simple. Just engage the SHIFT/LOCK key, and instead of drawing to the image, you will erase parts of the sprite. The f1 key clears the sprite, just as it cleared the screen in hi-res mode.

If you want to save or load a hi-res screen, you must do it from this sprite definition mode. (It doesn't save the sprite shape, only the hi-res screen you've created.) Hold the CTRL key while you press *L* for LOAD, and a series of prompts will then appear for loading from disk or tape. Likewise, holding CTRL and S allows you to save to disk or tape.

Anytime you wish to return to hi-res mode, simply hold f7 down for a moment. You can then use the sprite definition you have just created to produce intricate pictures on the hi-res screen.

Two Graphics Screens

The Editor contains a feature which allows you to have two full screens of graphics in memory at one time. Press *T* to toggle between them. When you first try this function, the screen will fill with garbage if nothing has been created on the alternate screen. (There is undefined data in this area.)

Clear the screen (using the f1 key) to start with a new palette. Draw a new design on this screen, and press T to return to the old screen. Pressing T again takes you back to your second creation, and so on.

Printing Your Creation

Since an image created on a computer screen will last only as long as the power is on, a hi-res screen dump is included. Just press the letter *P*, and your 1525 printer (or 1525-compatible printer) will print the contents (minus the sprite cursors) of the screen. Unfortunately, the new Commodore 1526 printer does not have the dot-addressable feature of the 1525 printer, so you won't be able to use this screen dump option if you have the 1526.

Here's a summary of the commands in the Hi-Res Graphics Editor:

Key	**Feature**
D	Draw Mode
SHIFT/ LOCK on	Erase draw (in sprite definition mode, erase parts of sprite)
A	Add Mode; overlay sprite with screen
C	Copy screen to sprite
S	Stamp Mode; replace what is onscreen with sprite image
E	Erase under sprite
F	Sequence through foreground colors
B	Sequence through background colors
T	Toggle between screens
f1	Clear screen (hi-res and sprite definition modes)
f7	Changes from hi-res to sprite definition and vice versa
CTRL-L	Load screen from disk or tape; available only from sprite definition mode
CTRL-S	Save screen from disk or tape, available only from sprite definition mode
P	Produce printout on Commodore 1525 printer

Program 1. Machine Language for Hi-Res Graphics Editor

For easy entry of this machine language program, be sure to use "The Machine Language Editor: MLX," Appendix D.

```
49152 :032,107,198,169,015,141,150
49158 :226,206,032,013,198,169,082
49164 :128,133,044,141,130,002,078
49170 :169,000,141,000,128,169,113
49176 :200,141,000,208,141,254,200
49182 :206,169,003,141,021,208,010
49188 :169,033,141,212,205,169,197
49194 :000,141,016,208,141,255,035
49200 :206,169,100,141,001,208,105
49206 :141,003,208,173,024,208,043
49212 :041,240,009,008,141,024,011
49218 :208,173,017,208,009,032,201
```

```
49224 :141,017,208,169,000,141,236
49230 :238,002,032,182,200,032,252
49236 :107,192,032,004,194,032,133
49242 :186,197,032,239,197,032,205
49248 :186,199,032,008,201,173,127
49254 :238,002,240,230,096,169,053
49260 :032,141,248,007,169,001,194
49266 :141,039,208,238,040,208,220
49272 :173,227,205,201,003,208,113
49278 :018,169,076,141,198,205,165
49284 :169,248,141,197,205,169,237
49290 :014,141,241,002,076,160,004
49296 :192,169,063,141,198,205,088
49302 :169,228,141,197,205,169,235
49308 :025,141,241,002,173,212,182
49314 :205,141,249,007,173,000,169
49320 :220,041,015,141,253,206,020
49326 :056,169,015,237,253,206,086
49332 :141,252,206,160,000,200,115
49338 :204,252,206,208,250,152,178
49344 :010,168,185,204,192,072,255
49350 :185,203,192,072,096,002,180
49356 :194,214,193,218,193,002,194
49362 :194,226,193,230,193,237,203
49368 :193,002,194,222,193,251,247
49374 :193,244,193,002,194,169,193
49380 :050,205,001,208,176,012,112
49386 :173,001,208,056,173,001,078
49392 :208,233,001,141,001,208,008
49398 :096,173,197,205,205,001,099
49404 :208,144,012,173,001,208,230
49410 :024,173,001,208,105,001,002
49416 :141,001,208,096,056,173,171
49422 :254,206,237,198,205,141,231
49428 :253,206,173,255,206,233,066
49434 :001,013,253,206,144,014,145
49440 :173,198,205,141,254,206,185
49446 :169,001,141,255,206,076,118
49452 :063,193,024,173,254,206,189
49458 :105,001,141,254,206,173,162
49464 :255,206,105,000,141,255,250
49470 :206,056,173,254,206,233,166
49476 :000,141,253,206,173,255,072
49482 :206,233,001,013,253,206,218
49488 :144,015,173,016,208,009,133
49494 :001,141,016,208,173,254,111
49500 :206,141,000,208,096,173,148
49506 :016,208,041,254,141,016,006
49512 :208,173,254,206,141,000,062
49518 :208,096,056,173,254,206,079
```

```
49524 :237,241,002,141,253,206,172
49530 :173,255,206,233,000,013,234
49536 :253,206,176,017,056,173,241
49542 :241,002,233,001,141,254,238
49548 :206,169,000,141,255,206,093
49554 :076,166,193,056,173,254,040
49560 :206,233,001,141,254,206,169
49566 :173,255,206,233,000,141,142
49572 :255,206,056,173,254,206,034
49578 :233,000,141,253,206,173,152
49584 :255,206,233,001,013,253,113
49590 :206,144,015,173,016,208,176
49596 :009,001,141,016,208,173,224
49602 :254,206,141,000,208,096,075
49608 :173,016,208,041,254,141,009
49614 :016,208,173,254,206,141,180
49620 :000,208,096,032,227,192,199
49626 :096,032,247,192,096,032,145
49632 :012,193,096,032,112,193,094
49638 :096,032,227,192,032,112,153
49644 :193,096,032,247,192,032,004
49650 :112,193,096,032,247,192,090
49656 :032,012,193,096,032,227,072
49662 :192,032,012,193,096,096,107
49668 :173,001,208,141,003,208,226
49674 :173,000,208,141,002,208,230
49680 :173,016,208,041,001,240,183
49686 :011,169,002,013,016,208,185
49692 :141,016,208,076,042,194,193
49698 :169,253,045,016,208,141,098
49704 :016,208,056,173,254,206,185
49710 :233,024,141,250,206,173,049
49716 :255,206,233,000,141,251,114
49722 :206,165,197,201,013,240,056
49728 :023,201,010,240,030,201,001
49734 :014,240,046,201,018,240,061
49740 :053,201,020,240,079,201,102
49746 :003,240,025,076,168,194,020
49752 :169,000,141,227,205,032,094
49758 :138,194,076,168,194,169,009
49764 :001,141,227,205,032,138,076
49770 :194,076,168,194,032,138,140
49776 :194,076,180,199,076,168,237
49782 :194,169,002,141,227,205,032
49788 :032,138,194,076,168,194,158
49794 :169,003,141,227,205,076,183
49800 :168,194,169,172,141,000,212
49806 :208,141,254,206,169,000,096
49812 :141,016,208,141,255,206,091
49818 :169,124,141,001,208,096,125
```

```
49824 :169,004,141,227,205,032,170
49830 :138,194,173,227,205,201,024
49836 :003,208,016,169,034,141,231
49842 :212,205,173,021,208,041,014
49848 :254,141,021,208,076,204,064
49854 :194,169,033,141,212,205,120
49860 :173,021,208,009,003,141,239
49866 :021,208,056,173,001,208,101
49872 :233,050,141,248,206,173,235
49878 :000,220,041,016,208,017,204
49884 :169,000,141,224,206,162,098
49890 :000,173,227,205,201,004,012
49896 :208,006,076,243,194,076,011
49902 :018,196,076,125,195,173,253
49908 :250,206,141,218,205,173,157
49914 :251,206,141,219,205,169,161
49920 :128,141,216,205,169,000,091
49926 :168,170,141,214,205,142,022
49932 :222,205,140,221,205,032,013
49938 :022,196,174,222,205,172,241
49944 :221,205,173,224,205,045,073
49950 :206,207,240,012,173,216,060
49956 :205,025,000,008,153,000,171
49962 :008,076,057,195,173,216,255
49968 :205,073,255,057,000,008,134
49974 :153,000,008,078,216,205,202
49980 :208,006,169,128,141,216,160
49986 :205,200,024,173,250,206,100
49992 :105,001,141,250,206,173,180
49998 :251,206,105,000,141,251,008
50004 :206,232,224,024,208,177,131
50010 :162,000,173,218,205,141,221
50016 :250,206,173,219,205,141,010
50022 :251,206,238,248,206,162,133
50028 :000,238,214,205,173,214,128
50034 :205,201,021,144,148,169,234
50040 :001,141,227,205,096,169,191
50046 :128,141,226,206,172,224,199
50052 :206,185,000,008,045,226,034
50058 :206,240,008,169,001,141,135
50064 :228,206,076,157,195,169,151
50070 :000,141,228,206,076,157,190
50076 :195,173,227,205,201,003,136
50082 :208,039,173,141,002,208,165
50088 :008,169,001,141,228,206,153
50094 :076,182,195,169,000,141,169
50100 :228,206,024,173,250,206,243
50106 :105,011,141,250,206,173,048
50112 :251,206,105,000,141,251,122
50118 :206,032,022,196,096,142,124
```

```
50124 :216,206,032,022,196,174,026
50130 :216,206,024,173,250,206,005
50136 :105,001,141,250,206,173,068
50142 :251,206,105,000,141,251,152
50148 :206,110,226,206,208,152,056
50154 :238,224,206,232,224,003,081
50160 :240,003,076,125,195,162,017
50166 :000,238,248,206,056,173,143
50172 :250,206,233,024,141,250,076
50178 :206,173,251,206,233,000,047
50184 :141,251,206,172,224,206,184
50190 :192,063,144,001,096,076,074
50196 :125,195,173,250,206,141,086
50202 :250,207,173,251,206,141,230
50208 :251,207,173,248,206,141,234
50214 :248,207,169,000,141,249,028
50220 :207,173,250,207,141,212,210
50226 :207,173,251,207,141,213,218
50232 :207,173,248,207,141,214,222
50238 :207,173,249,207,141,215,230
50244 :207,173,215,207,074,141,061
50250 :217,207,173,214,207,106,174
50256 :141,216,207,173,217,207,217
50262 :074,141,217,207,173,216,090
50268 :207,106,141,216,207,173,118
50274 :217,207,074,141,217,207,137
50280 :173,216,207,106,141,216,139
50286 :207,173,213,207,074,141,101
50292 :219,207,173,212,207,106,216
50298 :141,218,207,173,219,207,007
50304 :074,141,219,207,173,218,136
50310 :207,106,141,218,207,173,162
50316 :219,207,074,141,219,207,183
50322 :173,218,207,106,141,218,185
50328 :207,173,214,207,041,007,233
50334 :141,220,207,173,216,207,042
50340 :010,046,217,207,010,046,188
50346 :217,207,010,141,210,207,138
50352 :046,217,207,173,217,207,219
50358 :141,211,207,173,210,207,051
50364 :010,046,217,207,010,046,212
50370 :217,207,109,210,207,141,005
50376 :216,207,173,211,207,109,043
50382 :217,207,141,217,207,173,088
50388 :216,207,010,046,217,207,091
50394 :010,046,217,207,010,046,242
50400 :217,207,141,216,207,173,105
50406 :218,207,010,046,219,207,113
50412 :010,046,219,207,010,046,006
50418 :219,207,141,218,207,024,234
```

```
50424 :173,216,207,109,218,207,098
50430 :141,208,207,173,217,207,127
50436 :109,219,207,141,209,207,072
50442 :024,173,220,207,109,208,183
50448 :207,141,208,207,169,000,180
50454 :109,209,207,141,209,207,080
50460 :024,169,032,109,209,207,010
50466 :141,209,207,173,208,207,155
50472 :133,251,173,209,207,133,122
50478 :252,173,212,207,041,007,170
50484 :141,225,207,056,169,007,089
50490 :237,225,207,141,225,207,020
50496 :169,000,141,206,207,056,075
50502 :173,225,207,046,206,207,110
50508 :206,225,207,016,245,160,111
50514 :000,173,227,205,201,005,125
50520 :240,090,201,002,240,064,157
50526 :201,004,208,003,076,180,254
50532 :197,173,228,206,240,010,130
50538 :177,251,013,206,207,145,081
50544 :251,076,180,197,173,227,192
50550 :205,201,001,240,018,173,188
50556 :206,207,073,255,141,206,188
50562 :207,177,251,045,206,207,199
50568 :145,251,076,180,197,177,138
50574 :251,045,206,207,240,032,099
50580 :177,251,013,206,207,145,123
50586 :251,076,180,197,177,251,006
50592 :045,206,207,240,015,173,022
50598 :206,207,073,255,141,206,230
50604 :207,177,251,045,206,207,241
50610 :145,251,177,251,141,224,087
50616 :205,096,165,197,201,004,028
50622 :208,046,169,000,133,170,148
50628 :169,032,133,171,160,000,093
50634 :152,145,170,056,165,170,036
50640 :233,255,141,212,206,165,140
50646 :171,233,063,013,212,206,088
50652 :240,016,024,165,170,105,172
50658 :001,133,170,165,171,105,203
50664 :000,133,171,076,200,197,241
50670 :096,165,197,170,201,028,071
50676 :208,008,169,015,141,212,229
50682 :206,076,010,198,201,021,194
50688 :208,104,169,240,141,212,050
50694 :206,076,034,198,238,214,204
50700 :206,173,214,206,045,212,044
50706 :206,201,015,208,035,173,088
50712 :214,206,041,240,141,214,056
50718 :206,076,058,198,024,173,253
```

```
50724 :214,206,105,016,141,214,164
50730 :206,045,212,206,201,240,128
50736 :208,008,173,214,206,041,130
50742 :015,141,214,206,169,000,031
50748 :133,170,169,004,133,171,072
50754 :173,214,206,160,000,145,196
50760 :170,056,165,170,233,231,073
50766 :141,212,206,165,171,233,182
50772 :007,013,212,206,176,016,202
50778 :024,165,170,105,001,133,176
50784 :170,165,171,105,000,133,072
50790 :171,076,066,198,096,160,101
50796 :128,185,119,198,153,064,187
50802 :008,136,016,247,096,255,104
50808 :255,255,192,000,003,192,249
50814 :000,003,192,000,003,192,004
50820 :000,003,192,000,003,192,010
50826 :000,003,192,000,003,192,016
50832 :000,003,192,000,003,192,022
50838 :000,003,192,000,003,192,028
50844 :000,003,192,000,003,192,034
50850 :000,003,192,000,003,192,040
50856 :000,003,192,000,003,192,046
50862 :000,003,192,000,003,255,115
50868 :255,255,000,000,048,000,226
50874 :000,060,000,000,063,000,053
50880 :000,062,000,000,055,000,053
50886 :000,003,128,000,001,192,010
50892 :000,000,224,000,000,000,172
50898 :000,000,000,000,000,000,210
50904 :000,000,000,000,000,000,216
50910 :000,000,000,000,000,000,222
50916 :000,000,000,000,000,000,228
50922 :000,000,000,000,000,000,234
50928 :000,000,000,000,000,000,240
50934 :000,000,000,000,000,000,246
50940 :000,169,012,141,033,208,047
50946 :169,147,032,210,255,169,216
50952 :021,141,024,208,169,027,086
50958 :141,017,208,169,000,141,178
50964 :208,205,133,180,141,207,070
50970 :205,141,206,205,133,195,087
50976 :169,216,133,196,169,004,151
50982 :133,181,162,000,160,000,162
50988 :169,128,141,210,205,140,013
50994 :206,205,172,207,205,185,206
51000 :000,008,140,207,205,172,020
51006 :206,205,045,210,205,240,149
```

```
51Ø12 :Ø11,169,ØØ1,145,195,169,246
51Ø18 :16Ø,145,18Ø,Ø76,Ø88,199,154
51Ø24 :169,ØØØ,145,195,169,16Ø,15Ø
51Ø3Ø :145,18Ø,Ø24,165,195,1Ø5,132
51Ø36 :ØØ1,133,195,165,196,1Ø5,119
51Ø42 :ØØØ,133,196,Ø24,165,18Ø,Ø28
51Ø48 :1Ø5,ØØ1,133,18Ø,165,181,1Ø1
51Ø54 :1Ø5,ØØØ,133,181,Ø78,21Ø,Ø49
51Ø6Ø :2Ø5,173,21Ø,2Ø5,24Ø,ØØ3,128
51Ø66 :Ø76,Ø49,199,238,2Ø7,2Ø5,Ø72
51Ø72 :169,128,141,21Ø,2Ø5,232,189
51Ø78 :224,ØØ3,144,167,Ø24,165,Ø93
51Ø84 :18Ø,1Ø5,Ø16,133,18Ø,165,151
51Ø9Ø :181,1Ø5,ØØØ,133,181,Ø24,ØØ2
51Ø96 :165,195,1Ø5,Ø16,133,195,193
511Ø2 :165,196,1Ø5,ØØØ,133,196,185
511Ø8 :162,ØØØ,238,2Ø8,2Ø5,173,126
51114 :2Ø8,2Ø5,2Ø1,Ø21,176,ØØ3,216
5112Ø :Ø76,Ø49,199,Ø96,169,ØØ1,254
51126 :141,238,ØØ2,Ø96,165,197,253
51132 :2Ø1,Ø41,24Ø,ØØ1,Ø96,169,168
51138 :ØØØ,Ø32,189,255,169,ØØ4,Ø75
51144 :17Ø,16Ø,255,Ø32,186,255,234
5115Ø :Ø32,192,255,162,ØØ4,Ø32,115
51156 :2Ø1,255,176,ØØ3,Ø76,22Ø,119
51162 :199,Ø96,169,ØØ8,Ø32,21Ø,164
51168 :255,169,Ø13,Ø32,21Ø,255,134
51174 :162,ØØØ,169,ØØ1,141,2Ø4,139
5118Ø :2Ø5,169,ØØØ,141,25Ø,2Ø6,183
51186 :169,ØØØ,141,251,2Ø6,169,154
51192 :199,141,248,2Ø6,169,ØØ5,192
51198 :141,227,2Ø5,142,242,ØØ2,189
512Ø4 :Ø32,Ø22,196,174,242,ØØ2,16Ø
5121Ø :173,224,2Ø5,Ø45,2Ø6,2Ø7,Ø46
51216 :24Ø,Ø12,173,2Ø2,2Ø5,Ø13,Ø93
51222 :2Ø4,2Ø5,141,2Ø2,2Ø5,Ø76,Ø31
51228 :Ø41,2ØØ,173,2Ø4,2Ø5,Ø73,156
51234 :255,Ø45,2Ø2,2Ø5,141,2Ø2,Ø6Ø
5124Ø :2Ø5,Ø14,2Ø4,2Ø5,173,2Ø4,Ø21
51246 :2Ø5,2Ø1,128,24Ø,Ø2Ø,Ø24,Ø96
51252 :173,25Ø,2Ø6,1Ø5,ØØ1,141,16Ø
51258 :25Ø,2Ø6,173,251,2Ø6,1Ø5,225
51264 :ØØØ,141,251,2Ø6,Ø76,ØØ1,227
5127Ø :2ØØ,173,2Ø2,2Ø5,ØØ9,128,219
51276 :224,Ø45,144,Ø1Ø,173,2Ø2,1Ø6
51282 :2Ø5,Ø41,Ø31,ØØ9,128,141,125
51288 :2Ø2,2Ø5,168,Ø32,21Ø,255,136
```

```
51294 :152,032,210,255,169,001,145
51300 :141,204,205,169,000,141,192
51306 :202,205,056,173,250,206,174
51312 :233,006,141,250,206,173,097
51318 :251,206,233,000,141,251,176
51324 :206,206,248,206,173,248,131
51330 :206,201,255,240,003,076,087
51336 :001,200,224,045,176,031,045
51342 :024,173,250,206,105,007,139
51348 :141,250,206,173,251,206,095
51354 :105,000,141,251,206,232,065
51360 :169,199,141,248,206,169,012
51366 :013,032,210,255,076,001,241
51372 :200,169,013,032,210,255,027
51378 :032,231,255,096,174,240,182
51384 :002,160,255,136,208,253,174
51390 :202,208,248,096,173,167,004
51396 :002,174,168,002,160,001,191
51402 :032,186,255,173,169,002,251
51408 :162,172,160,002,032,189,157
51414 :255,169,000,162,000,160,192
51420 :032,032,213,255,096,173,253
51426 :167,002,174,168,002,160,131
51432 :001,032,186,255,173,169,024
51438 :002,162,172,160,002,032,000
51444 :189,255,169,032,133,254,252
51450 :169,000,133,253,169,253,203
51456 :162,255,160,063,032,216,120
51462 :255,096,165,197,201,022,174
51468 :240,001,096,169,000,133,139
51474 :170,169,032,133,171,169,094
51480 :000,133,180,169,096,133,223
51486 :181,160,000,177,170,141,091
51492 :062,003,177,180,141,064,151
51498 :003,173,062,003,145,180,096
51504 :173,064,003,145,170,024,115
51510 :165,170,105,001,133,170,030
51516 :165,171,105,000,133,171,037
51522 :024,165,180,105,001,133,162
51528 :180,165,181,105,000,133,068
51534 :181,056,165,170,233,255,114
51540 :141,200,205,165,171,233,175
51546 :063,013,200,205,144,193,140
51552 :096,013,013,013,013,013,001
```

Program 2. BASIC Portion of Hi-Res Graphics Editor

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C.

```
5 INPUT "{CLR}JOYSTICK SPEED (1-10)";JS$      :rem 137
6 IF VAL(JS$)<1OR VAL(JS$)>10 THEN5           :rem 192
7 POKE752,11-VAL(JS$)                         :rem 180
8 FOR T= 2048TO2048+64:POKET,0:NEXT           :rem 22
10 SYS50624                                   :rem 97
11 SYS49152                                   :rem 102
12 GETA$:IF PEEK(197)<>3THEN12                :rem 199
13 FOR T= 1 TO 300:NEXT                       :rem 188
15 SYS50941                                   :rem 104
16 VI=53248:POKEVI+21,1:POKEVI,21:POKEVI+16,PEEK(V
   I+16)OR1:POKEVI+1,100                      :rem 51
17 POKE2040,32                                :rem 238
20 SC= 1024:PX=0:PY=0:CN=0:OS=55296:OC=PEEK(OS)
                                              :rem 24
30 GET A$:IF A$=""THEN CN=CN+1                :rem 65
31 IF PEEK(197)=4 THEN FOR T=2048TO2048+64:POKET,0
   :NEXT:SYS50941                             :rem 196
32 IF PEEK(197)=3THENPOKE198,0:FORT=1TO300:NEXT:GO
   TO11                                       :rem 62
33 IF A$="{L}"THEN GOSUB 300:SYS51394:GOSUB400:SYS
   50941                                      :rem 242
34 IF A$="{HOME}"THEN GOSUB300:SYS51425:GOSUB400:S
   YS50941                                    :rem 245
40 IF CN= 2 THEN POKE SC,PEEK(SC)OR128:CN=0
                                              :rem 147
50 IF CN= 1 THEN POKE SC,PEEK(SC)AND127       :rem 140
60 IF(PEEK(56320)AND16)<>0 THEN 65            :rem 58
61 IF PEEK(653)THEN POKESC+54272,0:SH=1:GOSUB200:G
   OTO 65                                     :rem 246
63 POKESC+54272,1:SH=0:GOSUB 200              :rem 72
65 IF 15-PEEK(56320)=0 THEN 79                :rem 15
66 FL=0:OC=PEEK(SC+54272):OS=SC+54272         :rem 141
70 ON 15-PEEK(56320)AND15GOSUB 80,90,95,100,120,13
   0,140,150,160,170                          :rem 163
72 POKESC,(PEEK(SC)OR128)                     :rem 243
75 SC=1024+40*Y+X                             :rem 155
79 GOTO 30                                    :rem 12
80 Y =Y+(Y>0):RETURN                          :rem 180
90 Y=Y-(Y<20):RETURN                          :rem 231
95 RETURN                                     :rem 78
100 X=X+(X>0):RETURN                          :rem 218
110 RETURN                                    :rem 114
120 Y=Y+(Y>0):X=X+(X>0):RETURN                :rem 72
130 Y=Y-(Y<20):X=X+(X>0):RETURN               :rem 123
140 RETURN                                    :rem 117
150 X=X-(X<23):RETURN                         :rem 20
160 Y=Y+(Y>-0):X=X-(X<23):RETURN              :rem 174
```

```
170 Y=Y-(Y<20):X=X-(X<23):RETURN                  :rem 180
200 BO=Y*3+INT(X/8)                                :rem 60
210 BT= 2↑(7-(X-INT(X/8)*8)):P=64*PEEK(2040)+BO
                                                   :rem 49
220 IF SH=0 THENPOKEP,PEEK(P)ORBT:GOTO230         :rem 10
225 POKEP,PEEK(P)AND(255-BT):SH=0                 :rem 207
230 RETURN                                        :rem 117
300 PRINT"{BLK}{7 RIGHT}{CLR}{RVS}D{OFF}ISK OR
    {RVS}T{OFF}APE"                               :rem 144
301 GET J$:IF J$=""THEN301                         :rem 93
302 IF J$<>"D"AND J$<>"T"THEN 301                 :rem 170
303 INPUT "FILENAME";FI$                          :rem 153
305 IF LEFT$(J$,1)="D"THEN D=8:GOTO310             :rem 70
306 D=1                                            :rem 75
310 FOR T= 684 TO 684+LEN(FI$)-1:POKET,ASC(MID$(FI
    $,T-683,1)):NEXT                              :rem 150
320 POKE679,D:POKE680,D:POKE681,LEN(FI$):POKE682,1
    72:POKE683,2                                  :rem 159
325 RETURN                                        :rem 122
400 OPEN15,8,15:INPUT#15,A$,B$,C$,D$:PRINTA$;" ";B
    $" ";C$;" ";C$;" ";D$                          :rem 52
405 CLOSE15                                       :rem 117
410 FOR T= 1TO 3000 :NEXT :RETURN                  :rem 55
```

HiSprite

Michael J. Blyth

"HiSprite" is a machine language utility which gives you fast, easy control over Commodore 64's sprites from BASIC, including collision monitoring, joystick control, boundaries, and a high-resolution "pen."

If you've ever tried to write a fast-action game program or a complex graphic display using BASIC, Commodore 64 sprites, and high-resolution graphics, you've probably been frustrated by the slow speed of the program. BASIC is simply too slow when it comes to calculating new horizontal and vertical velocities and positions for multiple sprites, reading joysticks, monitoring collisions, and doing all the necessary PEEKing and POKEing for sprites and high-resolution (hi-res) graphics.

"HiSprite" is a powerful machine language utility which handles all these low-level tasks quickly, freeing you to use BASIC for high-level control. HiSprite allows fast, complex, and smooth control of all eight sprites for either BASIC or machine language programs. Variables define horizontal and vertical position, velocity, acceleration, and boundaries for each sprite. Other variables determine joystick control, hi-res plotting, and what action to take:

- at boundaries (stop, disappear, bounce, or wrap around)
- on collision with background (stop, disappear, bounce, or continue)
- on collision with another sprite (stop, disappear, bounce, "stick")

Finally, HiSprite can be used either as a subroutine (with SYS in BASIC or JSR in machine language) or in a continuous, interrupt-driven mode.

Entering HiSprite

First, you'll need to type in HiSprite, found at the end of this article. The list of numbers in Program 2 is machine language. Only with machine language can you get the speed and power necessary to move sprites easily about the screen. However, machine language programs aren't as easy to type in as BASIC. To help you with all this typing, you'll find MLX (Appendix D) an invaluable tool. Be sure to read Appendix D before you start entering HiSprite.

Type in and save the MLX program. When you're ready to enter HiSprite, turn your computer off, then on again (this clears it out). Load MLX from tape or disk and type RUN. MLX asks you for the starting and ending addresses. The addresses are:

Starting address: 49152
Ending address: 50705

Simply follow the directions in Appendix D to enter the program. You don't have to enter it at one sitting, but can save your work, typing HiSprite in several sessions.

How It Works

After you've entered HiSprite with MLX, you can load it with the command LOAD "HISPRITE",8,1 for disk or LOAD "HISPRITE",1,1 for tape. Type SYS 49152, then NEW and CLR. If you're using the demonstration program, Program 1, you only need to type it in (or if you've already entered it, type LOAD "HISPRITE.DEMO",8 for disk, LOAD "HISPRITE" for tape). If you're using HiSprite with a program of your own, you'll first have to SYS 49152, or make sure that statement is included in your program.

Although HiSprite is a complex program, with many variables and functions, it's easy to use once you've seen this step-by-step demonstration of its abilities.

In HiSprite, integer arrays hold the information needed for controlling each sprite. The horizontal and vertical (X and Y) directions are controlled independently. The variables and their functions are:

SP%(*i*,0 or 1) Position of upper left corner of sprite *i* (where *i* is from 0 to 7). SP%(*i*,0)=X position; SP%(*i*,1)=Y position. Any valid integer from −32767 is OK, but the screen shows the area from 24 to 344 horizontally and from 50 to 250 vertically. To position sprite 1 in the upper left corner of the screen, you could use: SP%(1,0)=24:SP%(1,1)=50.

SV%(*i*,0 or 1) X or Y velocity, again for sprite *i*. Each time HiSprite is called, 1/256 of this value is added to the current position. SV%(0,1)=128 thus means that on every other call, sprite 0 will move down one dot or pixel.

SA%(*i*,0 or 1) X or Y acceleration. Each time HiSprite is called, this value is added to the corresponding velocity. SA%(3,0)=10 means that SV%(3,0) (X velocity for sprite 3) will be automatically increased by 10 on each call.

SL%(*i*,0 or 1) Upper limits for X and Y position. If HiSprite detects that sprite *i* would move beyond its limits, it takes appropriate action (see below).

SL%(*i*,2 or 3) Lower limits for X and Y position, respectively.

SC%(*i*,0 or 1) Options such as joystick control and out-of-bounds action (details below).

SC%(*i*,2–7) Options for action to take when sprite *i* collides with another sprite.

Now we can get started. If you don't still have HiSprite loaded into your 64, type:

LOAD "HISPRITE",8,1 (LOAD "HISPRITE",1,1 for tape).

Then type NEW and CLR.

Seeing Sprites Move

To begin with, enter and save Program 1, "HiSprite Demo." Of course, you can leave out the REM statements.

Program 1. HiSprite Demo

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C.

```
5 SD=0:SE%=0:SF%=0:SG%=0:SH%=0:SI%=0:SJ%=0:SK%=0:S
  L%=0:HS=49274                                  :rem 145
10 DIM SP%(7,1),SV%(7,1),SA%(7,1),SL%(7,3),SC%(7,9
   ),SR%(7)                                      :rem 186
20 SE=0:SYS12*4096 :REM INITIALIZE                :rem 16
30 V=53248: POKE V+21,255 :REM ENABLE ALL 8 SPRITE
   S                                              :rem 17
40 FORI=0TO7: REM FOR EACH SPRITE                :rem 200
50 POKE 2040+I,13 :REM MEMORY LOCATION           :rem 174
60 POKE V+39+I,I :REM SPRITE COLOR               :rem 223
65 SP%(I,0)=60: SP%(I,1)=60: REM POSITION :rem 251
70 SV%(I,0)=50*I+5: SV%(I,1)=50*I+50: REM VELOCITY
                                                 :rem 209
80 SL%(I,0)=320: SL%(I,1)=230: REM UPPER LIMITS
                                                  :rem 55
85 SL%(I,2)=24: SL%(I,3)=50: REM LOWER LIMITS
                                                 :rem 222
90 SC%(I,0)=1:SC%(I,1)=1: REM OPTIONS             :rem 44
100 NEXT                                         :rem 208
```

```
200 FORI=832TO896: POKEI,255:NEXT :REM CREATE SPRI
    TE SHAPE                                 :rem 243
210 FORI=844TO885: POKEI,28: NEXT: FORI=859TO867:
    {SPACE}POKEI,0:NEXT                      :rem 197
220 POKE857,8: POKE869,8                      :rem 63
300 SYS HS                                    :rem 45
400 GOTO300                                   :rem 96
```

Program Notes

- The arrays and variables in lines 5 and 10 must be the first ones used in the program, and must be defined in the order shown.
- Line 50 defines the location (13*64) of the shape information, which is the same for all eight sprites in this program. You could alter the shape information to create your own sprites if you wanted. Line 60 sets a different color for each sprite.
- Lines 200–220 put the shape information where we want it.
- Line 300 is an infinite loop calling the main part of HiSprite.

Velocities, Borders, and Acceleration

Now run the program. You'll see all eight sprites moving downward and rightward, then bouncing against the borders defined in lines 80 and 90 (0 and 320 horizontally; 0 and 230 vertically). The speeds vary according to the definition in line 70. When you've seen enough, press RUN/STOP. Pressing RUN/STOP and RESTORE together erases the sprites. Let's try some changes. First, adjust the Y velocity in line 70 by changing SV%(I,1)=50*I+50 to SV%(I,1)=100*I+150 and rerunning the program. You can adjust the X velocity by changing SV%(I,0) to something similar.

Line 90 determines what happens when the borders are reached. Try using values 0 through 3 for SC%(I,1) and/or SC%(I,0) and see what happens with each one. Then return to using 1.

You can use the joysticks to control (*gate*) either the velocity or the acceleration of sprites. For example, if the vertical velocity is gated, then the sprite will only move vertically when the joystick is moved up or down. When acceleration is gated, the sprite moves continuously but is sped up and slowed down by the joystick. Change line 30 and add the other two lines:

```
30 V=53248: POKE V+21,3
270 SC%(0,0)=33:SC%(0,1)=33
275 SV%(0,0)=300:SV%(0,1)=300
```

Try it. X and Y velocities are gated by the joystick in port 2. Now change both values of 33 to 65 in line 270, add line 280, and rerun:

```
280 SA%(0,0)=10:SA%(0,1)=10
```

Now the accelerations are gated.

Friction and Drawing

There are other options available with the control or option variables SC%(*i*,0) and SC%(*i*,1). Unlike the other variables, these depend on the setting of individual bits (each integer consists of 16 bits numbered 0 to 15). For example, bits 1 and 0 control what happens at borders (01=bounce), while bit 5 controls velocity gating (1=gate). To determine the value which will set the desired bits, start with 0, add 1 to set bit 0, 2 to set bit 1, 4 for bit 2, 8 for bit 3, and so on up to 2048 for bit 11. Thus to bounce at borders and gate velocity, we add 1 (bounce) and 32 (gate velocity) to give 33. Take a look at Table 1 for the control option bits and values to set. Within a program you may want to set/clear given bits of SC% like this:
To set bit *K*: SC%(...)=SC%(...)OR2↑*K*
To clear bit *K*: SC%(...)=SC%(...) AND NOT2↑*K*

Now we can continue experimenting. Reset SC%(0,0) and SC%(0,1) to 1 in line 270, delete 280, and add:

```
290 SV%(0,1)=0:POKE V+21,1
310 SA%(0,1)=120-SP%(0,1)
```

Run the program to see the changes in effect. Line 290 cancels sprite 0's Y velocity and for clarity disables all other sprites; line 310 plots a sine curve by defining sprite 0's Y acceleration in terms of its distance from 120. The sprite acts like a mass on a spring, with the tension proportional to the stretch. We can add *friction* (acceleration opposite velocity) by changing 310 and rerunning:

```
310 SA%(0,1)=120-SP%(0,1)-.01*SV%(0,1)
```

Finally, here's a taste of hi-res graphics. Add the following line and rerun:

```
280 SC%(0,1)=257:SYS50647:SYS50577:SYS506
    15
```

This line does four things: makes sprite 0 start drawing; turns on the hi-res mode; clears the color information for hi-res screen; and clears the hi-res screen itself. To get back to the usual mode from hi-res, you can either use SYS 50679 or press RUN/STOP together with RESTORE to reset the computer.

To get interesting Lissajous patterns of motion, you can make sprite 0 *vibrate* in its X direction as well:

```
295 A%=RND(1)*8+2:B%=RND(1)*8+2
310 SA%(0,1)=(120-SP%(0,1))/A%
320 SA%(0,0)=(120-SP%(0,0))/B%
```

In the next section, we'll continue with hi-res and look at collisions, multicolor hi-res, and interrupt mode, so save what you've done so far.

Table 1. Summary of Control Variables

BASIC Variables			
SP%(*i*,0)	Horizontal position		
(*i*,1)	Vertical position		
SV%(*i*,0)	Horizontal velocity (dots per 256 calls)		
(*i*,1)	Vertical velocity		
SA%(*i*,0)	Horizontal acceleration (changes in velocity per call)		
(*i*,1)	Vertical acceleration		
SL%(*i*,0)	Horizontal upper boundary		
(*i*,1)	Vertical upper boundary		
(*i*,2)	Horizontal lower boundary		
(*i*,3)	Vertical lower boundary		
SC%(*i*,0)	Horizontal control:		
	Bits	Value	Function
	0–1		Action at boundary:
		0	Stop
		1	Bounce (reverse direction)
		2	Wrap (enter at opposite boundary)
		3	Disable (make sprite disappear)
	2–3		Action on collision with background:
		0	No action (ignores collision)
		4	Bounce
		8	Stop
		12	Disable
	4	16	Monitor sprite-sprite collisions
	5	32	Gate velocity
	6	64	Gate acceleration
	7	0	Joystick in port 2
		128	Joystick in port 1

SC%(*i*,1)	Vertical control. Same as SC%(*i*,0) plus: Bits / Value / Function 8 / 256 / Pen-down (for hi-res plotting) 9–10 / / Pen color (multicolor mode only): 0 Screen color 512 Upper 4 bits of screen memory 1024 Lower 4 bits of screen memory 1736 Color nybble (starting at location 55296)
SC%(*i*,2)/(*i*,3)	Horizontal/Vertical: stop on collision with sprite whose corresponding bits are set
(*i*,4)/(*i*,5)	Horizontal/Vertical: stick on sprite collision
(*i*,6)/(*i*,7)	Horizontal/Vertical: bounce on sprite collision
(*i*,8)/(*i*,9)	Horizontal/Vertical: disable on sprite collision
SR%(*i*)	Offset (from beginning of screen) of character lying under sprite *i*
SD	1=move sprites on interrupt 0=Don't move sprites on interrupt
SE%	Latched out of bounds flags (1 bit for each sprite)
SF%	Sprite-sprite collision flags
SG%	Background collision flags
SH%	Latched sprite-sprite collision flags
SI%	Latched background collision flags
SJ%	Release switches, sprite-sprite collision
SK%	Release switches, background collisions
SL%	Number of jiffies (1/60 second) required per call in interrupt mode (1–255)

Subroutine Addresses

Decimal	Hex	Function
49152	C000	Initialization
49274	C07A	Main mover subroutine
50577	C591	Fill main screen with character in 50607 ($C5AF) (color information in normal hi-res mode)
50615	C5B7	Clear hi-res screen
50644	C5D4	Turn on multicolor hi-res mode
50647	C5D7	Turn on standard hi-res mode
50679	C5F7	Turn off hi-res mode

Note: For the sake of speed, HiSprite does not "look up" the location of BASIC variables and arrays, but rather depends on their being defined in a fixed order. Therefore, any names can be attached to them, except for SP%(...) which must be the first array. For example, if the first program line is:

A=0:B%=0:C%=0:D%=0:F%=0:G%=0:H%=0:I%=0

then A will act as SD, B% as SE%, and so on.

Collision Handling

One of the most powerful features of HiSprite is its ability to monitor and flexibly react to sprite to background and sprite to sprite collisions. Let's deal first with the background collisions, that is, a collision between a sprite and anything on the screen besides another sprite. The sprite's X and Y motion is controlled by bits 2–3 of SC%(*i*,0) and SC%(*i*,1), respectively:

Bit 3 Set To	Bit 2 Set To	Total Value	Action
0	0	0	No action (collisions ignored)
0	1	4	Bounce (reverse direction)
1	0	8	Stop
1	1	12	Disable (disappear and stop moving)

Try it out. Put a REM at the beginning of line 280 of the altered version of Program 1 that you saved earlier, delete lines 310 and 320, and change 270 and 290 to:

```
270 SC%(0,0)=9:SC%(0,1)=5
290 SV%(0,1)=200:POKE V+21,1
```

Clear the screen, type a few characters here and there on it, and run the program. When sprite 0 hits a character it should stop its horizontal motion and reverse (bounce) vertically. Now make either SC%(0,0) or SC%(0,1)=13 and rerun. This will cause the sprite to disappear when it hits a character. Try out various combinations with different patterns of characters on the screen. For example, if you make both SC%(0,0) and SC%(0,1)=77 (that is, 1+12+64) and both SA%(0,0) and SA%(0,1)=10, you have a game where you must maneuver your sprite around obstacles. If you miss, your sprite vanishes.

Collisions between sprites are slightly more complicated because we want the flexibility of acting differently on collisions between different sprites. When bit 4 of SC%(*i*,0) or SC%(*i*,1) is set, sprite *i* is monitored for collisions with other sprites. SC%(*i*,2) through SC%(*i*,9) determine what happens when sprite *i* hits another:

Horizontal	Vertical	Action
SC%(*i*,2)	SC%(*i*,3)	Stop
SC%(*i*,4)	SC%(*i*,5)	Stick (stay with other sprite)
SC%(*i*,6)	SC%(*i*,7)	Bounce
SC%(*i*,8)	SC%(*i*,9)	Disable

Individual bits of SC% (*i*,2) through SC%(*i*,9) select the action to take when the corresponding sprite is hit. Setting bits 0 through 7 selects the corresponding sprite against which the action is taken. For example, bit 0 of SC%(3,2) means *stop sprite 3 horizontally on collision with sprite 0,* while bit 3 of SC%(2,5) means *sprite 2 sticks vertically on collision with sprite 3.* If we want sprite 0 to stick horizontally to sprites 1 and 3, and stop vertically when it hits sprites 2 or 4, we use:

SC%(0,0)=17 (monitor collisions; bounce at borders)
SC%(0,1)=17
SC%(0,3)=20 (bits 2 and 4 set: 4+16=20)
SC%(0,4)=10 (bits 1 and 3 set: 2+8=10)

Experiment again with various combinations before continuing. Remember that regardless of the settings of SC%(*i*,2) to (*i*,9), collisions are ignored if bit 4 is not set (value of 16) in SC%(0,0) or SC%(0,1). When more than two sprites collide at once, the results are sometimes not what you would expect. This is because the computer only keeps track of which sprites have collided, not between which sprites collisions have occurred.

You may have already noticed that we need a way to free sprites once they stop or stick. Two variables do this. Setting bit *i* of SJ% or SK% releases sprite *i* from sprite or background collisions, respectively. Once released, the sprite moves freely until it is *unstuck* (free), then its collision monitoring is resumed. Thus if sprite 6 is stuck on background, you can say SK%=SK%OR2↑6 or SK%=SK%OR64 to release it.

There are a few other useful variables for collisions. SF% and SG% contain the current sprite and background collision flags, respectively (bit *i* is set when sprite *i* collides with background or any sprite). SF% and SG% should be used rather than the usual PEEKs (locations 53278 and 53279), since PEEKing the flags clears them. SH% and SI% contain latched sprite and background collision flags; once one of these bits is set by a collision, it remains set until you clear it. This allows you to catch events without having to monitor each one constantly. Finally, SE% contains latched out-of-bounds flags; bit *i* is set when sprite *i* hits one of its boundaries.

High-Resolution

The hi-res features of HiSprite are best understood if you're familiar with the principles of hi-res on the 64 as outlined in the *Programmer's Reference Guide,* pages 100–105 and pages 121–128. Briefly, the hi-res screen is an 8K area of memory where every pixel of the video screen is represented. HiSprite can be used to draw on this screen. For starters, you need to get the computer into the hi-res mode. With HiSprite, SYSing 50647 does this. This places the hi-res screen at location 8192. If you need it somewhere else, as you might if you have a large program, you must set up the hi-res mode yourself.

Try this out by typing SYS 50647 and hitting RETURN. The screen will turn to garbage. Now type SYS 50615 and RETURN to clear the hi-res screen.You won't see what you're typing; if you make a mistake, type SHIFT-RETURN and start over. Why isn't the entire screen the same color? Because the hi-res color information comes from the usual screen memory, which now has miscellaneous text, including the SYS commands you just typed. To clear what is now the hi-res color screen, type SYS 50577 and RETURN. This fills the color area with whatever is in location 50607. There will be a little garbage at the screen bottom. Now return to the usual mode with SYS 50679 and RETURN. You'll see most of the screen filled with the letter *L.*

Drawing on Hi-Res

How do you draw on the screen once you're in hi-res mode? Setting bit eight of SC%(*i*,1) puts sprite *i*'s "pen down," causing a dot to be drawn near the center of the sprite (specifically, the sprite's twelfth column, tenth row). A moving sprite with its pen down draws a curve along its path. There is a limitation, however; the dots are drawn only in positions actually occupied by the sprite, not in any it may have passed over. Thus velocities greater than 256 (one dot per call) will leave discontinuous or dotted curves. Now go back to the original version of Program 1, enter line 280 and rerun:

```
280 SC%(0,1)=257:SYS 50647: SYS 50577: SY
    S 50615
```

Try playing around a little. For instance, you could change line 280 to read FOR A=0 TO 7:SC%(A,1)=257:SYS 50647:SYS 50577:SYS 50615:NEXT, and all eight sprites will use their *pen*. Remember that

SC%(0,1)=<anything>AND NOT256

will pick up the pen (no drawing) while

SC%(0,1)=<anything>OR256

will put it down.

Pen Colors

In standard hi-res mode, the color of an *on* dot is taken from the upper four bits of the corresponding screen memory location, while the color of *off* dots is from the lower four bits. All *on* dots in each character position have the same color scheme. If we want to set the color a sprite is drawing, we need to know what screen memory location to use. If sprite *i*'s pen is down, then SR%(*i*) gives the character position where it is drawing. The position is expressed as an offset from the beginning of the screen. To cause sprite 0 to draw light blue (color 14) on a black background (color 0), put 0+16*14 into the locations under the sprite. For instance:

POKE 1024+SR%(0),224

If you wanted the same pen/background color everywhere, you could use

POKE 50607,224: SYS 50577

to fill the entire color screen.

SR%(*i*) can also be used for other graphics modes as long as the sprite's pen is down:

POKE 1024+SR%(0),0:POKE55296+SR%(0),14

puts a light blue @ (character 0) where sprite 0 is. If you try this with Program 1, a light blue @ character should appear in the top left-hand corner of the screen.

Multicolor Hi-Res

In multicolor hi-res mode, the color of each dot can be set independently. The tradeoff is that each dot is twice as wide, so there is only half as much horizontal resolution. SYS 50644 turns on multicolor hi-res mode; SYS 50679 cancels hi-res and multicolor. While in multicolor mode, bits 9 and 10 of SC%(*i*,1) determine pen color, as follows:

Bit 10	Bit 9	Value	Source of Color Information
0	0	0	Background color
0	1	512	Upper four bits of screen memory
1	0	1024	Lower four bits of screen memory
1	1	1536	Nybble from color memory

As with the standard hi-res mode, SR%(*i*) contains the character location offset. To put a color code, say 3, in the color nybble under a sprite, POKE 55296+SR%(*i*),3.

Interrupt Mode

Ordinarily, HiSprite is active only when it's called, using SYS 49274 (see line 300 in Program 1; HS is set in line 5). In interrupt mode, however, HiSprite is automatically called up once each video frame, or about 60 times per second. As long as the array variable SP%(...) is defined, the sprites will move even when no BASIC program is running. This is most useful for designing and testing programs, as it allows you to manipulate the control variables in direct mode while you watch the results. To try this, take out line 280 again and run Program 1. Stop the program, and enter interrupt mode by typing in direct mode SD=1 and press RETURN. The sprites are moving again. Now change whatever variables you want and watch the results. Interrupt mode is turned off by SD=0. You can put everything into slow motion by making SL% greater than 1. SL% represents the number of video frames required to trigger a call to HiSprite. In direct mode, type SL%=5: The sprites will slow to a crawl.

There are two cautions in interrupt mode. First, since the interrupt can occur at any time, it will (although rarely) occur when BASIC has begun, but not yet finished, changing or reading a variable such as acceleration. This will seldom make any difference, but if it becomes a problem, set SD to 0 to prevent interrupts, do your critical operations, then reset SD to 1. Second, you should avoid I/O and screen editing while in interrupt mode. In fact, I/O may not work correctly while HiSprite is active at all, so you may need to hit RUN/STOP–RESTORE first.

HiSprite may seem complex, but if you experiment with it a bit at a time, you'll see how creative it is and how much you can do simply by manipulating a few variables and adding a little logic. For starters, a sprite that moves and draws is essentially a turtle, right? Sprites can easily push, pull, block, destroy, and bounce each other. Automatically maintained velocities and accelerations make it easy to have sprites act like physical objects such as balls, rockets, or molecules.

Using HiSprite with Machine Language

The only preparation required for using HiSprite from a machine language program is setting up the variables to look like BASIC variables. VARTAB ($2D–2E) and ARYTAB ($2F–$30) should point to the storage areas of the variables and arrays used by HiSprite. Integer array elements are two bytes long, with the high-order byte first, in twos complement form. Arrays are stored with the first subscript varying fastest. Thus, if SC%(0,0) is stored in $2000–2001, the other locations would be:

```
$2002-2003 SC%(1,0)
...
$200E-200F SC%(7,0)
$2010-2011 SC%(1,1)
...
```

Table 2 gives the required offsets from the location pointed to by ARYTAB or VARTAB to the high-order byte of the element shown:

Table 2. Offsets

Pointer	Offset (Decimal)	Variable/Element
VARTAB	2	SD (Interrupt mode is on if any of low-order four bits is 1)
	10	SE%
	17	SF%
	24	SG%
	31	SH%
	38	SI%
	45	SJ%
	52	SK%
	59	SL%

ARYTAB*	9	SP%(0,0)
	50	SV%(0,0)
	91	SA%(0,0)
	132	SL%(0,0)
	164	SL%(0,2)
	205	SC%(0,0)
	372	SR%(0)

*The first two bytes pointed to by ARYTAB must contain $D3D0 (representing "SP%"). Otherwise no header information is required.

Linking HiSprite to BASIC Programs

HiSprite must be loaded before a BASIC program can use it. The most straightforward way to do this is manually, that is, to enter

LOAD "HISPRITE",8,1 (or ...,1,1 for tape)

before running the main program(s). Another possibility is to have the main program load HiSprite each time it runs:

```
10 IF S=0 THEN S=1: LOAD "HISPRITE",8,1
20 CLR
30 . . . REST OF PROGRAM
```

If the main program is going to be run repeatedly, however, it's pointless to load HiSprite each time. So the third approach is to use a loader program to load HiSprite first and then the main program:

```
10 IF S=0 THEN S=1: LOAD"HISPRITE",8,1
20 PRINT"{CLR}{3 DOWN}LOAD"CHR$(34)"MAIN
   {SPACE}PRG"CHR$(34)",8"
30 PRINT"{HOME}":POKE 631,13:POKE 198,1:END
```

Program 2. HiSprite

For easy entry of this machine language program, be sure to use "The Machine Language Editor: MLX," Appendix D.

```
49152 :169,127,141,013,220,173,075
49158 :021,003,205,098,196,240,001
49164 :015,141,100,196,173,020,145
49170 :003,141,099,196,173,098,216
49176 :196,141,021,003,173,097,143
49182 :196,141,020,003,173,017,068
49188 :208,041,127,141,017,208,010
49194 :173,026,208,009,001,141,088
49200 :026,208,169,240,141,018,082
```

```
49206 :208,169,129,141,013,220,166
49212 :160,059,169,001,145,045,127
49218 :096,173,025,208,041,001,098
49224 :208,003,108,099,196,141,059
49230 :082,196,141,025,208,173,135
49236 :030,208,141,084,196,173,148
49242 :031,208,141,085,196,160,143
49248 :002,177,045,041,015,240,104
49254 :015,206,137,196,208,010,106
49260 :160,059,177,045,141,137,059
49266 :196,032,135,192,076,188,165
49272 :254,096,173,018,208,201,046
49278 :240,176,011,201,025,176,187
49284 :245,144,005,173,082,196,209
49290 :240,237,160,000,177,047,231
49296 :201,211,208,229,200,177,090
49302 :047,201,208,208,222,032,044
49308 :020,193,056,165,052,174,048
49314 :083,196,042,176,105,232,228
49320 :044,136,196,240,247,133,140
49326 :052,142,083,196,138,010,027
49332 :141,093,196,109,101,196,248
49338 :133,047,173,102,196,105,174
49344 :000,133,048,169,008,032,070
49350 :149,193,176,003,032,102,085
49356 :194,032,077,195,032,239,205
49362 :195,238,093,196,024,173,105
49368 :083,196,010,105,016,109,223
49374 :101,196,133,047,173,102,206
49380 :196,105,000,133,048,169,111
49386 :002,032,149,193,176,003,021
49392 :032,102,194,032,077,195,104
49398 :032,035,196,024,160,205,130
49404 :177,047,041,001,240,157,147
49410 :173,136,196,037,052,240,068
49416 :150,032,178,196,024,076,152
49422 :159,192,032,126,193,096,044
49428 :162,008,181,046,157,100,162
49434 :196,202,208,248,173,021,050
49440 :208,141,136,196,162,255,106
49446 :142,083,196,232,134,052,109
49452 :142,082,196,142,093,196,127
49458 :173,016,208,141,088,196,104
49464 :232,165,045,072,024,105,187
49470 :007,133,045,165,046,072,018
49476 :144,002,230,046,160,017,155
49482 :177,045,073,255,157,090,103
49488 :196,189,084,196,072,160,209
49494 :031,017,045,145,045,104,217
49500 :045,136,196,160,017,145,023
```

```
49506 :045,160,045,049,045,145,075
49512 :045,073,255,160,017,049,191
49518 :045,157,095,196,202,208,245
49524 :008,104,133,046,104,133,132
49530 :045,208,203,096,162,008,076
49536 :189,100,196,149,046,202,242
49542 :208,248,173,088,196,141,164
49548 :016,208,173,136,196,141,242
49554 :021,208,096,133,049,169,054
49560 :000,141,094,196,160,206,181
49566 :177,047,133,051,041,012,107
49572 :240,025,173,096,196,036,162
49578 :052,240,018,169,004,036,177
49584 :051,240,097,010,037,051,150
49590 :208,082,165,052,045,091,057
49596 :196,208,071,165,052,044,156
49602 :095,196,208,002,024,096,047
49608 :165,051,041,016,240,248,193
49614 :024,165,047,105,206,133,118
49620 :053,165,048,105,000,133,204
49626 :054,160,017,177,045,069,228
49632 :052,170,160,128,049,053,068
49638 :208,034,138,160,064,049,115
49644 :053,208,039,138,160,096,162
49650 :049,053,208,009,138,160,091
49656 :032,049,053,208,023,024,125
49662 :096,165,052,045,090,196,130
49668 :240,192,032,060,196,096,052
49674 :165,052,073,255,045,136,224
49680 :196,141,136,196,056,096,069
49686 :162,255,232,106,144,252,149
49692 :138,010,164,049,192,002,071
49698 :208,002,105,015,024,109,241
49704 :101,196,133,053,173,102,030
49710 :196,105,000,133,054,160,182
49716 :050,177,053,145,047,160,172
49722 :051,177,053,145,047,236,255
49728 :083,196,144,021,024,160,180
49734 :092,177,053,160,051,113,204
49740 :047,145,047,160,091,177,231
49746 :053,160,050,113,047,145,138
49752 :047,160,206,177,053,133,096
49758 :051,169,255,141,094,196,232
49764 :024,096,173,094,196,208,123
49770 :085,160,091,177,047,200,098
49776 :017,047,240,076,165,051,196
49782 :044,087,196,240,048,162,127
49788 :000,041,128,240,001,232,254
49794 :024,189,000,220,036,049,136
49800 :240,033,042,036,049,208,232
```

```
49806 :049,056,160,051,177,047,170
49812 :160,092,241,047,160,051,131
49818 :145,047,160,050,177,047,012
49824 :160,091,241,047,160,050,141
49830 :145,047,076,192,194,024,076
49836 :160,092,177,047,160,051,091
49842 :113,047,145,047,160,091,013
49848 :177,047,160,050,113,047,010
49854 :145,047,160,050,177,047,048
49860 :141,089,196,200,017,047,118
49866 :240,027,165,051,044,086,047
49872 :196,240,021,162,000,041,100
49878 :128,240,001,232,024,189,004
49884 :000,220,036,049,240,006,003
49890 :042,036,049,240,045,096,222
49896 :174,093,196,024,189,109,249
49902 :196,160,051,113,047,157,194
49908 :109,196,141,092,196,162,116
49914 :001,160,050,177,047,048,221
49920 :002,162,000,160,010,113,191
49926 :047,145,047,160,009,177,079
49932 :047,125,134,196,145,047,194
49938 :024,096,173,089,196,073,157
49944 :128,141,089,196,174,093,077
49950 :196,189,109,196,160,051,163
49956 :241,047,157,109,196,141,159
49962 :092,196,162,001,160,050,191
49968 :177,047,048,002,162,000,228
49974 :160,010,177,047,160,050,146
49980 :241,047,160,010,145,047,198
49986 :160,009,177,047,253,134,078
49992 :196,145,047,024,096,056,124
49998 :162,132,169,000,237,092,102
50004 :196,160,133,177,047,160,189
50010 :010,241,047,160,132,177,089
50016 :047,160,009,241,047,048,136
50022 :027,056,162,164,169,000,168
50028 :237,092,196,160,165,177,111
50034 :047,160,010,241,047,160,011
50040 :164,177,047,160,009,241,150
50046 :047,016,001,096,134,050,214
50052 :160,010,177,045,005,052,069
50058 :145,045,165,051,041,003,076
50064 :240,062,201,001,240,034,154
50070 :201,003,240,074,056,169,125
50076 :040,229,050,168,177,047,099
50082 :170,200,177,047,160,010,158
50088 :145,047,138,160,009,145,044
50094 :047,174,093,196,169,000,085
50100 :157,109,196,096,166,050,186
```

```
50106 :173,089,196,048,005,224,153
50112 :132,240,005,096,224,164,029
50118 :208,251,032,060,196,164,085
50124 :050,076,160,195,169,000,086
50130 :160,050,036,051,112,002,109
50136 :160,091,145,047,200,145,236
50142 :047,164,050,076,160,195,146
50148 :165,052,073,255,045,136,186
50154 :196,141,136,196,096,173,148
50160 :083,196,010,170,024,160,115
50166 :009,177,047,168,208,013,100
50172 :165,052,073,255,045,088,162
50178 :196,141,088,196,076,027,214
50184 :196,165,052,013,088,196,206
50190 :141,088,196,192,001,240,104
50196 :006,169,255,157,000,208,047
50202 :096,160,010,177,047,157,161
50208 :000,208,096,173,083,196,020
50214 :010,170,160,009,177,047,099
50220 :240,006,169,255,157,001,104
50226 :208,096,160,010,177,047,236
50232 :157,001,208,096,056,169,231
50238 :000,160,051,241,047,160,209
50244 :051,145,047,169,000,160,128
50250 :050,241,047,160,050,145,255
50256 :047,096,000,000,000,000,223
50262 :032,064,000,000,000,000,182
50268 :000,000,000,000,000,067,159
50274 :192,049,234,000,240,012,057
50280 :201,045,208,238,032,115,175
50286 :000,032,107,169,208,230,088
50292 :165,020,005,021,208,006,029
50298 :169,255,133,037,026,036,010
50304 :049,044,032,015,016,011,039
50310 :000,255,000,001,067,079,024
50316 :080,089,082,073,071,072,095
50322 :084,032,077,032,066,076,001
50328 :089,084,072,044,032,049,010
50334 :057,056,051,000,128,064,002
50340 :032,016,008,004,002,001,227
50346 :192,192,048,048,012,012,162
50352 :003,003,173,083,196,010,132
50358 :170,189,001,208,201,240,167
50364 :176,004,201,040,176,001,018
50370 :096,056,233,040,168,041,060
50376 :007,141,161,196,169,000,106
50382 :133,054,152,041,248,133,199
50388 :053,010,038,054,010,038,159
50394 :054,024,101,053,133,053,124
50400 :144,002,230,054,173,088,147
```

```
50406 :196,037,052,201,001,189,138
50412 :000,208,133,050,106,201,166
50418 :166,176,205,201,006,144,116
50424 :201,056,233,006,024,106,106
50430 :024,106,024,101,053,230,024
50436 :048,160,101,145,047,133,126
50442 :053,165,054,105,000,133,008
50448 :054,136,145,047,198,048,132
50454 :160,003,006,053,038,054,080
50460 :136,208,249,173,161,196,127
50466 :005,053,133,053,173,017,212
50472 :208,041,032,240,074,032,155
50478 :120,197,024,101,054,133,163
50484 :054,165,050,056,233,012,110
50490 :041,007,170,160,205,177,050
50496 :047,168,173,022,208,041,211
50502 :016,208,010,189,162,196,083
50508 :160,000,017,053,145,053,248
50514 :096,152,106,106,106,106,242
50520 :041,192,224,002,144,014,193
50526 :134,050,202,024,106,106,204
50532 :202,202,240,002,016,248,242
50538 :166,050,160,000,081,053,104
50544 :061,170,196,081,053,145,050
50550 :053,096,173,000,221,041,190
50556 :003,073,003,010,010,010,233
50562 :010,141,161,196,173,024,067
50568 :208,041,014,013,161,196,001
50574 :010,010,096,173,000,221,140
50580 :041,003,073,003,133,054,199
50586 :173,024,208,041,240,102,174
50592 :054,106,102,054,106,024,094
50598 :105,003,133,054,169,000,118
50604 :133,053,169,012,160,231,162
50610 :162,004,076,199,197,032,080
50616 :120,197,024,105,031,133,026
50622 :054,169,000,133,053,160,247
50628 :063,162,032,145,053,136,019
50634 :192,255,208,249,198,054,078
50640 :202,208,244,096,056,176,166
50646 :001,024,173,024,208,041,173
50652 :240,009,008,141,024,208,082
50658 :173,017,208,009,032,141,038
50664 :017,208,173,022,208,041,133
50670 :239,144,002,009,016,141,021
50676 :022,208,096,173,024,208,207
50682 :041,240,009,004,141,024,197
50688 :208,173,017,208,041,223,102
50694 :141,017,208,173,022,208,007
50700 :041,239,141,022,208,096,247
```

64 Paintbox

Chris Metcalf

One of the most powerful features of the Commodore 64, its high-resolution color graphics, can be difficult to use. This machine language program makes accessing this capability easy. By using Atari graphics commands, you can plot points, set colors, or draw lines with just one statement. You can even type in programs originally written for Atari graphics modes 7 and 8 on your 64.

The Commodore 64 is an undeniably powerful computer; its capabilities in high-resolution color graphics, for example, surpass those of the Atari and Apple computers. Nonetheless, these capabilities can be difficult to access; the POKEs and PEEKs required are slow to calculate and slow to execute. "64 Paintbox" takes Atari's far more powerful command set and makes it available to the Commodore 64 user.

BASIC programs written for Atari graphics modes 7 and 8 are easily transferred to the Commodore 64 when this graphics pack is in place. You can type in the program, line by line, adding an exclamation mark (!) before each graphics command to let the 64 BASIC interpreter know that it is a special command. Once this is done, the program will run on the 64 just as it would on an Atari.

64 Paintbox

To enter Program 1, 64 Paintbox, you first need to load and run the MLX program found in Appendix D. MLX makes it easy to type in a machine language program like 64 Paintbox and insures you'll have a working copy the first time. Once you've run MLX, it asks for two addresses. They are:

Starting address: 49152
Ending address: 51197

Now you can begin typing in Program 1. When you're through, save it to tape or disk, using the filename 64

PAINTBOX if you want to use the autoload program described below.

Load 64 Paintbox by entering:

LOAD"*filename*",8,1 (for disk)
LOAD"*filename*",1,1 (for tape)

Then type

SYS 49152:NEW

to initialize the program and reset the pointers. You're now ready to begin typing in any Atari program which uses graphics mode 0, 7, or 8.

To simplify loading the program, you may use Program 2, "64 Boot," the program following the listing of 64 Paintbox. Use "The Automatic Proofreader" program in Appendix C to type in this short autoload routine. Save it on the same disk as 64 Paintbox. (If you're using tape, 64 Boot should precede 64 Paintbox on the tape. You also need to change line 230 so that the 8 is a 1.) Type LOAD"64 BOOT",8 (or just LOAD"64 BOOT" if you've got a Datassette) and RUN; the program will display the command set, load in 64 Paintbox, initialize 64 Paintbox, and execute a NEW. At that point, you can start entering Atari programs.

No matter which method you use to load 64 Paintbox, the Atari graphics commands are easily used. Each command must be preceded by an exclamation mark (and a colon, if following an IF-THEN statement). The command name can be spelled out in full, or abbreviated with a period as on the Atari. However, these abbreviations are *not* expanded when the program is listed. The various parameters follow the command name. Thus a typical syntax might be:

!PLOT 100,100

to plot a point at 100,100.

As with normal BASIC commands, spaces are ignored, whether in the command name or in the parameters.

Since the 64 Paintbox commands are not standard BASIC, the IF-THEN routine will not recognize them as being legal commands unless they're preceded with a colon. So, if you want to plot a point (for example) only if there is no point there already, you might have in the program:

!LOCATE 10,15,A : IF A = 0 THEN : !COLOR 1 : !PLOT 10,15

64 Paintbox Commands

The commands themselves are as follows (abbreviations are enclosed within parentheses):

!GRAPHICS n, (!G.) This command mirrors the Atari GRAPHICS command, and takes only one parameter, *n*, the graphics mode. Since only graphics modes 7 and 8 are supported, all graphics commands between 1 and 6 are treated as if they were 0. As with the Atari, either 7 or 8 may have 16, 32, or 48 added to it. Plus 16 gives no text window; +32 does not clear the graphics screen; and +48 combines the two. Without any of these extra numbers (just !GRAPHICS 8, for instance), the graphics screen will clear and a four-line text window will be set up at the bottom. Regardless of the additional numbers, however, the screens will always be reset to standard Atari graphics colors.

Do not try to use tape or disk with the text window enabled. For example, if you enter LOAD and hit RUN/STOP, the interrupts will be partially disabled, and you will need to reenter the graphics mode (with +32). Attempted disk access will return a ?DEVICE NOT PRESENT ERROR.

The Atari does not allow plotting to the area "under" the text window, but 64 Paintbox does, although the graphics remain concealed until you view what you have done with a !GRAPHICS *n*+48 where *n* is 7 or 8. Furthermore, when working with the graphics screen in immediate mode, it does not need a text window, as the Atari itself does.

!PLOT x,y (!P.) This is the PLOT command. X and y are offset from the top left corner of the screen, and have a range of 0–319 for x and 0–199 for y in graphics mode 8. In GRAPHICS 7, the ranges are 0–159 for x and 0–99 for y. The command is not set up to work in graphics mode 0. The PLOT command plots in the current color register (see the SETCOLOR and COLOR commands). PLOT also sets the starting point for the DRAWTO command.

!POSITION x,y (!PO.) The POSITION command sets the starting point for the DRAWTO command without actually altering the display. X and y are the same as in the PLOT command. This command, like PLOT, positions the graphics screen "cursor" (not the actual text cursor), regardless of the graphics mode.

!DRAWTO x,y, (!.) This command, DRAWTO, draws a line connecting the old starting point to the specified x,y, using the current color register, and then sets the starting point for the next DRAWTO to the specified x,y. The x,y parameters have the same range as for PLOT and POSITION. This command does not affect the screen in GRAPHICS 0.

!SETCOLOR r,c1,c2 (!S.) The SETCOLOR command changes the specified *r* to hue (*c*1) and luminance (c2) in the range 0–15. The format is identical to that of the Atari. The various registers set the colors of the border, the background, the characters, and the pixels according to the table. Note that bit pairs (00, 01, 10, and 11) are used to define single pixels in graphics mode 7. The number below is the graphics register *r* (the first parameter).

SETCOLOR r Values

GRAPHICS 0	GRAPHICS 7	GRAPHICS 8
0 ______	01 pair pixels	______
1 Characters	10 pair pixels	Characters/pixels
2 Background	11 pair pixels	Background
3 ______	______	______
4 Border	Screen color	Border

An unfortunate problem with the way the 64 and the Atari are configured is that in graphics mode 7 the 64's character color in the window is set by SETCOLOR register 2, not 1, and that the text window cannot be set to its own color but takes that of the rest of the screen.

Another problem with register 2 in graphics mode 7 is that this register is set to the background color (or white on old 64s) whenever the screen is cleared. Thus, printing the "clearscreen" character when in graphics mode 7 (even with no window) must be avoided, as all the 11 pixel pairs will become background color: in other words, invisible. Furthermore, any scrolling of the text window in GRAPHICS 7 will scroll strange color data into the 11 pixel pairs. This is, however, no problem in graphics mode 8.

You may be interested to know that executing a !SETCOLOR 2,*c*1,*c*2 in GRAPHICS 7 or a !SETCOLOR 1,*c*1,*c*2 in GRAPHICS 8 causes the character color register at 646 to be set to colors *c*1,*c*2. Thus, previous color codes are disregarded when a !SETCOLOR or !GRAPHICS command is executed (!GRAPHICS calls !SETCOLOR to set up default colors).

The numbers (0–15) that you can use for *c*1 and *c*2 in

SETCOLOR *do* correspond to various hue and luminance settings on the Atari. Take a look at the following chart to see what values in 64 Paintbox match Atari's hue and luminance values.

Matching Atari Hue and Luminance to 64 Paintbox Color Codes

	Luminance							
Hue	0	2	4	6	8	10	12	14
0	0	11	11	11	12	12	15	1
1	0	12	7	7	7	7	1	1
2	0	2	8	8	8	8	15	15
3	0	9	2	2	2	2	8	8
4	0	9	2	2	2	2	8	8
5	0	6	6	6	4	4	4	4
6	0	6	6	6	4	4	4	4
7	0	6	6	6	14	14	14	14
8	0	6	6	6	14	14	14	14
9	0	6	14	14	14	14	3	3
10	0	6	14	14	5	5	13	13
11	0	6	14	14	5	5	13	13
12	0	5	5	5	5	5	13	13
13	0	5	5	5	13	13	7	7
14	0	8	8	8	5	5	13	13
15	0	8	8	8	10	10	10	10

!COLOR r (!C.) This command specifies which color register (given above for !S.) is to be used for plotting and line drawing. In both graphics modes, 0 has the same effect: It erases pixels. In GRAPHICS 8, an odd number for *r* always sets the computer to plot pixels. Registers 1–3 are used in GRAPHICS 7, where register 1 sets bit pair 01, 2 sets 10, and 3 sets 11 (note that this is the SETCOLOR number plus one).

!LOCATE x,y,v (!L.) The LOCATE command returns in floating-point variable v the pixel currently at location x,y and sets the starting point for DRAWTO to the LOCATEd pixel. Thus, for GRAPHICS 8, either a zero (no pixel) or a one (pixel present) is returned. In GRAPHICS 7, a zero also indicates no pixel, while one to three correspond to bit pairs 01, 10 and 11. Using the LOCATE command with a non-floating-point variable does nonproductive (though interesting) things, so it's best to stick to floating-point variables (that is, no % or $ symbol after the variable).

!FILL x,y (!F.) This command is a more powerful version for the Atari XIO fill command. It will fill any area, regardless of the shape. It will stop at any *on* pixel, as well as at the edges of the screen. The x and y parameters determine where it will start and also set a begin-point for future DRAWTO commands. Atari users, remember to draw a line at the left of whatever you are going to fill, as this FILL needs a border to stop at. However, it's much more flexible than the XIO command.

!TEXT x,y,"string" (!T.) The TEXT command allows text to be located starting at any column and row on the GRAPHICS 8 screen (it will execute on GRAPHICS 7 screens, but produces strange multicolored characters). The "string" can be characters enclosed in quotes, a string variable, or combinations of the two. An additional parameter can be passed before the "string"; a 0 or 1 in this position determines whether the computer will use lower/uppercase text or graphics and uppercase. The program is initially set up to use lower- and uppercase. No control characters will be printed, but the RVS ON and RVS OFF characters have their usual effect of putting the characters in-between in reverse video (or inverse video for Atari people). Remember that the x and y parameters must be specified for each TEXT command, although the uppercase/graphics need be set only once to be used repeatedly. The reverse video, however, turns off at the end of the string.

!QUIT (!Q.) This command cuts 64 Paintbox out of the command processing loop and removes the check on error-message display. The program can be restarted with SYS(49152). Calling SYS49152 repeatedly will not, by the way, create any difficulty.

Programmer Notes

Locations 3 and 4 hold two variables used by the interrupt that drives the text window to determine uppercase/graphics for the window and hi-res/multicolor for the graphics. To use location 3 to control the case in the window, POKE 3 with 21 for uppercase/graphics and with 23 for lowercase. (And note that *lowercase is required* for entering commands in lower/uppercase mode.) Register 4 is used by the program to determine pixel plots, LOCATE returns, and so forth, and so may be used to flip between hi-res (8) and multicolor (24).

Other values generate interesting, and harmless, effects. Memory configuration for 64 Paintbox is:

Location	**Function**
0400–07E7	Used as the text window (the bottom four lines, at least)
0800–9FFF	Unused and completely free for BASIC programs
A000–BC7F	BASIC ROM with RAM underneath
BC80–BFFF	Used for data tables and the FILL routine stacks
C000–C7FF	The 2000 bytes of actual program
C800–CBFF	Used as the color screen for all but 11 pixels in GRAPHICS 8
CC00–CFFF	Left free for use by the DOS Wedge or other utility
E000–FFFF	Operating System ROM, with the graphics screen under it

Variable storage is:
Permanent: locations 3–6, 251–254 (interrupt shadows: 3=53272, 4=53270)
Temporary: locations 27–42, 107–113, 158–159, 163–164, 167–170
Non-zero page storage: locations 670–699

Abbreviations for 64 Paintbox Commands

Command	**Abbreviation**
DRAWTO	!. (This takes the place of REM in Atari BASIC.)
PLOT	!P.
POSITION	!PO.
GRAPHICS	!G.
COLOR	!C.
LOCATE	!L.
FILL	!F.
TEXT	!T.
QUIT	!Q.

Demonstrations

Program 3 is a short program which illustrates how 64 Paintbox can be used. It draws several figures on the screen and then waits for a keypress from you to continue. To see this demonstration, make sure 64 Paintbox is in memory (if you load it manually, remember to type SYS 49152 and NEW), then load Program 3. Run it and watch the effects.

Program 1. 64 Paintbox

For easy entry of this machine language program, be sure to use "The Machine Language Editor: MLX," Appendix D.

```
49152 :169,054,133,001,169,224,238
49158 :141,160,188,169,000,141,037
49164 :128,188,170,189,128,188,235
49170 :024,105,064,157,129,188,173
49176 :189,160,188,105,001,157,056
49182 :161,188,232,224,024,144,235
49188 :234,169,001,160,007,153,248
49194 :199,188,153,192,188,010,204
49200 :153,207,188,136,153,192,053
49206 :188,010,136,016,238,169,043
49212 :003,160,006,153,216,188,018
49218 :010,010,136,136,016,247,109
49224 :169,254,160,007,153,224,015
49230 :188,056,042,136,016,248,252
49236 :169,252,160,007,153,231,032
49242 :188,153,239,188,153,247,234
49248 :188,056,042,056,042,136,104
49254 :136,016,239,169,066,141,101
49260 :000,003,169,197,141,001,107
49266 :003,169,134,141,008,003,060
49272 :169,192,141,009,003,169,035
49278 :008,133,004,169,055,133,116
49284 :001,096,160,001,177,122,177
49290 :201,033,240,003,076,228,151
49296 :167,165,212,208,249,032,153
49302 :115,000,165,122,133,158,075
49308 :165,123,133,159,162,255,129
49314 :160,000,165,158,133,122,132
49320 :165,159,133,123,232,032,244
49326 :115,000,041,127,221,242,152
49332 :192,240,245,201,046,240,064
49338 :026,009,128,221,242,192,236
49344 :240,019,189,242,192,048,098
49350 :003,232,208,248,200,200,009
49356 :224,053,144,212,162,011,242
49362 :076,066,197,185,040,193,199
49368 :141,233,192,185,041,193,177
49374 :141,234,192,032,115,000,168
49380 :169,054,133,001,032,046,151
49386 :194,169,055,133,001,076,094
49392 :174,167,068,082,065,087,115
49398 :164,080,076,079,212,080,169
49404 :079,083,073,084,073,079,211
49410 :206,076,079,067,065,084,067
49416 :197,083,069,084,067,079,075
49422 :076,176,067,079,076,176,152
```

```
49428 :071,082,065,080,072,073,207
49434 :067,211,070,073,076,204,215
49440 :081,085,073,212,084,069,124
49446 :088,212,138,194,046,194,142
49452 :031,194,181,196,199,195,016
49458 :150,196,081,193,242,197,085
49464 :060,193,252,198,169,228,132
49470 :141,008,003,169,167,141,179
49476 :009,003,169,139,141,000,017
49482 :003,169,227,141,001,003,106
49488 :096,032,042,197,208,039,182
49494 :138,048,036,041,015,168,020
49500 :192,007,176,032,120,032,139
49506 :000,194,088,169,027,141,205
49512 :017,208,169,023,141,024,174
49518 :208,169,008,141,022,208,098
49524 :133,004,169,199,141,000,250
49530 :221,208,102,076,061,197,219
49536 :192,009,176,249,120,169,019
49542 :059,141,017,208,169,040,000
49548 :141,024,208,169,196,141,251
49554 :000,221,169,008,192,007,231
49560 :208,002,169,024,133,004,180
49566 :141,022,208,169,023,133,086
49572 :003,138,041,016,208,035,093
49578 :169,127,141,013,220,169,241
49584 :001,141,026,208,141,018,199
49590 :208,169,198,141,038,003,171
49596 :169,197,141,039,003,169,138
49602 :100,141,020,003,169,197,056
49608 :141,021,003,208,003,032,096
49614 :000,194,088,138,041,032,187
49620 :208,018,160,000,132,168,130
49626 :169,000,133,170,162,224,052
49632 :032,093,196,169,147,032,125
49638 :210,255,169,004,133,158,135
49644 :166,158,188,251,193,132,044
49650 :168,032,008,196,198,158,234
49656 :016,242,096,008,014,006,118
49662 :009,000,169,000,141,026,087
49668 :208,169,129,141,013,220,116
49674 :169,202,141,038,003,169,220
49680 :241,141,039,003,169,049,146
49686 :141,020,003,169,234,141,218
49692 :021,003,096,032,228,196,092
49698 :160,002,185,167,002,153,191
49704 :251,000,136,016,247,096,018
49710 :032,031,194,032,024,197,044
49716 :240,007,230,253,032,061,107
49722 :194,198,253,032,066,194,227
```

```
49728 :240,045,165,253,074,074,147
49734 :074,170,165,251,069,253,028
49740 :041,248,069,253,024,125,068
49746 :128,188,133,195,189,160,051
49752 :188,101,252,133,196,165,099
49758 :251,041,007,032,024,197,134
49764 :240,005,041,254,013,170,055
49770 :002,170,160,000,096,169,191
49776 :053,120,133,001,177,195,023
49782 :160,054,132,001,088,061,102
49788 :224,188,164,254,240,005,175
49794 :029,192,188,160,000,145,076
49800 :195,096,032,228,196,173,032
49806 :167,002,056,229,251,141,220
49812 :180,002,173,168,002,229,134
49818 :252,141,181,002,173,169,048
49824 :002,056,229,253,133,107,172
49830 :160,001,162,000,032,024,033
49836 :197,240,001,200,165,252,203
49842 :205,168,002,144,036,208,173
49848 :007,173,167,002,197,251,213
49854 :176,027,160,255,162,255,201
49860 :032,024,197,240,001,136,058
49866 :165,251,056,237,167,002,056
49872 :141,180,002,165,252,237,161
49878 :168,002,141,181,002,132,072
49884 :111,134,112,160,001,032,002
49890 :024,197,240,001,200,173,037
49896 :169,002,197,253,176,015,020
49902 :152,073,255,024,105,001,080
49908 :168,165,253,056,237,169,012
49914 :002,133,107,132,167,169,192
49920 :000,141,182,002,133,163,109
49926 :174,180,002,172,181,002,205
49932 :208,014,228,107,176,010,243
49938 :166,107,032,037,195,133,176
49944 :163,076,046,195,032,037,061
49950 :195,141,182,002,076,046,160
49956 :195,132,110,152,074,134,065
49962 :109,138,106,096,169,000,148
49968 :133,158,133,159,133,164,160
49974 :141,183,002,032,049,194,143
49980 :165,252,205,168,002,208,036
49986 :017,165,251,205,167,002,105
49992 :208,010,165,253,205,169,058
49998 :002,208,003,076,034,194,083
50004 :165,163,024,109,180,002,215
50010 :133,163,165,164,109,181,237
50016 :002,133,164,197,110,240,174
50022 :004,144,033,176,006,165,118
```

```
50028 :163,197,109,144,025,165,143
50034 :163,229,109,133,163,165,052
50040 :164,229,110,133,164,165,061
50046 :251,024,101,111,133,251,229
50052 :165,252,101,112,133,252,123
50058 :173,182,002,024,101,107,215
50064 :141,182,002,173,183,002,059
50070 :105,000,141,183,002,197,010
50076 :110,240,004,144,032,208,126
50082 :007,173,182,002,197,109,064
50088 :144,023,173,182,002,229,153
50094 :109,141,182,002,173,183,196
50100 :002,229,110,141,183,002,079
50106 :165,253,024,101,167,133,005
50112 :253,076,057,195,076,061,142
50118 :197,032,042,197,208,248,098
50124 :224,005,176,244,138,072,039
50130 :032,035,197,138,041,015,156
50136 :010,010,133,168,032,035,092
50142 :197,138,041,015,074,170,089
50148 :240,003,074,005,168,133,083
50154 :168,074,168,185,118,196,119
50160 :176,004,074,074,074,074,204
50166 :041,015,164,168,192,003,061
50172 :208,006,224,007,208,002,139
50178 :169,001,133,168,104,170,235
50184 :224,003,240,036,160,240,143
50190 :165,168,032,024,197,208,040
50196 :028,224,000,240,023,202,225
50202 :208,005,032,082,196,240,021
50208 :031,224,001,208,005,032,021
50214 :045,196,240,032,202,202,187
50220 :202,157,032,208,096,202,173
50226 :048,012,202,048,019,240,107
50232 :025,202,032,045,196,202,246
50238 :240,237,160,015,165,168,023
50244 :010,010,010,010,133,168,153
50250 :169,204,133,170,162,200,088
50256 :208,011,162,216,169,220,042
50262 :133,170,165,168,141,134,229
50268 :002,132,006,160,000,132,012
50274 :195,134,196,177,195,037,008
50280 :006,005,168,145,195,200,055
50286 :208,245,232,228,170,208,121
50292 :238,096,011,207,199,113,212
50298 :040,143,146,040,153,170,046
50304 :102,068,102,068,102,238,040
50310 :102,238,100,227,110,227,114
50316 :110,093,085,093,085,215,053
50322 :136,093,136,170,032,042,243
```

```
50328 :197,138,041,003,032,024,075
50334 :197,208,005,041,001,133,231
50340 :254,096,133,254,201,000,078
50346 :208,002,169,001,010,010,058
50352 :010,141,170,002,096,032,115
50358 :031,194,032,234,198,032,135
50364 :170,198,072,169,055,133,217
50370 :001,032,115,000,032,139,001
50376 :176,032,133,177,104,168,222
50382 :169,000,032,145,179,165,128
50388 :098,041,127,133,098,160,101
50394 :004,185,097,000,145,071,208
50400 :136,016,248,096,032,042,026
50406 :197,032,012,197,152,240,036
50412 :008,192,002,176,076,224,146
50418 :064,176,072,142,167,002,097
50424 :140,168,002,032,035,197,054
50430 :032,012,197,152,208,057,144
50436 :224,200,176,053,142,169,200
50442 :002,096,032,024,197,240,089
50448 :006,138,010,170,152,042,022
50454 :168,096,133,170,165,004,246
50460 :041,016,008,165,170,040,212
50466 :096,169,055,133,001,032,008
50472 :253,174,169,055,133,001,057
50478 :032,158,173,032,247,183,103
50484 :169,054,133,001,166,020,083
50490 :164,021,096,162,246,154,133
50496 :162,014,224,128,176,027,027
50502 :134,163,072,169,055,133,028
50508 :001,174,021,003,224,197,184
50514 :240,010,169,032,044,017,082
50520 :208,240,003,032,096,193,092
50526 :104,166,163,076,139,227,201
50532 :173,025,208,141,025,208,112
50538 :169,027,141,017,208,169,069
50544 :199,141,000,221,169,023,097
50550 :141,024,208,169,008,141,041
50556 :022,208,162,000,173,018,195
50562 :208,048,022,162,218,169,189
50568 :196,141,000,221,169,059,154
50574 :141,017,208,169,040,141,090
50580 :024,208,169,008,141,022,208
50586 :208,142,018,208,173,013,148
50592 :220,041,001,240,003,076,229
50598 :049,234,056,032,240,255,008
50604 :224,021,176,006,162,021,014
50610 :024,032,240,255,165,003,129
50616 :141,117,197,165,004,141,181
50622 :151,197,104,168,104,170,060
```

```
50628 :104,064,072,041,127,201,037
50634 :032,144,004,104,076,202,252
50640 :241,104,032,202,241,008,012
50646 :133,170,134,158,132,159,076
50652 :056,032,240,255,224,021,024
50658 :176,006,162,021,024,032,135
50664 :240,255,166,158,164,159,094
50670 :165,170,040,096,032,031,004
50676 :194,032,234,198,169,000,047
50682 :141,174,002,169,000,141,109
50688 :176,002,141,175,002,165,149
50694 :252,208,004,165,251,240,102
50700 :033,165,251,056,237,177,163
50706 :002,133,251,165,252,233,030
50712 :000,133,252,032,170,198,041
50718 :240,229,165,251,024,109,024
50724 :177,002,133,251,165,252,248
50730 :105,000,133,252,230,253,247
50736 :032,170,198,208,011,173,072
50742 :176,002,208,011,032,212,183
50748 :198,169,001,044,169,000,129
50754 :141,176,002,198,253,198,010
50760 :253,032,170,198,208,011,176
50766 :173,175,002,208,011,032,167
50772 :212,198,169,001,044,169,109
50778 :000,141,175,002,230,253,123
50784 :032,061,194,165,251,024,055
50790 :109,177,002,133,251,165,171
50796 :252,105,000,133,252,165,247
50802 :197,201,063,240,048,165,004
50808 :252,240,006,165,251,201,211
50814 :064,176,005,032,170,198,003
50820 :240,168,172,174,002,240,104
50826 :028,136,185,000,189,133,041
50832 :253,185,000,190,133,252,133
50838 :185,000,191,133,251,140,026
50844 :174,002,165,253,201,200,127
50850 :176,226,076,253,197,076,142
50856 :034,194,032,066,194,134,054
50862 :170,189,224,188,073,255,249
50868 :162,053,120,134,001,049,187
50874 :195,230,001,088,072,165,169
50880 :170,041,007,170,104,236,152
50886 :178,002,176,007,074,232,099
50892 :236,178,002,144,249,201,190
50898 :000,096,172,174,002,165,051
50904 :251,153,000,191,165,252,204
50910 :153,000,190,165,253,153,112
50916 :000,189,238,174,002,096,159
50922 :162,001,160,007,032,024,108
```

```
50928 :197,240,002,232,136,142,165
50934 :177,002,140,178,002,096,073
50940 :032,042,197,208,015,224,202
50946 :040,176,011,134,163,032,046
50952 :035,197,208,004,224,025,189
50958 :144,005,162,014,076,066,225
50964 :197,169,000,133,196,165,112
50970 :163,010,010,010,038,196,197
50976 :024,125,128,188,133,195,057
50982 :165,196,125,160,188,133,237
50988 :196,169,055,133,001,032,118
50994 :115,000,032,158,173,165,181
51000 :013,048,025,032,247,183,092
51006 :165,020,041,001,008,173,214
51012 :160,199,040,208,003,041,207
51018 :247,044,009,008,141,160,171
51024 :199,076,049,199,165,098,098
51030 :208,015,032,133,177,160,043
51036 :002,177,071,153,097,000,080
51042 :136,016,248,048,011,165,210
51048 :023,133,022,165,023,056,014
51054 :233,003,133,023,165,097,252
51060 :240,089,169,000,141,180,167
51066 :002,173,160,199,041,251,180
51072 :141,160,199,169,000,133,162
51078 :159,172,180,002,177,098,154
51084 :032,208,199,144,052,010,017
51090 :038,159,010,038,159,010,048
51096 :038,159,133,158,165,159,196
51102 :024,105,216,133,159,160,187
51108 :007,162,055,169,051,120,216
51114 :133,001,177,158,145,195,211
51120 :136,016,249,134,001,088,032
51126 :165,195,024,105,008,133,044
51132 :195,144,006,230,196,165,100
51138 :196,240,010,238,180,002,036
51144 :173,180,002,197,097,208,033
51150 :180,096,170,201,018,208,055
51156 :008,173,160,199,009,004,253
51162 :141,160,199,201,146,208,249
51168 :008,173,160,199,041,251,032
51174 :141,160,199,138,041,127,012
51180 :201,032,144,010,138,201,194
51186 :128,041,191,144,002,233,213
51192 :064,056,096,013,013,013,247
```

Program 2. 64 Boot

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C, when you type in the next two programs.

```
100 IFA=1THENSYS49152:NEW                         :rem 38
110 PRINT:PRINT"{CLR}{DOWN}{15 RIGHT}64 PAINTBOX"
                                                  :rem 80
130 PRINT"{DOWN} {YEL}!GRAPHICS[7] SELECTS GRAPHIC
    S 0,7,8"                                      :rem 123
140 PRINT" {YEL}!COLOR[7] SELECTS COLOR REGISTER"
                                                  :rem 53
150 PRINT" {YEL}!SETCOLOR[7] SETS THE REGISTER'S C
    OLOR"                                         :rem 169
160 PRINT" {YEL}!POSITION[7] PLACES THE GRAPHICS C
    URSOR"                                        :rem 254
170 PRINT" {YEL}!PLOT[7] PLOTS THE POINT SET BY CO
    LOR"                                          :rem 4
180 PRINT" {YEL}!DRAWTO[7] DRAWS TO THE SPECIFIED
    {SPACE}POINT"                                 :rem 175
190 PRINT" {YEL}!LOCATE[7] PUTS THE POINT IN THE V
    ARIABLE"                                      :rem 241
195 PRINT" {YEL}!TEXT[7] PUTS TEXT ON THE SCREEN"
                                                  :rem 221
200 PRINT" {YEL}!QUIT[7] TURNS OFF 64 PAINTBOX"
                                                  :rem 69
210 PRINT"{DOWN}{RIGHT}ALL COMMANDS CAN BE ABBREVI
    ATED WITH":PRINT" A PERIOD (.)"               :rem 249
220 PRINT"{DOWN}{RIGHT}LOADING 64 PAINTBOX FROM 49
    152 TO 51200"                                 :rem 52
230 A=1:LOAD"64 PAINTBOX",8,1                     :rem 114
```

Program 3. 64 Paintbox Demonstrations

```
100 :                                             :rem 203
110 REM DEMOS FOR 64 PAINTBOX                     :rem 164
130 :                                             :rem 206
140 GOSUB700                                      :rem 172
150 DATA "{WHT}SIMPLE FIGURE NUMBER 1"            :rem 127
160 DATA "HIT ANY KEY AFTER THIS DESIGN, AND ALL"
                                                  :rem 231
170 DATA "FOLLOWING DESIGNS, ARE COMPLETE" :rem 17
180 DATA "TO GO ON TO THE NEXT ONE.",          :rem 204
190 FORI=0TO270STEP5:!P.I,100+SIN(I/50)*100:!.319-
    I,100+COS(I/25)*50:NEXT                       :rem 98
200 GETA$:IFA$=""THEN200                          :rem 71
210 GOSUB700                                      :rem 170
220 DATA "THIS FIGURE IS DRAWN IN HI-RES THEN"
                                                  :rem 114
230 DATA "REDISPLAYED IN MULTICOLOR FOR AN":rem 64
240 DATA "INTERESTING EFFECT",                    :rem 25
```

```
250 FORI=0TO309STEP2:!P.I,100+SIN(I/50)*100:!.I+10
    ,100+SIN(I/50)*50:NEXT                      :rem 36
260 GOSUB640:GOSUB700                            :rem 3
270 DATA "HI-RES/MULTICOLOR FIGURE NUMBER 2",
                                                :rem 193
280 FORI=0TO309STEP2:!P.I,100+COS(I/50)*100:!.I+10
    ,100+SIN(I/50)*50:NEXT                      :rem 34
290 GOSUB640:GOSUB700                            :rem 6
300 DATA "SIMPLE FIGURE NUMBER 2",             :rem 164
310 FORI=0TO319STEP2:!P.I,100+SIN(I/50)*100:!.319-
    I,100+COS(I/50)*50:NEXT                     :rem 91
330 GETA$:IFA$=""THEN330                        :rem 79
340 GOSUB700                                   :rem 174
350 DATA "SIMPLE FIGURE NUMBER 3",             :rem 170
390 FORI=0TO310STEP5:!P.I,100+SIN(I/50)*100:!.319-
    I,100+SIN(I/50)*50:NEXT                     :rem 98
420 GETA$:IFA$=""THEN420                        :rem 79
430 GOSUB 700                                  :rem 174
440 DATA "THE NEXT IMAGE IS A CIRCLE",          :rem 52
460 FORI=0TO2*↑-↑/100STEP↑/100:!P.160,100:!.160+CO
    S(I)*100,100-SIN(I)*80                     :rem 206
470 NEXT:C=0:I=2                               :rem 182
480 !S.1,C,I:I=I+1:IFI=16THENI=2:C=C+1:IFC=16THENC
    =0                                          :rem 61
490 GETA$:IFA$=""THEN480                        :rem 92
500 DATA "THIS IS A MULTICOLOR IMAGE"          :rem 117
510 DATA "CREATED WITH LINE AND FILL ROUTINES",
                                                :rem 239
520 !G.7+16:!C.1:N=32:FORI=0TO2*↑STEP↑/N       :rem 170
530 !C.1:!P.80,50:!.80+COS(I)*40,50-SIN(I)*32:NEXT
                                                :rem 160
540 N=16:!C.2:FORI=0TO2*↑STEP↑/N:X=80+COS(I)*50:Y=
    50-SIN(I)*40                               :rem 250
550 !P.X,Y:!.80+COS(I+↑/N)*50,50-SIN(I+↑/N)*40:NEX
    T                                          :rem 215
560 !C.3:!P.0,0:!.159,0:!.159,99:!.0,99:!.0,0
                                                :rem 123
590 GETA$:IFA$=""THEN590                        :rem 95
620 !G.7:!G.0:END                              :rem 118
630 :                                          :rem 211
640 GETA$:IFA$=""THEN640                        :rem 87
650 !G.7+32+16:!S.0,2,8:!S.1,5,8:!S.2,0,14     :rem 37
660 GETA$:IFA$=""THEN660                        :rem 91
670 GOTO750                                    :rem 114
690 :                                          :rem 217
700 PRINT"{CLR}{DOWN}":!G.0:K=0                :rem 254
710 READN$:IFN$=""THEN730                      :rem 171
```

```
720 PRINTTAB(20-LEN(N$)/2)N$"{DOWN}":K=K+1:GOTO710
                                               :rem 27
730 PRINTTAB(17)"E6 @3":PRINTTAB(17)"{RVS} WAIT
    {UP}"                                      :rem 70
740 FORI=1TO350*K:GETA$:IFA$=""THENNEXT       :rem 133
750 !G.8+16:!S.2,0,0:!S.1,RND(1)*15,10:!C.1:RETURN
                                              :rem 149
```

Three Handy Graphics Utilities for the Commodore 64

Colorfill, Underline, and Realtime Clock

Christopher J. Newman

These three utilities are short, yet you'll find many uses for them in your own programs. Change screen color with a single SYS, convert the cursor to an underline instead of a blinking square, and display a realtime clock on the screen with these routines. Best of all, even though two are machine language, they're placed into memory by a BASIC loader, so you don't need to know anything about ML to use them.

Colorfill

Program 1 fills color RAM with a single color. It has several applications. For example, when it's used with a program that POKEs characters to the screen, the color POKE is no longer necessary. Thus programming space can be saved if you're using numerous screen POKEs. This feature is also useful when converting PET programs without the emulator: You no longer need to insert a color POKE for every screen POKE.

Using "Colorfill," you can change the color of all the characters on the screen instantly with one SYS command. In the program listing, line 40 POKEs random colors into location 838 (SL+10). If you want just one color, replace line 40 with something like:

```
40 POKE SL+10,1:SYSSL
```

and all of color RAM fills with white. Any text you see on the screen instantly turns white. Of course, you can change the value POKEd into SL+10 to see other colors.

The machine language program can be relocated by changing the value of SL in line 30 to a new starting location. The machine language portion takes 25 bytes.

Underline

Program 2 replaces the normal blinking square cursor with an underline, for those who prefer this kind of cursor. All the reverse video characters are changed to underlined, normal characters.

It's possible to switch between the standard Commodore character set and the underline set. Once the program has been run, the reverse-video characters can be accessed with POKE 53272,21. You can switch back to underline mode with POKE 53272,31.

The underlined character set is stored in memory locations 14336–16383. To relocate the character set to the bottom of memory, first run this one-line program:

```
10 POKE 44,16:POKE 4096,0:CLR:NEW
```

Then load the underline program, LIST it, delete the *7 characters in line 5, and change *POKE 53272,31* in line 10 to *POKE 53272,19.*

Pressing the RUN/STOP–RESTORE keys will disable the underline cursor function. To reenable it, you'll have to rerun the program.

Realtime Clock

Program 3 is a modification of the idea first demonstrated in the article "Realtime Clock On Your PET Screen," which appeared in the January 1982 issue of *COMPUTE!* magazine. The clock will appear in white, showing the time in tenths of seconds through hours. The color can be changed by changing the two items of data in the last two DATA statements with the value *1* (the color code for white) to your desired color code.

If you accidentally hit RUN/STOP–RESTORE, the clock disappears. You can put it back on the screen by typing

POKE 788,74:POKE 789,3

However, it will have lost time.

The program will not work with the tape cassette, as it occupies the cassette buffer. When you access the disk drive, the clock will briefly stop. The pause lasts only a few tenths of a second.

Program 1. Colorfill

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C, as you enter the following three programs.

```
10 DATA169,216,133,114,169,0,133,113,168,169,14,14
   5,113,200,208,251,230,114                  :rem 139
20 DATA165,114,201,220,208,241,96,            :rem 169
30 SL=828:RESTORE:FORI=SLTOSL+24:READA:POKEI,A:NEX
   T                                          :rem 135
40 POKESL+10,RND(1)*16:SYSSL:GOTO40           :rem 144
```

Program 2. Underline

```
5 Q=2048*7:R=Q+1024:S=53248                    :rem 206
6 FORI=0TO255:POKE1024+I,I:POKE55296+I,14:NEXT
                                                :rem 29
10 POKE53272,31:POKE56334,PEEK(56334)AND254:POKE1,
   PEEK(1)AND251                               :rem 126
20 FORI=0TO1023:POKEQ+I,PEEK(S+I):POKER+I,PEEK(S+I
   ):NEXT:POKE1,PEEK(1)OR4                      :rem 82
30 POKE56334,PEEK(56334)OR1                     :rem 16
40 FORI=7TO1023STEP8:POKER+I,255:NEXT           :rem 85
```

Program 3. Realtime Clock

```
10 FORA=1TO108                                  :rem 51
20 READB                                       :rem 192
30 POKE825+A,B                                  :rem 11
40 NEXT                                        :rem 163
100 PRINT"{CLR}{6 SPACES}HHMMSS"               :rem 197
110 INPUT"TIME";A$                              :rem 53
120 TI$=A$                                     :rem 246
130 FORA=1TO6                                    :rem 3
140 D=VAL(MID$(A$,A,1))                        :rem 200
150 POKE832+A,D                                 :rem 62
160 NEXT                                       :rem 214
200 POKE788,74:POKE789,3                       :rem 111
250 NEW                                        :rem 129
1000 DATA10,10,6,10,6,10,10,0,0,0,0,0,0,0,87,90,16
     5,162                                     :rem 117
1003 DATA105,5,141,73,3,169,86,141,20,3,165,162,14
     1,72,3,205,73,3,48                         :rem 52
```

```
1006 DATA38,105,5,237,73,3,109,72,3,141,73,3,238,7
     1,3,162,7,189                              :rem 79
1009 DATA64,3,221,57,3,48,14,169,0,157,64,3,202,24
     0,6,254,64,3                               :rem 20
1012 DATA76,112,3,162,7,189,64,3,105,48,157,31,4,1
     69,1,157,31,216,202,208,240               :rem 247
1015 DATA169,58,141,31,4,169,1,141,31,216,76,49,23
     4,0,0,0,0                                 :rem 115
```

5
Utilities

Programming Without the Keyboard

Joystick Enhanced Programming

George Leotti

Using a computer's keyboard can be difficult, even impossible for some. Physically handicapped people who want to program on a computer may not be able to use the keyboard. But with "Joystick Enhanced Programming" (JEP), COMPUTE! Publications's first program dedicated to handicapped computer users, the joystick can completely replace the keyboard.

This program isn't only for the handicapped, however. Young children who are not comfortable with using the keyboard may find using the joystick easier and less intimidating.

Being physically disabled myself, I know what it's like to be denied access to something I want or need. So when my friend Marc said he couldn't type for longer than 15 minutes before fatigue became a problem, "Joystick Enhanced Programming" (JEP) came to mind.

Marc has MD (muscular dystrophy), which causes his rapid fatigue. I believe JEP can be useful to other people who thought they couldn't use a computer, because of the keyboard limitations or physical disabilities.

I dedicate JEP to Marc Goldberg.

What is JEP? To put it simply, JEP is a machine language program that allows you to program in BASIC using a joystick plugged into port 2.

Only Once at the Keyboard

Of course, to enter JEP, someone will have to type it in using the keyboard. That may sound like a Catch-22 (you want to use something *besides* the keyboard, but you have to use the keyboard to be able to do *that*), but there's no way around it.

To make it easier to type in JEP, we've provided "The Machine Language Editor: MLX." Make sure you read Appendix D and have a copy of MLX on tape or disk before you begin entering JEP. (Again, someone will have to enter MLX using the keyboard, since you have to have it *before* you start typing in JEP.)

After you've loaded and run MLX, it will ask for two addresses. Those are:

Starting address: 49152
Ending address: 51413

Type in Program 1, JEP. Then, using MLX's Save option, save the program to tape or disk. To load JEP, you have to use the following format:

LOAD"*filename*",8,1 (for disk)
LOAD"*filename*",1,1 (for tape)

Once JEP is loaded, type SYS 49152 and then NEW. The program is instantly available for your use.

Automatic Loading

Another way to load JEP would be to create an autoboot program for it. This is simple if you use the program in "Autoload," another article in this book. After you've entered and saved JEP, just follow the directions in Autoload and you can easily create a routine which automatically loads JEP. The boot program will even do the SYS for you.

The Menu

When JEP is active, the top ten lines of your screen are reserved for a menu from which you make selections to build a BASIC program.

There are two cursors on the screen when the program is active—the normal flashing cursor of the BASIC editor and a nonflashing menu cursor. To prevent confusion and keep a clean menu display, neither cursor is permitted to cross screen line ten. Therefore, if you try to home the editor cursor, it will jump to the first space on line eleven.

To make a selection, move the menu cursor by pushing the joystick in the appropriate direction until it's over the desired character, keyword, or other symbol; then press the fire button. This prints your selection at the editor cursor location.

The menu is *dynamic* in that it changes in response to input from the joystick. What follows is a line-by-line breakdown of the dynamic menu:

ASCII codes. The top line contains the characters with ASCII codes from 33 to 95. This is where the dynamic part comes in. There are 63 characters with only 40 columns to display them. But by moving the menu cursor off the left or right end of this line, you'll cause every character to scroll, or move, one position in the opposite direction.

An example: When you run JEP, the menu cursor is over ASCII character 33 (the exclamation point), in the upper left-hand corner of the screen. By pushing the joystick left *once,* all the characters on the top line will move one position to the right. The character under the menu cursor is now ASCII 95 (left arrow). If you push the stick left (or right if you're on that end of the line), you'll get a continuous scroll.

BASIC keywords. The second line contains every BASIC keyword in the 64's vocabulary (listed alphabetically), including the left parenthesis on words that need it: ASC(, SQR(, LEFT$(, and so on. The number sign is included in keywords that need it—PRINT#, GET#, and INPUT#. The reserved variables ST, TI, and TI$ are represented with their full spelling (TIME$, for example).

The second line scrolls just like the top line. Since there are more characters on this line (because of the number of BASIC keywords), it can take some time to scroll through the entire list. To speed things up, once the scroll begins, hold down the fire button. The words will zip by, but you can still pick out some letters to give you an idea of where you are.

If you release the stick *before* the button, during a speed scroll, you may get something printed that you didn't want. That brings us to the next line.

Special keys. The third line is for the special keys on the 64's keyboard. The keys are: RETURN; SPACE; cursor controls, including up, down, left, right; CLR; HOME; INST; and

DEL. These will work exactly like their keyboard equivalents, with the exception that you can't move the editor cursor above the eleventh line.

Function keys. The fourth line contains the function keys on the right side of the 64 keyboard. On this same line you'll find the letters *BBC.* The first B is for border, the second for background, and the C for character. By putting the menu cursor on any of these letters and pushing the fire button, you may change the color combinations of your entire screen.

Also on line four are the abbreviations COMDRE and CTRL. These represent the Commodore and control keys. By selecting either COMDRE or CTRL, then moving the menu cursor to the top line and selecting a character, it will be printed as if you'd pressed the Commodore or control key first. *Use these for one keystroke only.*

SHIFT and AUTO. Line five displays SHIFT, LOCK, COMDRE, CTRL, AUTO, and finally OFF. SHIFT works like COMDRE and CTRL on line four except that a shifted character is printed.

LOCK, COMDRE, and CTRL, when selected from this line, lock in the SHIFT, Commodore, and control keys respectively. When you select one of these words (LOCK, for instance), it will be reversed as a reminder that function is enabled.

A program like JEP would be much less worthwhile without an automatic line-numbering function. AUTO gives you this feature.

Move the menu cursor over the word AUTO. Press the fire button, then enter a line number using the characters on the top line. That's the last line number you will have to enter. When you terminate the line with RETURN (not to be confused with RETURN from subroutine) from the third menu line, the next line number will be automatically printed for you. The line numbers will increment by 10.

AUTO may also be used as a multiline delete. If you want to delete a range of lines, say 150 to 300, enter the number 150 using the top line. Then move the menu cursor over the word RETURN on the third line, and hold down the fire button until line number 310 is printed. Don't RETURN again, or line 310 will be erased as well. Delete the number 310 using DEL (third line) and you're safe.

To turn off AUTO, move the menu cursor over AUTO and press the fire button. The letters will revert to normal characters, and the AUTO routine will be disengaged.

The final word on line five is OFF. Don't confuse this with turning off AUTO numbers or your computer. It will, when selected, disengage or turn off JEP. If OFF is selected by accident, you may restart JEP by typing SYS 49152, followed by RETURN from the keyboard.

Turning JEP off and on will not affect your BASIC program. In fact, it's similar to pushing the RUN/STOP and RESTORE keys.

You *may* use the RUN/STOP and RESTORE keys to turn off JEP. However, you will have to save the BASIC program you were working on, reload JEP, and type SYS 49152:NEW if you want to use it again. The better way is to use OFF to keep JEP ready to run again.

System Defaults

I have the speed of the menu cursor set to where it's comfortable for Marc. Things may move too fast or too slowly for you, but there are ways to speed things up or slow them down.

Once you have JEP entered, save it to tape or disk. After you have it saved, run it by typing SYS 49152. Now you may maneuver the menu cursor with your joystick and select different things from the menu to get a feel for the way JEP responds. Here are three locations that you can POKE to change speeds:

POKE 49292,2: This is a general location but can be used to change left/right speeds.

POKE 49531,6: This controls up/down speed.

POKE 49603,4: This is for button response.

The numbers POKEd into these locations are the present settings. The lower the number, the faster the response; the higher the number, the slower the response. Note: Since these locations count down to 0, 1 is the fastest setting and 0 is the slowest.

Location 49292 will slow down or speed up *all* responses. Set this first to a comfortable speed for left/right movement. Then set the other locations.

When you've got everything at a comfortable speed, turn JEP OFF and save it by entering SAVE"filename",8,1 for disk or SAVE"filename",1,1 for tape. Now you won't have to set things up each time.

To change the size of the increment of the AUTO line numbers, you can POKE a different increment into location 50415, which now contains 10. Numbers from 1 to 255 are allowed.

During the process of writing a program, it's often desirable to run the program to look for errors. JEP will remain active when a BASIC program is run, but the menu will not be displayed. Even after the test run, the menu won't be displayed unless a scroll occurred upon exiting your program. This is to allow you to read whatever may be printed on the screen.

There are several ways to get the menu back after a BASIC program runs. First, if you used the menu to select RUN, then RETURN, hold down the fire button until the menu appears. You could also use the keyboard RETURN key, cursor down, clear screen, and even cursor up; or the DEL key to restore the menu.

If you would like to keep the menu on screen during a RUN to use for input, instead of the keyboard, you must use the following POKEs *exactly*:

```
POKE 56333,127:POKE 49275,234:POKE 49276,234:POKE 56333,129
```

If you make these changes in *direct mode* (no line numbers), enter all four POKEs on the same line. Why? Because the first POKE disables interrupts, which kills the keyboard until the last POKE. The two POKEs in the middle put the machine language instruction NOP (No OPeration) in place of a branch instruction (BEQ) and its offset.

To return JEP to normal, enter the following on one line:

```
POKE 56333,127:POKE 49275,240:POKE 49276,89:POKE 56333,129
```

Automatic Proofreader for JEP

One of the most useful programs from COMPUTE! Publications, "The Automatic Proofreader," virtually insures error-free programs when you type them in. All the Commodore 64 programs published in *COMPUTE!* magazine, *COMPUTE!'s Gazette,* and COMPUTE! books use the Proofreader to help you type those programs in.

You can use the Proofreader, or at least a variation of it, when you use JEP. "JEProof," this modified version of the Proofreader, is included here as Program 2. Just like Program 1, it's in MLX format. To enter it, make sure you use the MLX program from Appendix D. You need to provide two addresses, which are:

Starting address: 51500
Ending address: 51667

Type in JEProof and save it to tape or disk.

If you want to use JEProof with JEP, this is the process you need to follow.

- LOAD"JEP",8,1 and press RETURN
- Type NEW
- LOAD"JEPROOF",8,1 and press RETURN
- Type NEW
- SYS 49152 and press RETURN

Now you'll see the usual JEP menu at the top of the screen. JEP is active and you can use the joystick to enter:

- SYS 51400 and press (or enter) RETURN

JEProof is enabled and ready to use. To see it at work, enter a simple BASIC line, such as *10 REM* and then RETURN. You'll see a number in reverse video (in this case it should be 069) just to the right of the OFF in the fifth menu line. That's the Proofreader's checksum number. If you look at the BASIC programs in this book, you'll see *:rem xx* (where xx is a number) at the end of each line. That's the number you should see in reverse video if you entered the line correctly. (For more detailed information about the Proofreader—JEProof works in much the same way—read Appendix C.)

There's only one problem with JEProof. If you enter a line at the very bottom of the screen, the reverse video number will appear for just a brief moment, not long enough to really see. For this reason, when you're using JEProof, *make sure not to enter BASIC lines on the very bottom screen line.* You can get around this by scrolling the BASIC lines up the screen using the cursor down (D in menu line 3), then moving the cursor back up to resume typing. It's a bit of a bother, but you should get used to it rather quickly.

Disadvantages of JEP

You'll find difficulty when you try to use other machine language enhancements with JEP in operation. You won't be able to use any program, whether it's BASIC or machine language, which uses memory locations 49152 through 51413.

The DOS wedge supplied with the 1541 does work. Sorry, but JEP won't work with Simon's BASIC.

One other thing to be aware of. If you use a printer attached to the user port, deactivate JEP before using it. JEP uses the locations that are reserved for the RS-232 I/O buffers.

I already know this program is useful to one person. I hope this program will also extend *your* programming time, providing you with many hours of fun (or even frustration) that are part of programming.

Program 1. Joystick Enhanced Programming (JEP)

For easy entry of the two machine language programs which follow, be sure to use "The Machine Language Editor: MLX," Appendix D.

```
49152 :169,147,032,210,255,169,214
49158 :000,133,172,133,173,133,238
49164 :252,133,248,133,247,133,134
49170 :249,169,004,133,250,133,188
49176 :251,141,168,002,169,006,249
49182 :141,167,002,173,048,003,052
49188 :141,112,197,173,049,003,199
49194 :141,113,197,169,102,141,137
49200 :048,003,169,197,141,049,143
49206 :003,173,050,003,141,134,046
49212 :197,173,051,003,141,135,248
49218 :197,169,124,141,050,003,238
49224 :169,197,141,051,003,173,038
49230 :164,197,133,253,173,165,139
49236 :197,133,254,032,198,195,069
49242 :120,173,020,003,072,173,139
49248 :021,003,072,173,168,197,218
49254 :141,020,003,173,169,197,037
49260 :141,021,003,104,141,169,175
49266 :197,104,141,168,197,088,241
49272 :096,165,157,240,089,024,123
49278 :165,172,101,173,240,003,212
49284 :032,198,195,198,251,208,190
49290 :059,169,002,133,251,160,144
49296 :000,162,000,173,000,220,187
49302 :074,176,003,202,144,015,252
49308 :074,176,003,232,144,009,026
49314 :074,176,001,136,074,176,031
49320 :001,200,074,152,208,011,046
49326 :138,208,008,176,019,032,243
49332 :188,193,076,198,192,152,155
49338 :072,032,096,196,104,168,086
49344 :032,217,192,032,096,196,189
49350 :056,032,240,255,224,010,247
49356 :176,008,162,010,032,240,064
```

```
49362 :255,032,198,195,108,168,142
49368 :197,008,104,133,002,152,044
49374 :208,003,076,117,193,016,067
49380 :005,198,248,048,010,096,065
49386 :230,248,165,248,201,040,086
49392 :176,067,096,230,248,165,198
49398 :247,208,010,198,252,016,153
49404 :073,169,063,133,252,208,126
49410 :067,201,002,208,041,200,209
49416 :162,001,070,002,176,002,165
49422 :162,005,198,253,165,253,026
49428 :201,255,208,002,198,254,114
49434 :177,253,240,005,202,208,087
49440 :239,240,010,173,166,197,033
49446 :133,253,173,167,197,133,070
49452 :254,076,255,195,169,039,008
49458 :133,248,096,198,248,165,114
49464 :247,208,014,230,252,164,147
49470 :252,185,170,197,208,002,052
49476 :133,252,076,233,195,201,134
49482 :002,208,035,162,001,070,040
49488 :002,176,002,162,005,230,145
49494 :253,208,002,230,254,177,186
49500 :253,240,005,202,208,243,219
49506 :240,201,173,164,197,133,182
49512 :253,173,165,197,133,254,255
49518 :208,189,169,000,133,248,033
49524 :096,206,167,002,208,036,063
49530 :169,006,141,167,002,160,255
49536 :001,138,016,027,165,247,210
49542 :208,031,160,004,024,165,214
49548 :249,105,080,133,249,165,097
49554 :250,105,000,133,250,230,090
49560 :247,230,247,136,208,236,176
49566 :096,165,247,201,008,208,059
49572 :229,160,004,056,165,249,003
49578 :233,080,133,249,165,250,000
49584 :233,000,133,250,198,247,213
49590 :198,247,136,208,236,096,023
49596 :206,168,002,240,001,096,133
49602 :169,004,141,168,002,166,076
49608 :198,208,211,164,248,165,114
49614 :247,208,067,024,152,101,237
49620 :252,201,063,144,002,233,083
49626 :063,168,173,169,002,240,009
49632 :010,048,003,206,169,002,150
49638 :185,234,197,208,035,173,238
49644 :170,002,240,010,048,003,197
49650 :206,170,002,185,042,198,021
49656 :208,020,173,171,002,240,038
```

```
49662 :012,048,003,206,171,002,184
49668 :185,106,198,208,005,240,178
49674 :003,185,170,197,157,119,073
49680 :002,230,198,096,201,002,233
49686 :208,101,152,024,101,253,093
49692 :141,162,197,165,254,105,028
49698 :000,141,163,197,205,167,139
49704 :197,144,038,240,002,176,069
49710 :010,173,162,197,205,166,191
49716 :197,240,026,144,024,024,195
49722 :173,162,197,237,166,197,166
49728 :133,248,173,164,197,133,088
49734 :253,173,165,197,133,254,221
49740 :032,198,195,164,248,177,066
49746 :253,201,032,240,016,136,192
49752 :016,247,200,198,253,165,143
49758 :253,201,255,208,238,198,167
49764 :254,208,234,166,198,200,080
49770 :177,253,201,032,240,010,251
49776 :157,119,002,232,236,137,227
49782 :002,144,240,202,134,198,014
49788 :096,201,004,208,009,185,059
49794 :170,198,157,119,002,230,238
49800 :198,096,201,006,208,017,094
49806 :192,024,176,009,185,209,169
49812 :198,157,119,002,230,198,028
49818 :096,192,024,208,016,208,130
49824 :120,174,032,208,232,224,126
49830 :016,208,002,162,000,142,184
49836 :032,208,096,192,025,208,165
49842 :014,174,033,208,232,224,039
49848 :016,208,002,162,000,142,202
49854 :033,208,096,192,026,208,185
49860 :058,174,134,002,232,224,252
49866 :016,208,002,162,000,142,220
49872 :134,002,165,253,141,162,041
49878 :197,165,254,141,163,197,051
49884 :169,000,133,253,169,216,136
49890 :133,254,138,160,000,145,032
49896 :253,200,208,251,230,254,092
49902 :165,254,201,220,208,240,246
49908 :173,162,197,133,253,173,055
49914 :163,197,133,254,096,192,005
49920 :035,176,011,173,170,002,055
49926 :048,005,169,001,141,170,028
49932 :002,096,173,171,002,048,248
49938 :250,169,001,141,171,002,240
49944 :096,192,006,176,011,173,166
49950 :169,002,048,235,169,001,142
49956 :141,169,002,096,192,011,135
```

```
49962 :176,027,173,169,002,073,150
49968 :128,141,169,002,162,004,142
49974 :160,006,185,058,199,073,223
49980 :128,153,058,199,200,202,232
49986 :208,244,076,198,195,192,155
49992 :018,176,014,173,170,002,113
49998 :073,128,141,170,002,162,242
50004 :006,160,011,208,223,192,116
50010 :023,176,014,173,171,002,137
50016 :073,128,141,171,002,162,005
50022 :004,160,018,208,205,192,121
50028 :028,144,019,192,031,144,154
50034 :001,096,032,090,192,173,186
50040 :152,197,016,003,032,130,138
50046 :195,108,046,003,173,152,035
50052 :197,073,128,141,152,197,252
50058 :162,004,160,023,201,128,048
50064 :240,014,173,106,196,141,246
50070 :036,003,173,107,196,141,038
50076 :037,003,208,201,173,036,046
50082 :003,141,106,196,173,037,050
50088 :003,141,107,196,169,105,121
50094 :141,036,003,169,196,141,092
50100 :037,003,169,000,132,002,011
50106 :160,004,153,000,002,136,129
50112 :016,250,164,002,208,163,227
50118 :173,134,002,160,000,153,052
50124 :000,216,200,208,250,153,207
50130 :000,217,200,192,144,208,147
50136 :248,032,233,195,032,255,187
50142 :195,032,045,196,032,073,027
50148 :196,032,096,196,096,165,241
50154 :252,162,000,168,185,170,147
50160 :197,240,250,041,063,157,164
50166 :000,004,200,232,224,040,178
50172 :208,240,096,165,253,141,075
50178 :014,196,165,254,141,015,019
50184 :196,160,000,162,000,185,199
50190 :255,255,208,015,168,173,064
50196 :164,197,141,014,196,173,137
50202 :165,197,141,015,196,208,180
50208 :236,041,063,157,080,004,101
50214 :200,232,224,040,208,225,143
50220 :096,160,039,185,234,198,188
50226 :041,063,153,160,004,185,144
50232 :018,199,041,063,153,240,002
50238 :004,185,058,199,153,064,213
50244 :005,136,016,231,096,160,200
50250 :039,169,064,153,040,004,031
50256 :153,120,004,153,200,004,202
```

```
50262 :153,024,005,153,104,005,018
50268 :136,016,238,096,164,248,222
50274 :177,249,073,128,145,249,095
50280 :096,032,087,241,141,159,092
50286 :197,142,160,197,140,161,083
50292 :197,008,201,013,208,003,234
50298 :032,136,196,173,159,197,247
50304 :174,160,197,172,161,197,165
50310 :040,096,160,000,140,154,212
50316 :197,140,155,197,185,000,246
50322 :002,140,153,197,201,058,129
50328 :176,077,201,048,144,073,103
50334 :041,015,170,173,154,197,140
50340 :141,156,197,173,155,197,159
50346 :141,157,197,014,154,197,006
50352 :046,155,197,014,154,197,171
50358 :046,155,197,024,173,154,163
50364 :197,109,156,197,141,154,118
50370 :197,173,155,197,109,157,158
50376 :197,141,155,197,014,154,034
50382 :197,046,155,197,138,024,195
50388 :109,154,197,141,154,197,140
50394 :169,000,109,155,197,141,221
50400 :155,197,200,192,005,208,157
50406 :169,173,153,197,208,001,107
50412 :096,024,169,010,109,154,030
50418 :197,141,154,197,169,000,076
50424 :109,155,197,141,155,197,178
50430 :160,000,140,156,197,140,023
50436 :157,197,140,158,197,140,225
50442 :153,197,162,015,014,154,193
50448 :197,046,155,197,120,248,211
50454 :173,156,197,109,156,197,242
50460 :141,156,197,173,157,197,025
50466 :109,157,197,141,157,197,224
50472 :173,158,197,109,158,197,008
50478 :141,158,197,216,088,202,024
50484 :016,216,164,198,162,002,042
50490 :189,156,197,072,074,074,052
50496 :074,074,032,087,197,104,120
50502 :041,015,032,087,197,202,132
50508 :016,236,169,032,153,119,033
50514 :002,200,132,198,096,205,147
50520 :153,197,240,009,009,048,232
50526 :141,153,197,153,119,002,091
50532 :200,096,141,159,197,032,157
50538 :090,192,173,159,197,032,181
50544 :255,255,141,159,197,032,127
50550 :090,192,173,159,197,096,001
50556 :141,159,197,032,090,192,167
```

```
50562 :173,159,197,032,255,255,177
50568 :141,159,197,032,090,192,179
50574 :169,000,133,172,133,173,154
50580 :173,159,197,096,000,000,005
50586 :000,000,000,000,000,000,154
50592 :000,000,000,000,099,199,202
50598 :211,200,121,192,033,034,189
50604 :035,036,037,038,039,040,141
50610 :041,042,043,044,045,046,183
50616 :047,048,049,050,051,052,225
50622 :053,054,055,056,057,058,011
50628 :059,060,061,062,063,064,053
50634 :065,066,067,068,069,070,095
50640 :071,072,073,074,075,076,137
50646 :077,078,079,080,081,082,179
50652 :083,084,085,086,087,088,221
50658 :089,090,091,092,093,094,007
50664 :095,000,033,034,035,036,209
50670 :037,038,039,040,041,192,113
50676 :219,060,221,062,063,048,149
50682 :033,034,035,036,037,038,207
50688 :039,040,041,091,093,060,108
50694 :061,062,063,186,193,194,253
50700 :195,196,197,198,199,200,173
50706 :201,202,203,204,205,206,215
50712 :207,208,209,210,211,212,001
50718 :213,214,215,216,217,218,043
50724 :091,169,093,255,095,000,227
50730 :129,149,150,151,152,153,158
50736 :154,155,041,223,166,060,079
50742 :220,062,063,048,129,149,213
50748 :150,151,152,153,154,155,207
50754 :041,091,093,060,061,062,218
50760 :063,164,176,191,188,172,002
50766 :177,187,165,180,162,181,106
50772 :161,182,167,170,185,175,100
50778 :171,178,174,163,184,190,126
50784 :179,189,183,173,091,168,055
50790 :093,255,095,000,144,005,182
50796 :028,159,156,030,031,158,158
50802 :018,042,043,044,045,046,096
50808 :047,146,144,005,028,159,137
50814 :156,030,031,158,018,027,034
50820 :029,060,031,062,063,000,121
50826 :001,002,003,004,005,006,159
50832 :007,008,009,074,075,076,137
50838 :013,014,015,016,017,018,243
50844 :019,020,021,022,023,024,029
50850 :025,026,027,028,029,030,071
50856 :006,000,013,013,013,013,226
```

```
50862 :013,013,013,032,032,032,053
50868 :032,032,032,145,145,017,071
50874 :017,157,157,029,029,147,210
50880 :147,147,147,019,019,019,178
50886 :019,019,148,148,148,148,060
50892 :148,020,020,020,020,133,053
50898 :133,133,137,137,137,134,253
50904 :134,134,138,138,138,135,009
50910 :135,135,139,139,139,136,021
50916 :136,136,140,140,140,000,152
50922 :082,069,084,085,082,078,202
50928 :032,083,080,065,067,069,124
50934 :032,085,032,068,032,076,059
50940 :032,082,032,067,076,082,111
50946 :032,072,079,077,069,032,107
50952 :073,078,083,084,032,068,170
50958 :069,076,032,032,070,049,086
50964 :032,070,050,032,070,051,069
50970 :032,070,052,032,070,053,079
50976 :032,070,054,032,070,055,089
50982 :032,070,056,032,066,066,104
50988 :067,032,067,079,077,068,178
50994 :082,069,032,067,084,082,210
51000 :076,032,019,008,009,006,206
51006 :020,032,012,015,003,011,155
51012 :032,003,015,013,004,018,153
51018 :005,032,003,020,018,012,164
51024 :032,001,021,020,015,032,201
51030 :015,006,006,032,032,032,209
51036 :032,032,032,032,032,032,028
51042 :000,032,065,066,083,040,128
51048 :032,065,078,068,032,065,188
51054 :083,067,040,032,065,084,225
51060 :078,040,032,067,072,082,231
51066 :036,040,032,067,076,079,196
51072 :083,069,032,067,076,082,025
51078 :032,067,077,068,032,067,221
51084 :079,078,084,032,067,079,047
51090 :083,040,032,068,065,084,006
51096 :065,032,068,069,070,032,232
51102 :070,078,032,068,073,077,044
51108 :032,069,078,068,032,069,000
51114 :088,080,040,032,070,078,046
51120 :032,070,079,082,032,084,043
51126 :079,032,070,082,069,040,042
51132 :032,071,069,084,032,071,035
51138 :069,084,035,032,071,079,052
51144 :083,085,066,032,071,079,104
51150 :084,079,032,073,070,032,064
51156 :084,072,069,078,032,073,108
```

```
51162 :078,080,085,084,032,073,138
51168 :078,080,085,084,035,032,106
51174 :073,078,084,040,032,076,101
51180 :069,070,084,036,040,032,055
51186 :076,069,078,040,032,076,101
51192 :069,084,032,076,073,083,153
51198 :084,032,076,079,065,068,146
51204 :032,076,079,071,040,032,078
51210 :077,073,068,036,040,032,080
51216 :078,069,087,032,078,069,173
51222 :088,084,032,078,079,084,211
51228 :032,079,078,032,079,080,152
51234 :069,078,032,079,082,032,150
51240 :080,069,069,075,040,032,149
51246 :080,079,075,069,032,080,205
51252 :079,083,040,032,080,082,192
51258 :073,078,084,032,080,082,231
51264 :073,078,084,035,032,082,192
51270 :069,065,068,032,082,069,199
51276 :077,032,082,069,083,084,247
51282 :079,082,069,032,082,069,239
51288 :084,085,082,078,032,082,019
51294 :073,071,072,084,036,040,214
51300 :032,082,078,068,040,032,176
51306 :082,085,078,032,083,065,019
51312 :086,069,032,083,071,078,019
51318 :040,032,083,073,078,040,208
51324 :032,083,080,067,040,032,202
51330 :083,081,082,040,032,083,019
51336 :084,065,084,085,083,032,057
51342 :083,084,069,080,032,083,061
51348 :084,079,080,032,083,084,078
51354 :082,036,040,032,083,089,004
51360 :083,032,084,065,066,040,018
51366 :032,084,065,078,040,032,241
51372 :084,072,069,078,032,084,079
51378 :073,077,069,032,084,073,074
51384 :077,069,036,032,084,079,049
51390 :032,085,083,082,032,086,078
51396 :065,076,040,032,086,069,052
51402 :082,073,070,089,032,087,123
51408 :065,073,084,032,000,013,219
```

Program 2. JEProof

```
51500 :173,036,003,201,077,208,230
51506 :001,096,141,078,201,173,228
51512 :037,003,141,079,201,169,174
51518 :077,141,036,003,169,201,177
51524 :141,037,003,169,000,141,047
51530 :213,201,096,032,087,241,176
```

```
51536 :141,210,201,142,211,201,162
51542 :140,212,201,008,201,013,093
51548 :240,022,201,032,240,007,066
51554 :024,109,213,201,141,213,231
51560 :201,173,210,201,174,211,250
51566 :201,172,212,201,040,096,008
51572 :174,213,201,248,169,000,097
51578 :141,000,001,141,001,001,151
51584 :224,000,240,021,202,024,071
51590 :173,000,001,105,001,141,043
51596 :000,001,173,001,001,105,165
51602 :000,141,001,001,076,128,237
51608 :201,216,173,001,001,009,241
51614 :176,141,002,001,173,000,139
51620 :001,074,074,074,074,009,214
51626 :176,141,001,001,173,000,150
51632 :001,041,015,009,176,141,047
51638 :000,001,162,002,160,000,251
51644 :140,213,201,173,134,002,027
51650 :153,096,217,189,000,001,082
51656 :153,096,005,200,202,016,104
51662 :240,076,105,201,013,013,086
```

One-Touch Keywords

Mark Niggemann

This powerful programming utility puts 52 of the most common BASIC keywords at your fingertips.

The less time spent typing, the more time you have for programming. "One-Touch Keywords" lets you use any of the letter keys in combination with either the SHIFT or Commodore key to instantly print a BASIC keyword on the screen. For example, instead of typing GOSUB, you can hold down SHIFT and press G, and GOSUB will appear as if you had typed the whole keyword. See the table for a list of all the keywords available.

Activating the Keywords

The program is a BASIC loader which moves the machine language from DATA statements into the upper part of free memory. It also protects the machine language from interference by BASIC.

Type in One-Touch Keywords by using "The Automatic Proofreader" program found in Appendix C. It will save you considerable time you might otherwise spend in checking and rechecking your program listing.

A final checksum routine (lines 710–750) is included to aid in finding any errors in the machine language data. After you run the program once, type RUN 700 and the program will check your typing. Recheck the DATA statements if you get an error message. This final checksum is added insurance to the line-by-line checksum provided by the Proofreader.

To activate the machine language, type SYS followed by the number displayed on the screen as the on/off address, then press RETURN. The one-touch keywords will remain enabled even after the RESTORE key has been pressed. To disable the keywords, SYS the on/off address again.

Keywords

Key	w/ SHIFT	w/ Commodore
A	PRINT	PRINT#
B	AND	OR
C	CHR$	ASC
D	READ	DATA
E	GET	END
F	FOR	NEXT
G	GOSUB	RETURN
H	TO	STEP
I	INPUT	INPUT#
J	GOTO	ON
K	DIM	RESTORE
L	LOAD	SAVE
M	MID$	LEN
N	INT	RND
O	OPEN	CLOSE
P	POKE	PEEK
Q	TAB(	SPC(
R	RIGHT$	LEFT$
S	STR$	VAL
T	IF	THEN
U	TAN	SQR
V	VERIFY	CMD
W	DEF	FN
X	LIST	FRE
Y	SIN	COS
Z	RUN	SYS

One-Touch Keywords

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C.

```
140 IF PEEK(PEEK(56)*256)<>120THENPOKE56,PEEK(56)-
    1:CLR                                      :rem 158
150 HI=PEEK(56):BASE=HI*256                     :rem 47
160 PRINT"{CLR}PATIENCE..."                    :rem 206
170 FOR AD=0 TO 211: READ BY                   :rem 153
180 POKE BASE+AD,BY: NEXT AD                    :rem 88
190 :                                          :rem 212
200 REM RELOCATION ADJUSTMENTS                 :rem 184
210 POKE BASE+26,HI: POKE BASE+81,HI             :rem 2
220 POKE BASE+123,HI: POKE BASE+133,HI          :rem 95
230 :                                          :rem 207
231 ::IF PEEK(65532)=34 GOTO 240               :rem 135
```

```
232 ::POKE BASE+9,72: POKE BASE+48,194      :rem 51
233 ::POKE BASE+52,235: POKE BASE+92,160    :rem 139
234 ::POKE BASE+154,72: POKE BASE+157,224 :rem 193
235 ::POKE BASE+158,234                     :rem 230
236 ::                                       :rem 15
240 PRINT"{CLR}* ONE-TOUCH KEYWORDS *"      :rem 88
250 PRINT"ON/OFF:{3 SPACES}SYS{RVS}";BASE :rem 176
260 END                                     :rem 111
270 DATA 120, 173, 143, 2, 201, 32          :rem 127
280 DATA 208, 12, 169, 220, 141, 143        :rem 239
290 DATA 2, 169, 235, 141, 144, 2            :rem 94
300 DATA 88, 96, 169, 32, 141, 143          :rem 155
310 DATA 2, 169,{2 SPACES}0, 141, 144, 2   :rem 237
320 DATA 88, 96, 165, 212, 208, 117         :rem 206
330 DATA 173, 141, 2, 201, 3, 176            :rem 83
340 DATA 110, 201, 0, 240, 106, 169         :rem 175
350 DATA 159, 133, 245, 169, 236, 133        :rem 49
360 DATA 246, 165, 215, 201, 193, 144        :rem 40
370 DATA 95, 201, 219, 176, 91, 56          :rem 160
380 DATA 233, 193, 174, 141, 2, 224         :rem 194
390 DATA 2, 208, 3, 24, 105, 26             :rem 245
400 DATA 170, 189, 159,{2 SPACES}0, 162, 0 :rem 92
410 DATA 134, 198, 170, 160, 158, 132        :rem 40
420 DATA 34, 160, 192, 132, 35, 160         :rem 187
430 DATA 0, 10, 240, 16, 202, 16             :rem 22
440 DATA 12, 230, 34, 208, 2, 230            :rem 78
450 DATA 35, 177, 34, 16, 246, 48           :rem 108
460 DATA 241, 200, 177, 34, 48, 17          :rem 147
470 DATA 8, 142, 211,{2 SPACES}0, 230, 198 :rem 91
480 DATA 166, 198, 157, 119, 2, 174         :rem 215
490 DATA 211,{2 SPACES}0, 40, 208, 234, 230
                                            :rem 131
500 DATA 198, 166, 198, 41, 127, 157          :rem 8
510 DATA 119, 2, 230, 198, 169, 20          :rem 146
520 DATA 141, 119, 2, 76, 220, 235          :rem 139
530 DATA 76, 67, 236                        :rem 127
540 :                                       :rem 211
550 REM *TOKENS FOR SHIFT KEY               :rem 202
560 :                                       :rem 213
570 DATA 153, 175, 199, 135, 161, 129        :rem 56
580 DATA 141, 164, 133, 137, 134, 147        :rem 42
590 DATA 202, 181, 159, 151, 163, 201        :rem 37
600 DATA 196, 139, 192, 149, 150, 155        :rem 52
610 DATA 191, 138                            :rem 20
620 :                                       :rem 210
630 REM *TOKENS FOR COMMODORE KEY           :rem 240
640 :                                       :rem 212
```

```
650 DATA 152, 176, 198, 131, 128, 130        :rem 45
660 DATA 142, 169, 132, 145, 140, 148        :rem 43
670 DATA 195, 187, 160, 194, 166, 200        :rem 54
680 DATA 197, 167, 186, 157, 165, 184        :rem 72
690 DATA 190, 158, 0                        :rem 121
700 ::                                       :rem 11
710 ::REM *CHECKSUM ROUTINE                 :rem 147
720 ::                                       :rem 13
730 ::FOR AD=0 TO 158 : READ BY              :rem 25
740 ::CHKSUM = CHKSUM + BY : NEXT AD        :rem 166
750 ::IF CHKSUM <> 20347 THEN PRINT "ERROR!"
                                            :rem 143
```

Autoload

Dan Carmichael

Have you ever wanted to type LOAD"",8,1 and have your favorite program automatically load and run itself like commercial software packages do? "Autoload" will create a program to do just that.*

When using commercial software, you've probably noticed that typing and entering LOAD"*",8,1 will automatically load and start a program running without having to enter RUN. The first program loaded is known as a *boot program*. It's this program that loads and executes other programs on the disk.

There are a number of different techniques that can accomplish this, such as overwriting the stack or changing *vectors*. (A vector is a pointer to the starting location of a machine language subroutine.) "Autoload" uses the latter method.

Manipulating the Vectors

In the Commodore 64, there's an area of unused memory from locations 679 to 767 ($02A7–$02FF). Like the cassette buffer, this 89-byte area is perfect for holding small machine language programs.

Just past the end of this area of memory is a table of important vectors. In the 64, these vectors are two bytes each, using the low byte/high byte format. By changing the values of these pointers, you can redirect the system to your own programs.

The vector we'll be using for Autoload is the *BASIC Warm Start Vector* at 770–771 ($0302–$0303). This vector points to the main BASIC program loop. This one loop is executed more often than any other routine of BASIC. It checks the keyboard again and again, waiting for input. When a key is pressed, it prints the character on the screen. It also watches for the RETURN key; pressing it sends the routine into action. This BASIC routine looks at the beginning of the line for a number as well. If it finds one, it assumes you're writing a program and enters it as a BASIC line. When no line number is found, it executes the statement in direct mode. After executing the program (or the statement, if there's no number), the computer goes back to the main BASIC program loop, waiting patiently for more from the keyboard.

This vector is also utilized when loading a program. After a program is loaded into the computer, the system returns to the BASIC program mode by looking at this pointer and executing the BASIC warm start program at 42115 ($A483).

By changing the values in this vector, the computer can be directed to execute any machine language program instead of the normal BASIC warm start. In Autoload, changing the pointer value is accomplished by loading a program (which includes the new pointer values) over the pointer.

The automatic boot program that will be created (by Autoload) and saved to disk is placed into the area between 679–750 ($02A7–$02EE). Before it's saved, the vector is changed to point to the start of the autoboot program which is at 679 ($02A7). Then the program and the pointer (locations 679–771, $02A7–$0303) are saved to disk as one module.

This becomes our autoboot program. Here's how it works:

The autoboot program (along with the vector with the changed values) is loaded into memory. If it's the first program in the disk's directory, it can be loaded with the LOAD"*",8,1 format. After the LOAD is finished, the computer looks at the BASIC warm start vector. Because the vector now points to the start of the autoboot program (location 679), that program is executed instead of the normal BASIC warm start routine. The autoboot program, in turn, loads in and executes the program you've specified.

A Newly Created Program

Type in Autoload. It's a BASIC program that POKEs a machine language program into memory. When you're through, save it to disk.

Because Autoload is in the form of a BASIC loader, you can use "The Automatic Proofreader" from Appendix C to help you type it in. Make sure you've read Appendix C and have a copy of the Proofreader on disk before you begin entering this program. There are also two other checksums included in Autoload to verify that the DATA statements were entered correctly.

If you wish to autoboot a program using the

LOAD"*",8,1 syntax, format a new disk and don't save any files on it until after you've created the autoboot program. This will insure that the autoboot program is the first entry in the disk directory.

The first prompt will ask if the program you want to be automatically loaded and run is a BASIC or machine language program. Press *B* or *M*. If you press M for machine language, you'll be asked to supply the beginning address of the ML program. This is the SYS address that starts the ML program running. (In the ML programs in the book, for instance, you'll find that SYS mentioned near the beginning of the article, where details on how to enter and run it are described.) Enter a number, then press RETURN.

Next, enter the name of the program you want to be automatically loaded. The program then instructs you to insert a newly formatted disk into the disk drive. Actually, the disk needs to be freshly formatted only if you wish to use the LOAD"*",8,1 syntax. Saving the autoboot program to a disk that contains other files is fine.

Enter the name you wish to give to the autoboot program you'll be creating. For future reference, you might want to indicate in the filename that it's a boot program. For example, if you want to automatically load and run SPACEGAME, you could name the autoboot program for that game SPACEGAME.BOOT.

After the Autoload program has run and created the autoboot program on the disk, turn off your 64 to reset the system. Be sure to then save a copy of the program you wish to have loaded and run on the same disk as the autoboot program. (It can't load and run a program that isn't there.) Be sure that you save the program with the same filename you told the autoboot program to look for.

To use the autoboot program, type LOAD"*filename*",8,1 where *filename* is the name of the autoboot program you created, *not* the name of the program that autoboot is to load and run. For example, typing LOAD"SPACEGAME.BOOT",8,1 will automatically load and run "SPACEGAME". If you've done everything correctly, the program you specified should automatically run.

Remember that for every program you want to load automatically, you will have to create a separate autoboot program. You can't just enter LOAD"*",8,1 and expect every program on the disk to automatically load. That would be a more complicated program. It wouldn't be that difficult, though; have Autoload load and run your own boot program, and it, in turn, could load any other programs you wanted.

Autoload

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C.

```
5 PRINT"{CLR} PLEASE WAIT..."                     :rem 18
10 B=679:C=767:TT=0                               :rem 51
20 FORA=BTOC:READD:TT=TT+D:POKEA,D:NEXT           :rem 82
25 IFTT<>8554THENPRINT"CHECK DATA STATEMENTS";B;"T
   O";C:END                                       :rem 156
30 B=7168:C=7623:TT=0                             :rem 147
40 FORA=BTOC:READD:TT=TT+D:POKEA,D:NEXT           :rem 84
45 IFTT<>42577THENPRINT"CHECK DATA STATEMENTS";B;"
   TO";C:END                                      :rem 209
50 PRINT "{CLR}{DOWN} AUTO-LOAD A {RVS}B{OFF}ASIC
   {SPACE}OR {RVS}M{OFF}ACHINE LANGUAGE{3 SPACES}P
   ROGRAM?"                                       :rem 124
70 GETA$:IFA$=""THEN70                            :rem 241
80 IFA$="M"THENGOSUB300                           :rem 108
299 SYS7168:END                                   :rem 138
300 PRINT"{CLR}{DOWN} ENTER STARTING ADDRESS OF MA
    CHINE LANG. PROGRAM."                         :rem 24
330 INPUTN:IFN<0ORN>65535THEN300                  :rem 238
340 NN=INT(N/256):POKE722,N-(NN*256):POKE723,NN:PO
    KE721,32                                      :rem 134
345 POKE693,1                                     :rem 202
350 POKE718,32:POKE719,66:POKE720,166:POKE724,76:P
    OKE725,116:POKE726,164:RETURN                 :rem 184
679 DATA169,131,141,2,3,169,164,141               :rem 245
687 DATA3,3,169,8,170,160,0,32                    :rem 245
695 DATA186,255,169,2,162,239,160,2               :rem 255
703 DATA32,189,255,169,0,166,43,164               :rem 250
711 DATA44,32,213,255,32,231,255,165              :rem 25
719 DATA174,133,45,133,47,165,175,133             :rem 93
727 DATA46,133,48,234,169,82,141,119              :rem 47
735 DATA2,169,213,141,120,2,169,13                :rem 180
743 DATA141,121,2,169,3,133,198,96                :rem 195
751 DATA0,0,0,0,0,0,0,0                           :rem 107
759 DATA0,0,0,0,0,0,0,0,0                         :rem 207
7168 DATA162,0,189,171,28,32,210,255              :rem 38
7176 DATA232,224,98,208,245,162,0,32              :rem 38
7184 DATA207,255,201,13,240,8,157,239             :rem 87
7192 DATA2,232,224,16,208,241,142,186             :rem 80
7200 DATA2,162,0,189,13,29,32,210                 :rem 123
7208 DATA255,232,224,59,208,245,162,0             :rem 86
7216 DATA160,0,232,234,208,252,200,208            :rem 116
7224 DATA249,165,197,201,64,240,250,169           :rem 195
```

```
7232 DATA0,133,198,162,0,189,74,29         :rem 198
7240 DATA32,210,255,232,224,85,208,245     :rem 128
7248 DATA162,0,32,207,255,201,13,240       :rem 19
7256 DATA8,157,200,29,232,224,16,208       :rem 36
7264 DATA241,142,111,28,169,167,141,2      :rem 82
7272 DATA3,169,2,141,3,3,169,0             :rem 241
7280 DATA162,200,160,29,32,189,255,169     :rem 143
7288 DATA8,170,160,255,32,186,255,169      :rem 106
7296 DATA167,133,251,169,2,133,252,169     :rem 149
7304 DATA251,162,4,160,3,32,216,255        :rem 229
7312 DATA32,231,255,169,131,141,2,3        :rem 226
7320 DATA169,164,141,3,3,162,0,189         :rem 188
7328 DATA159,29,32,210,255,232,224,41      :rem 84
7336 DATA208,245,96,147,17,32,69,78        :rem 12
7344 DATA84,69,82,32,78,65,77,69           :rem 129
7352 DATA32,79,70,32,13,80,82,79           :rem 102
7360 DATA71,82,65,77,32,84,72,65           :rem 110
7368 DATA84,32,73,83,32,84,79,32           :rem 114
7376 DATA66,69,13,65,85,84,79,77           :rem 134
7384 DATA65,84,73,67,65,76,76,89           :rem 136
7392 DATA32,66,79,79,84,69,68,46           :rem 135
7400 DATA13,17,32,77,65,88,73,77           :rem 107
7408 DATA85,77,32,76,69,78,71,84           :rem 128
7416 DATA72,32,61,32,49,54,13,67           :rem 97
7424 DATA72,65,82,65,67,84,69,82           :rem 122
7432 DATA83,46,17,17,13,147,17,80          :rem 147
7440 DATA76,65,67,69,32,78,69,87           :rem 131
7448 DATA76,89,32,70,79,82,77,65           :rem 130
7456 DATA84,84,69,68,13,32,68,73           :rem 122
7464 DATA83,75,32,73,78,32,68,73           :rem 117
7472 DATA83,75,32,68,82,73,86,69           :rem 125
7480 DATA44,13,84,72,69,78,32,80           :rem 109
7488 DATA82,69,83,83,32,70,49,46           :rem 123
7496 DATA17,13,147,17,32,69,78,84          :rem 167
7504 DATA69,82,32,78,65,77,69,32           :rem 120
7512 DATA79,70,32,66,79,79,84,13           :rem 117
7520 DATA80,82,79,71,46,32,157,84          :rem 158
7528 DATA72,69,78,32,80,82,69,83           :rem 124
7536 DATA83,13,82,69,84,85,82,78           :rem 125
7544 DATA46,17,13,32,77,65,88,73           :rem 112
7552 DATA77,85,77,32,76,69,78,71           :rem 130
7560 DATA84,72,32,61,32,49,54,13           :rem 96
7568 DATA32,67,72,65,82,65,67,84           :rem 124
7576 DATA69,82,83,46,17,17,13,147          :rem 165
7584 DATA17,32,18,84,85,82,78,32           :rem 115
7592 DATA67,79,77,80,85,84,69,82           :rem 138
7600 DATA32,79,70,70,47,79,78,13           :rem 108
7608 DATA84,79,32,82,69,83,69,84           :rem 131
7616 DATA32,86,69,67,84,79,82,83           :rem 130
```

Crunch

Mike Tranchemontagne

Can't decide whether to use lots of REMs and extra spaces to make your program more readable, or keep it tight so that it executes faster? When you have "Crunch" in your programmer's toolbox, you won't have to make that decision. This machine language utility quickly compacts any BASIC program, and even makes sure that vital lines are retained.

When you program, it's almost as if you're being pulled in two opposite directions. On the one hand, you'd like to include lots of REMarks and spaces between keywords to make the program more readable, and to make it easier to locate sections as you debug. But on the other hand, you'd like to use as little memory as possible. The shorter the program, the faster it will run.

This short (264 bytes) machine language program *crunches* a BASIC program in memory. It removes extra spaces and REM statements, making the program shorter. Now you don't have to worry about those two opposite directions; you can write a program overflowing with REMs and spaces, save it for documentation, and then crunch it to increase speed and free up memory. That's the version you'll actually use.

What Gets Crunched?

"Crunch" checks each BASIC program line for unnecessary spaces (those that aren't in quotes or part of a DATA statement), and then removes them.

REM statements are handled with extra care to insure that the BASIC program works exactly the same after crunching. Since it's not unheard of for GOTOs and GOSUBs to refer to a line which contains only a REM, they all can't be deleted. Therefore, any REM-only line is compacted, but not completely eliminated. The line number and the REM statement remain, but any text following *REM* is deleted. We'll call this an *empty REM.*

All other REMs (for instance, those included as part of a line which contains other, non-REM statements) are entirely erased. The connecting colon (:) is also deleted in this case.

As Crunch runs, it prints a number sign (#) for each line of the BASIC program. Each time Crunch removes one or more spaces, or removes part or all of a REM, you'll see a left arrow (←) to show that memory is being compacted. When all BASIC program lines have been crunched, a CLR is performed, control returns to BASIC, and the READY prompt appears. The program has been compacted and is ready to use.

Note that the spaces between a line number and the first statement on that line are not in memory, but are printed by the LIST command. Also, empty REMs, as described above, take up only five bytes each. These lines are easy to spot when you list a program; if you want to remove them, you'll have to do it manually by typing the line number and pressing RETURN. Make sure those empty REM lines are not target lines for GOTO, GOSUB, or IF-THEN statements.

Entering and Running Crunch

Crunch is fully relocatable, and starts at the LOAD address. To enter the program, use MLX, "The Machine Language Editor" found in Appendix D. Unlike other methods of entering machine language programs, MLX is easy to use and will almost insure that you have a working copy of the program when you finish typing it in. MLX will ask for two addresses after you've loaded and run it. Those are:

Starting address: 50400
Ending address: 50663

Now you can type in Crunch. Save it (through the Save option of MLX) to tape or disk. You can load Crunch by entering

LOAD"*filename*",8,1 for disk

or

LOAD"*filename*",1,1 for tape

After loading Crunch, type NEW and press RETURN to reset BASIC's pointers. Now you can enter or load any BASIC program as usual. To start Crunch, type SYS 50400 and hit RETURN. After several seconds (the time depends on how long the BASIC program is), the READY prompt will show. You can list, save, and run the crunched BASIC program as you would any other. (It would be a good idea to first save it to tape or disk, just in case.)

Crunch

For easy entry of this machine language program, be sure to use "The Machine Language Editor: MLX," Appendix D.

```
50400 :165,043,133,251,165,044,001
50406 :133,252,160,001,162,000,170
50412 :177,251,240,053,169,035,137
50418 :032,210,255,160,003,200,078
50424 :177,251,201,000,240,062,155
50430 :201,131,240,058,201,034,095
50436 :240,072,201,143,240,081,213
50442 :201,032,208,233,152,072,140
50448 :024,101,251,133,253,169,179
50454 :000,101,252,133,254,200,194
50460 :232,177,251,201,032,240,137
50466 :248,208,091,133,002,165,113
50472 :045,133,047,133,049,165,100
50478 :046,133,048,133,050,165,109
50484 :055,133,051,165,056,133,133
50490 :052,096,160,000,177,251,026
50496 :170,200,177,251,133,252,223
50502 :134,251,240,160,208,158,197
50508 :240,170,200,177,251,201,035
50514 :000,240,231,201,034,208,228
50520 :245,240,156,192,006,144,047
50526 :002,136,136,152,024,105,137
50532 :001,072,101,251,133,253,143
50538 :169,000,101,252,133,254,247
50544 :200,232,177,251,201,000,149
50550 :208,248,202,208,005,104,069
50556 :208,190,240,204,134,002,078
50562 :169,095,032,210,255,056,179
50568 :165,045,229,002,133,045,243
50574 :165,046,233,000,133,046,253
50580 :162,000,164,002,177,253,138
50586 :129,253,024,165,253,105,059
50592 :001,133,253,165,254,105,047
50598 :000,133,254,197,046,208,236
50604 :235,165,253,197,045,208,251
50610 :229,165,251,072,165,252,032
```

```
50616 :072,160,000,056,177,251,132
50622 :229,002,145,251,170,200,163
50628 :177,251,240,013,233,000,086
50634 :145,251,133,252,134,251,088
50640 :136,240,232,208,230,170,144
50646 :129,251,104,133,252,104,163
50652 :133,251,104,168,169,000,021
50658 :240,154,035,005,255,013,160
```

Disk Surgeon

Gerald E. Sanders

Many operations with your 1540 or 1541 disk drive can be tedious and difficult. This menu-driven program allows you to change a disk name, unscratch and scratch disk files, and even print out various lists of disk files, all with just a few keypresses.

Have you ever needed to unscratch a program or file on a Commodore 1540/1541 disk? Did you ever want to rename an old disk without erasing the other files? Have you ever saved a program to disk and then seen a funny-looking title when you listed the directory? Or found you couldn't determine the right combination of characters to scratch the unwanted file? And then did you search the disk manual in vain to find the commands to rescue you from your predicament?

While there are no neat, one-word commands to solve these types of problems, all the necessary information is there in the manual. The trouble is, it's somewhat scattered and cryptic. It may take some time to find what you're looking for.

But by using "Disk Surgeon," a menu-driven program that allows you to perform several disk operations, you can avoid the disk manual and frustration altogether.

On Call

Use "The Automatic Proofreader" (from Appendix C) to help you type in Disk Surgeon. The Proofreader insures that you'll enter Disk Surgeon correctly the first time. Once you have it typed in, save it to disk. Run it as you would any BASIC program by entering LOAD"*filename*",8.

Simply insert the disk you want to operate on and press any key. The disk drive will whir for a moment, and the disk's name and ID will display. If this is the disk you intended (you've got a chance to change your mind at this point), press the *Y* key and the disk's directory is read into memory. It may take a few moments, so be patient.

You'll see an eight-option menu on the screen. Now you can go to work. The program is self-explanatory for the most part, and takes you step by step through whatever process you select, but a quick review of the options and their features may help.

Operations

Once started, Disk Surgeon can be stopped only by exiting through the main menu. The POKE 808,234 in line 10 disables the RUN/STOP and RESTORE keys. This was done to prevent leaving the program at a point where a direct access file might be left open and a change not completed. That could have unfortunate results. If the fact that the RUN/STOP and RESTORE keys are disabled bothers you, just delete the POKE from line 10.

At any time after the main menu has appeared, you can exit an operation before it's completed by pressing the f1 key at a *Y/N* prompt. Hitting the f1 key returns you to the main menu.

In the unlikely event of a disk read/write error, Disk Surgeon won't crash, but will ask if you want to stop or restart the program. It will restart from the very beginning.

Notice that the messages are color-coded. Blue characters are used for general information and data. Black letters indicate a wait. White's used to signal you that the program is waiting for input. Cyan characters echo your input where necessary, and red letters show errors or cautions.

Here are the eight operations you can perform with the Surgeon.

Operation 1—Change disk name. To change the disk's name, just hit the 1 key and type in the new name. Up to 16 characters are allowed. You can use the DELete key to erase characters if you change your mind or mistype something. There's a check to insure the name is what you wanted; press *Y* if it's okay, *N* if it's not. Disk Surgeon then changes the disk's name. You'll see the new name in the status line at the top of the screen when the main menu again appears.

Operation 2—View directory. Use this to look at the disk's directory. Ten files are displayed per screen. Press any key to continue with the viewing. Once all the files have been shown, you can look at them again by pressing the f7 key, or even print them out (assuming you have a printer connected to your 64) by hitting the f3 key.

Operation 3—Unscratch file. Possibly the most valuable operation in Disk Surgeon, this feature allows you to recover files that you previously scratched (either through Disk Surgeon, or through the *"S0:filename"* method), provided that DOS (Disk Operating System) has not already overwritten them with another file. Operation 3 is 100 percent effective in recovering scratched files if you use it *immediately* after the scratch is performed. The likelihood of success diminishes rapidly as the number of files written to the disk after the scratch increases.

There's another reason you should try to unscratch a file immediately after it's been scratched. If you scratch a file, save a different file to that same disk, then try to unscratch, you may damage the file saved between the scratch and unscratch operations. That's because the saved file may have used some of the sectors freed by the earlier scratch operation. When you try to unscratch, the program may try to retrieve those sectors, ruining the saved file.

Pressing the 3 key sends Disk Surgeon to work. It finds all the scratched files (all the ones not already written over) and displays them one at a time, asking if you want to recover each one. Answer with a *Y* or *N* keypress. If you do want to unscratch that file, you then have to tell the Surgeon what type of file it is. You've got a choice between sequential, program, user, or relative files. Disk Surgeon works for a moment and then validates the disk. The validation is automatically done (in this operation, as well as the two scratch operations) to insure that files are not ruined. It takes a moment. You can verify that the file is back by viewing the directory again when you return to the main menu.

Warning: validation of a disk will de-allocate blocks allocated for random access files. Don't use Disk Surgeon on disks that contain random access files.

Operation 4—Scratch file—leave on directory. One of the two scratch operations, this one allows you to scratch the file, but retains it on the directory. This feature can be useful, especially when you later decide you want to *unscratch* it. As long as its name is still on the directory, you shouldn't have any trouble locating it. Just as in operation 3, you'll see the filenames one at a time. Pressing the *Y* key begins the scratch feature; hitting the *N* key moves you to the next filename. (Warning: After pressing the *Y* key, there is no chance to change your mind. Make sure you want to scratch that file, or you'll have to unscratch it with operation 3.) Before this operation returns to the main menu, it validates the disk.

Operation 5—Scratch file—take off directory. Identical to operation 4, except that the scratched filename is dropped from the directory.

Operation 6—Print directory. If you have a printer connected to your Commodore 64, you can use this operation to print the entire directory, just the valid files, only the deleted files, or all the program files. These options are available once you press the 6 key when the main menu is on the screen.

Operation 7—Go to another disk. Once the Surgeon is working, you can change disks by hitting the 7 key and inserting the new disk you want to operate on.

Operation 8—Exit. Unless you delete the POKE 808,234 from line 10, this is the only way you can exit the Surgeon. You'll see the READY prompt on the screen. Type RUN and press RETURN if you've changed your mind and want to use the Surgeon again.

Not a Medical School

This program is simply a utility. You don't have to know how DOS works in order to use it. But if you do want more detailed information on DOS and how it operates, take a look at "Disk Tricks," which I also wrote. It's in the September 1984 issue of *COMPUTE!'s Gazette.* (In fact, Disk Surgeon is, for the most part, a software package which includes three of the four small programs listed in that article.)

You'll find Disk Surgeon easy to use, and best of all, a tremendous help in several disk manipulations.

Disk Surgeon

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C.

```
10 POKE53280,11:POKE53281,12:PRINT"{BLU}":POKE808,
   234                                        :rem 19
20 DIMF$(144,4),T$(4),M$(3)                   :rem 58
30 FORX=679TO718:READA:POKEX,A:NEXT            :rem 6
40 T$(0)="DELETED":T$(1)="SEQUENT.":T$(2)="PROGRAM
   ":T$(3)="USER"                            :rem 161
45 T$(4)="RELATIVE"                           :rem 67
50 M$(0)="{BLK}PRINTING ENTIRE DIRECTORY...PLEASE
   {SPACE}WAIT."                             :rem 246
60 M$(1)="{BLK} PRINTING VALID FILES ... PLEASE WA
   IT."                                       :rem 95
```

```
70 M$(2)="{BLK} PRINTING SCRATCHED FILES...PLEASE
   {SPACE}WAIT.{BLU}"                          :rem 161
80 M$(3)="{BLK} PRINTING PROGRAM FILES...PLEASE WA
   IT."                                         :rem 11
90 GOTO710                                      :rem 58
100 INPUT#15,E,E$,ET,ES                         :rem 57
110 IFE<20THENRETURN                            :rem 19
120 PRINT"{CLR}{RED}{RVS}{10 SPACES}DISK ERROR !!!
    {16 SPACES}{BLU}":PRINT:PRINT:PRINT        :rem 177
125 PRINT:PRINT                                :rem 236
130 PRINTE,E$:PRINTET,ES                        :rem 31
140 PRINT#15,"I"                               :rem 100
150 CLOSE8:CLOSE15                              :rem 90
160 PRINT:PRINT:PRINT:PRINT"{WHT} PRESS ANY KEY TO
     RESTART PROGRAM."                          :rem 46
170 PRINT:PRINT:PRINT:PRINT:PRINT"{RED}{4 SPACES}H
    IT 'F1' TO QUIT."                           :rem 95
180 WAIT198,1:GETA$:IFA$="{F1}"THEN2410        :rem 118
190 RUN                                        :rem 143
200 PRINT"{BLU}{2 SPACES}MAIN MENU:"            :rem 22
210 PRINT:PRINT"{2 SPACES}1. CHANGE DISK NAME"
                                               :rem 124
230 PRINT:PRINT"{2 SPACES}2. VIEW DIRECTORY"
                                               :rem 125
240 PRINT:PRINT"{2 SPACES}3. UNSCRATCH FILE"
                                                :rem 90
250 PRINT:PRINT"{2 SPACES}4. SCRATCH FILE-LEAVE ON
     DIRECTORY"                                :rem 165
260 PRINT:PRINT"{2 SPACES}5. SCRATCH FILE-TAKE OFF
     DIRECTORY"                                :rem 157
270 PRINT:PRINT"{2 SPACES}6. PRINT DIRECTORY"
                                               :rem 215
280 PRINT:PRINT"{2 SPACES}7. GO TO ANOTHER DISK"
                                                :rem 12
290 PRINT:PRINT"{2 SPACES}8. QUIT PROGRAM":rem 244
300 PRINT:PRINTSPC(9)"{WHT}WHICH OPTION? (1-8)
                                               :rem 240
310 WAIT198,1:GETA$:IFA$<"1"ORA$>"8"THEN310
                                               :rem 169
320 RETURN                                     :rem 117
330 OPEN15,8,15,"I":GOSUB100                    :rem 40
340 OPEN8,8,8,"#":GOSUB100                     :rem 167
350 PRINT#15,"U1:"8;0;T;S:GOSUB100             :rem 233
360 PRINT#15,"B-P:"8;BP:GOSUB100               :rem 104
370 PRINT#8,D$;:GOSUB100                       :rem 156
380 PRINT#15,"U2:"8;0;T;S:GOSUB100             :rem 237
390 CLOSE8:CLOSE15:RETURN                      :rem 122
400 OPEN15,8,15,"I"                            :rem 219
410 OPEN8,8,8,"#"                               :rem 90
420 PRINT#15,"U1:"8;0;T;S:GOSUB100             :rem 231
```

```
430 GET#8,NT$:IFNT$=""THENNT$=CHR$(Ø)        :rem 136
440 NT=ASC(NT$)                               :rem 1Ø1
450 GET#8,NS$:IFNS$=""THENNS$=CHR$(Ø)        :rem 135
460 NS=ASC(NS$)                               :rem 1Ø1
470 BL$=""                                    :rem 2Ø6
480 FORX=1TO254                               :rem 135
490 GET#8,A$:IFA$=""THENA$=CHR$(Ø)           :rem 1Ø7
500 BL$=BL$+A$                                :rem 198
510 NEXT                                      :rem 213
520 GOSUB1ØØ                                  :rem 168
530 CLOSE8:CLOSE15:RETURN                     :rem 118
540 PRINT:PRINT:PRINTSPC(2)"{BLK}DISK WILL NOW BE
    {SPACE}VALIDATED ...."                    :rem 21
545 PRINT:PRINT:PRINTSPC(13)"PLEASE WAIT.":rem 181
550 OPEN15,8,15,"V":CLOSE15:RETURN            :rem 3Ø
560 SYS679:IFA=5THEN1Ø1Ø                      :rem 173
570 PRINTM$(A-1):OPEN4,4:PRINT#4:PRINT#4:PRINT#4,"
    PROGRAM NAME:";CHR$(16);                  :rem 246
580 PRINT #4,"2ØTYPE:":PRINT#4:PRINT#4        :rem 11
590 FORX=ØTO143                               :rem 133
600 IFA=1THENIFLEFT$(F$(X,1),1)=CHR$(Ø)THEN66Ø
                                              :rem 137
610 IFA=2THENIFASC(F$(X,Ø))<129THEN66Ø        :rem 2Ø6
620 IFA=3THENIFASC(F$(X,Ø))>128ORLEFT$(F$(X,1),1)=
    CHR$(Ø)THEN66Ø                            :rem 158
630 IFA=4THENIFASC(F$(X,Ø))<>13ØTHEN66Ø       :rem 8
640 P$=F$(X,1):T=ASC(F$(X,Ø))-128:IFT<ØTHENT=Ø
                                              :rem 94
650 PRINT#4,P$;CHR$(16);"2Ø";T$(T)            :rem 76
660 NEXT                                      :rem 219
670 CLOSE4:PRINTSPC(8)"{BLU}PRINTOUT COMPLETE ..."
                                              :rem 72
680 PRINT:PRINT:PRINT:PRINT"{WHT}HIT ANY KEY TO RE
    TURN TO THE MAIN MENU."                   :rem 14
690 PRINT:PRINT"HIT {RVS}F3{OFF}TO GO TO PRINT OPT
    ION MENU."                                :rem 222
700 WAIT198,1:GETA$:RETURN                    :rem 1ØØ
710 PRINT"{CLR}{RVS}{BLU}{13 SPACES}DISK SURGEON
    {15 SPACES}"                              :rem 123
720 PRINT:PRINT:PRINTSPC(5)"{BLU}PLEASE INSERT THE
     DISK TO BE"                              :rem 72
730 PRINT:PRINTSPC(12)"OPERATED UPON."        :rem 144
740 PRINT:PRINT:PRINT:PRINTSPC(6)"{WHT}PRESS ANY K
    EY WHEN READY."                           :rem 1Ø2
750 WAIT198,1:GETA$:SYS679                    :rem 46
760 PRINT:PRINT:PRINT:PRINT:PRINT             :rem 7Ø
765 PRINTSPC(1)"{BLK}READING DISK CONTENTS, PLEASE
     WAIT."                                   :rem 71
770 T=18:S=Ø:GOSUB4ØØ                         :rem 224
780 FORX=143TO158                             :rem 244
```

```
790 IFMID$(BL$,X,1)=CHR$(96)THEN800          :rem 214
795 IFMID$(BL$,X,1)<>CHR$(160)THENDN$=DN$+MID$(BL$
    ,X,1)                                      :rem 94
800 NEXT                                      :rem 215
810 DI$=MID$(BL$,161,2):SYS679                :rem 137
820 PRINT"{BLU}DISK NAME: "DN$:PRINT:PRINT"DISK ID
    : "DI$                                     :rem 59
830 PRINT:PRINT:PRINTSPC(2)"{WHT}IS THIS THE CORRE
    CT DISK? (Y/N)"                           :rem 192
840 WAIT198,1:GETA$:IFA$<>"Y"ANDA$<>"N"THEN840
                                              :rem 163
850 IFA$="N"THENRUN                           :rem 132
860 SYS679:PRINT:PRINT:PRINT:PRINTSPC(4)"{BLK}READ
    ING DIRECTORY INTO MEMORY."               :rem 142
870 PRINT:PRINT:PRINTSPC(13)"PLEASE WAIT.":rem 182
880 T=18:S=1:R=0                              :rem 142
890 GOSUB400                                  :rem 181
900 FORX=0TO7                                  :rem 31
910 F$(X+R*8,0)=MID$(BL$,X*32+1,1)            :rem 242
920 F$(X+R*8,1)=MID$(BL$,X*32+4,16)            :rem 45
930 F$(X+R*8,2)=CHR$(T)                       :rem 207
940 F$(X+R*8,3)=CHR$(S)                       :rem 208
950 F$(X+R*8,4)=CHR$(X*32+2)                  :rem 195
960 NEXT                                      :rem 222
970 IFNT<>0ANDNS<=18THENT=NT:S=NS:R=R+1:GOTO890
                                               :rem 41
980 IF(X+R*8)>=143THEN1010                    :rem 188
990 FORZ=(X+R*8)TO144:F$(Z,0)=CHR$(0):F$(Z,1)=CHR$
    (0):F$(Z,2)=CHR$(0)                        :rem 37
1000 F$(Z,3)=CHR$(0):F$(Z,4)=CHR$(0):NEXT :rem 219
1010 IFLEN(DN$)<16THENDN$=DN$+" ":GOTO1010 :rem 86
1020 TL$="{CLR}{RVS}{BLU} DISK NAME: "+DN$+"
     {3 SPACES}ID: "+DI$+"{3 SPACES}"        :rem 180
1030 PRINTTL$                                  :rem 21
1040 GOSUB200                                 :rem 215
1050 SYS679:A=VAL(A$):ONAGOTO1060,1480,1670,1950,2
     120,2290,2400,2410                       :rem 145
1060 PRINT:PRINT"{BLU}INPUT NEW DISK NAME UP TO 16
      CHARACTERS."                              :rem 1
1070 PRINT:PRINTSPC(6)"{WHT}PRESS {RVS}RETURN{OFF}
      WHEN FINISHED."                         :rem 141
1080 NN$="":X=0:PRINT:PRINTSPC(6)"{CYN}";   :rem 34
1090 WAIT198,1:GETA$                          :rem 125
1100 IFA$=CHR$(13)THEN1160                    :rem 160
1110 IFA$="{F1}"THEN1010                      :rem 174
1120 IFA$=CHR$(20)ANDLEN(NN$)>0THENX=X-1:NN$=LEFT$
     (NN$,(LEN(NN$)-1))                       :rem 182
1125 IFA$=CHR$(20)ANDLEN(NN$)>0THENPRINTCHR$(20);:
     GOTO1090                                 :rem 199
1130 IFA$=CHR$(20)ANDLEN(NN$)=0THEN1090      :rem 211
```

```
1140 NN$=NN$+A$:X=X+1:PRINTA$;:IFX=16THEN1160
                                              :rem 185
1150 GOTO1090                                 :rem 202
1160 SYS679:PRINT"{BLU}NEW NAME: ";NN$        :rem 215
1170 PRINT:PRINT:PRINTSPC(8)"{WHT}IS THIS CORRECT?
      (Y/N)"                                  :rem 232
1180 WAIT198,1:GETA$:IFA$<>"Y"ANDA$<>"N"ANDA$<>"
     {F1}"THEN1180                            :rem 122
1190 IFA$="N"THENSYS679:GOTO1060              :rem 156
1200 IFA$="{F1}"THEN1010                      :rem 174
1210 IFLEN(NN$)<16THENNN$=NN$+CHR$(160):GOTO1210
                                               :rem 29
1220 SYS679:PRINT:PRINT:PRINT:PRINT:PRINTSPC(10)"
     {BLK}CHANGING DISK NAME."                :rem 114
1230 T=18:S=0:BP=144:D$=NN$:GOSUB330           :rem 75
1240 OPEN15,8,15,"I":CLOSE15                   :rem 36
1250 SYS679:PRINT:PRINT:PRINT:PRINT:PRINTSPC(9)"
     {BLU}NAME CHANGE COMPLETE."              :rem 113
1260 DN$=NN$:FORX=0TO500:NEXT:GOTO1010         :rem 72
1480 PRINT"{BLU} VIEW DIRECTORY:"             :rem 231
1490 PRINT:PRINT" NO.","FILE TYPE","{2 SPACES}FILE
      NAME{BLU}":PRINT                        :rem 154
1500 Z=0:POKE686,4:POKE698,200                :rem 200
1510 FORX=0TO9                                 :rem 79
1520 A=(ASC(F$(X+Z*10,0)))-128:IFA<0THENA=0:rem 89
1530 IF(A=0ANDF$(X+Z*10,1)="")OR(A=0AND(ASC(F$(X+Z
     *10,1))=0))THEN1590                        :rem 2
1540 PRINTX+(Z*10)+1,T$(A),"{2 SPACES}";F$(X+Z*10,
     1)                                       :rem 205
1550 NEXT                                      :rem 10
1560 PRINT:PRINT:PRINTSPC(3)"{WHT}PRESS ANY KEY TO
      CONTINUE LIST{BLU}":WAIT198,1:GETA$      :rem 72
1570 IFA$="{F1}"THEN1630                      :rem 192
1580 SYS679:PRINT:PRINT:Z=Z+1:GOTO1510        :rem 194
1590 PRINT:PRINT:PRINT"{BLU}LIST COMPLETE. {WHT}PR
     ESS {RVS}F7{OFF} TO VIEW AGAIN."          :rem 47
1600 PRINT:PRINTSPC(4)"PRESS {RVS}F3{OFF} TO PRINT
      DIRECTORY."                             :rem 135
1610 PRINT:PRINT" PRESS ANY KEY TO GO TO THE MAIN
     {SPACE}MENU."                              :rem 3
1620 WAIT198,1:GETA$                          :rem 124
1630 POKE686,2:POKE698,40                     :rem 155
1640 IFA$="{F7}"THENSYS679:GOTO1480           :rem 220
1650 IFA$="{F3}"THENSYS679:GOTO2290           :rem 219
1660 GOTO1010                                 :rem 200
1670 PRINT"{BLU}UNSCRATCH FILE:":POKE686,4:POKE698
     ,160                                       :rem 3
1680 V=0                                      :rem 146
1690 FORX=0TO143:A=(ASC(F$(X,0)))-128:IFA<0THENA=0
                                              :rem 114
```

```
1700 IFA>0THENGOTO1890                          :rem 64
1710 IFA=0AND(LEFT$(F$(X,1),1)=CHR$(0)ORLEFT$(F$(X
     ,1),1)=CHR$(160))THEN1890                  :rem 139
1720 SYS679:PRINT:PRINT:PRINTX+1,F$(X,1)      :rem 20
1730 PRINT:PRINT:PRINTSPC(2)"{WHT}WANT TO UNSCRATC
     H THIS FILE?{2 SPACES}(Y/N){BLU}"        :rem 253
1740 WAIT198,1:GETA$:IFA$<>"Y"AND A$<>"N"ANDA$<>"
     {F1}"THEN1740                            :rem 126
1750 IFA$="N"THEN1900                         :rem 137
1760 IFA$="{F1}"THENPOKE686,2:POKE698,40:GOTO1010
                                              :rem 253
1770 SYS679:PRINT"{BLU}WHAT FILE TYPE?"       :rem 115
1780 PRINT:PRINT"{2 SPACES}1. SEQUENTIAL" :rem 194
1790 PRINT:PRINT"{2 SPACES}2. PROGRAM"        :rem 225
1800 PRINT:PRINT"{2 SPACES}3. USER"             :rem 1
1810 PRINT:PRINT"{2 SPACES}4. RELATIVE"        :rem 32
1820 PRINT:PRINT"{RED}{2 SPACES}5. ABORT UNSCRATCH
     {BLU}"                                    :rem 36
1830 PRINT:PRINT:PRINTSPC(10)"{WHT}WHICH TYPE? (1-
     5)"                                      :rem 160
1840 WAIT198,1:GETA$:IFA$<"1"ORA$>"5"THEN1840
                                               :rem 24
1850 IFA$="5"THENPOKE686,2:POKE698,40:GOTO1010
                                              :rem 173
1860 FT=VAL(A$)+128:FT$=CHR$(FT):BP=ASC(F$(X,4)):T
     =ASC(F$(X,2)):S=ASC(F$(X,3))             :rem 146
1870 D$=FT$:GOSUB330                          :rem 131
1880 F$(X,0)=FT$:V=1                           :rem 57
1890 SYS679:PRINT:PRINT:PRINTSPC(11)"{BLK}... WORK
     ING ...{BLU}"                            :rem 141
1900 NEXT                                       :rem 9
1910 SYS679:PRINTSPC(2)"{BLU}NO MORE DELETED FILES
      ON THIS DISK."                          :rem 107
1920 IFV=1THENGOSUB540                        :rem 103
1930 PRINT:PRINT:PRINTSPC(2)"{WHT}HIT ANT KEY TO R
     ETURN TO MAIN MENU."                     :rem 249
1940 WAIT198,1:GETA$:POKE686,2:POKE698,40:GOTO1010
                                              :rem 193
1950 PRINT"{BLU}SCRATCH FILE - LEAVE ON DIRECTORY:
     ":POKE686,4:POKE698,200                   :rem 72
1960 V=0                                      :rem 147
1970 FORX=0TO143:A=(ASC(F$(X,0)))-128:IFA<0THEN206
     0                                        :rem 141
1980 SYS679:PRINT:PRINT:PRINTX+1,F$(X,1)      :rem 28
1990 PRINT:PRINT:PRINTSPC(2)"{WHT}WANT TO SCRATCH
     {SPACE}THIS FILE?{3 SPACES}(Y/N){BLU}":rem 98
2000 WAIT198,1:GETA$:IFA$<>"Y"ANDA$<>"N"ANDA$<>"
     {F1}"THEN2000                            :rem 106
2010 IFA$="N"THEN2070                         :rem 126
```

```
2020 IFA$="{F1}"THENPOKE686,2:POKE698,40:GOTO1010
                                           :rem 243
2030 BP=ASC(F$(X,4)):T=ASC(F$(X,2)):S=ASC(F$(X,3))
                                           :rem 247
2040 D$=CHR$(128):GOSUB330                 :rem 168
2050 F$(X,0)=CHR$(128):V=1                  :rem 94
2060 SYS679:PRINT:PRINT:PRINTSPC(11)"{BLK} ... WOR
     KING ...{BLU}"                        :rem 131
2070 NEXT                                     :rem 8
2080 SYS679:PRINTSPC(5)"{BLU}NO MORE FILES ON THIS
      DISK."                               :rem 118
2090 IFV=1THENGOSUB540                     :rem 102
2100 PRINT:PRINT:PRINTSPC(2)"{WHT}HIT ANT KEY TO R
     ETURN TO MAIN MENU."                  :rem 239
2110 WAIT198,1:GETA$:POKE686,2:POKE698,40:GOTO1010
                                           :rem 183
2120 PRINT"{BLU}SCRATCH FILE - TAKE OFF DIRECTORY:
     ":POKE686,4:POKE698,200                :rem 52
2130 V=0                                   :rem 137
2140 FORX=0TO143:A=(ASC(F$(X,0)))-128:IFA<0THEN223
     0                                     :rem 130
2150 SYS679:PRINT:PRINT:PRINTX+1,F$(X,1)    :rem 18
2160 PRINT:PRINT:PRINTSPC(2)"{WHT}WANT TO SCRATCH
     {SPACE}THIS FILE?{3 SPACES}(Y/N){BLU}":rem 88
2170 WAIT198,1:GETA$:IFA$<>"Y"ANDA$<>"N"ANDA$<>"
     {F1}"THEN2170                         :rem 122
2180 IFA$="N"THEN2240                      :rem 133
2190 IFA$="{F1}"THENPOKE686,2:POKE698,40:GOTO1010
                                           :rem 251
2200 BP=ASC(F$(X,4)):T=ASC(F$(X,2)):S=ASC(F$(X,3))
                                           :rem 246
2210 D$=CHR$(0):GOSUB330                    :rem 60
2220 F$(X,0)=CHR$(0):V=1                   :rem 242
2230 SYS679:PRINT:PRINT:PRINTSPC(11)"{BLK}... WORK
     ING ...{BLU}"                         :rem 130
2240 NEXT                                     :rem 7
2250 SYS679:PRINTSPC(5)"{BLU}NO MORE FILES ON THIS
      DISK."                               :rem 117
2260 IFV=1THENGOSUB540                     :rem 101
2270 PRINT:PRINT:PRINTSPC(2)"{WHT}HIT ANT KEY TO R
     ETURN TO MAIN MENU."                  :rem 247
2280 WAIT198,1:GETA$:POKE686,2:POKE698,40:GOTO1010
                                           :rem 191
2290 PRINT:PRINT:PRINT"{BLU}PRINT DIRECTORY OPTION
     S:"                                   :rem 243
2300 PRINT:PRINT"{2 SPACES}1. PRINT ENTIRE DIRECTO
     RY"                                   :rem 197
2310 PRINT:PRINT"{2 SPACES}2. PRINT ONLY VALID FIL
     ES"                                   :rem 112
```

```
2320 PRINT:PRINT"{2 SPACES}3. PRINT ONLY DELETED F
     ILES"                                  :rem 249
2330 PRINT:PRINT"{2 SPACES}4. PRINT ONLY PROGRAM F
     ILES"                                  :rem 28
2340 PRINT:PRINT"{2 SPACES}5. ABORT PRINT OPTION"
                                            :rem 162
2350 PRINT:PRINT:PRINTSPC(9)"{WHT}WHICH OPTION? (1
     -5)"                                   :rem 13
2360 WAIT198,1:GETA$:IFA$<"1"ORA$>"5"THEN2360
                                            :rem 20
2370 A=VAL(A$):GOSUB560                     :rem 56
2380 IFA$="{F3}"THENSYS679:GOTO2290         :rem 220
2390 POKE686,2:POKE698,40:GOTO1010          :rem 212
2400 RUN                                    :rem 187
2410 PRINT"{CLR}{BLU}":POKE808,237:CLOSE8:CLOSE15:
     END                                    :rem 42
2420 DATA8,72,138,72,152,72,162,2,160,0,24,32,240,
     255,160,0,169,32,153,40,4              :rem 133
2430 DATA153,0,5,153,0,6,153,0,7,200,208,241,104,1
     68,104,170,104,40,96                   :rem 132
```

Machine Language Saver

John O. Battle

Here's an easy way to save machine language programs to tape or disk from your Commodore 64.

You've just written the ultimate character movement routine for your latest videogame, and, of course, it's written in machine language for speed. Now you want to save it for future use. (You certainly don't want to type the routine in and debug it again.) But how do you get it onto tape or disk? The BASIC command SAVE works only for programs written in BASIC. You could load in a machine language monitor program and use its SAVE feature, but perhaps you don't have a monitor; loading the monitor might even overwrite the routine you want to save.

SAVE and LOAD

Here's the solution. "ML Saver" is a BASIC program which loads in a short machine language routine of its own. This routine allows you to easily save other machine language programs to tape or disk. And since it's in machine language, it's extremely fast.

Because it's in the form of a BASIC loader, you can use "The Automatic Proofreader" from Appendix C to help in typing it in. The Proofreader makes it almost impossible to make a mistake when you enter a program.

Once ML Saver is typed in and saved to tape or disk, enter RUN. Since the numbers in the DATA statements in lines 1000–1300 make up a machine language program, they must be typed in *exactly*, no errors allowed. (For that reason, it's an excellent idea to save the program before you try to run it.) The program is self-prompting—simply press the letter *T* (for SAVE to tape) or *D* (for disk) when asked. Then enter the beginning address for the SAVE and press RETURN.

The program will next ask for the final address in the block of memory to be saved. If you press RETURN without entering an ending address, the program will ask instead for the total number of bytes you wish to save (beginning with the byte at the starting address). If your final address is not greater than your starting address, you'll be asked to enter both addresses again.

Finally, the program will allow you to specify a filename for the SAVEd program. This name can be no more than ten characters long.

In order to load a machine language routine that was put on tape or disk by ML Saver, use the standard BASIC command LOAD, but be sure to follow the device number with a comma and a one. For example:

LOAD"*filename*",8,1 (for disk)
LOAD"*filename*",1,1 (for tape)

The ,1 at the end of the LOAD command tells the computer to load the routine into the same memory locations from which it was saved. Without it, the auto-relocating feature of the 64's LOAD command would cause the routine to be stored beginning at the normal start-of-BASIC location.

ML Saver

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C.

```
10 PRINT "{CLR}{9 DOWN}{9 RIGHT}{RVS}MACHINE LANGU
   AGE SAVE{RVS}"                              :rem 239
70 FOR I=7424 TO 7489                          :rem 36
80 READ X                                      :rem 220
90 POKE I,X :NEXT I                            :rem 39
95 FOR I=1 TO 3000:NEXT I                      :rem 52
100 PRINT"{CLR}{10 DOWN}{6 RIGHT}"             :rem 77
110 PRINT "{RVS}T{OFF}APE OR {RVS}D{OFF}ISK"
                                               :rem 161
120 GET D$:IF D$="" THEN 120                   :rem 79
130 IF D$="T" THEN PRINT"{UP}TAPE SELECTED":LF=1:D
    N=1:SA=2                                   :rem 21
140 IF D$="D" THEN PRINT"{UP}DISK SELECTED":LF=15:
    DN=8:SA=15                                 :rem 119
150 IF D$<>"T" AND D$<>"D" THEN PRINT"{UP}":GOTO 1
    20                                         :rem 115
160 POKE 7661,LF                               :rem 88
170 POKE 7662,DN                               :rem 90
180 POKE 7663,SA                               :rem 94
200 INPUT"STARTING ADDRESS FOR SAVE";S         :rem 124
210 S1=INT(S/256)                              :rem 175
```

```
22Ø S2=S-S1*256                                 :rem 33
23Ø POKE 251,S2                                 :rem 13
24Ø POKE 252,S1                                 :rem 14
245 A$=""                                       :rem 129
25Ø INPUT"FINAL ADDRESS OF SAVE";A$             :rem 63
26Ø IF A$="" THEN 3ØØ                           :rem 2Ø7
27Ø F=VAL(A$)                                   :rem 181
28Ø GOTO 32Ø                                    :rem 1Ø4
3ØØ INPUT "{UP}NUMBER OF BYTES TO BE SAVED";N
                                                 :rem 3
31Ø F=S+N-1                                     :rem 65
32Ø F1=INT(F/256)                               :rem 151
33Ø F2=F-F1*256                                 :rem 252
335 IF F<S THEN PRINT"{3 UP}":GOTO 2ØØ          :rem 183
34Ø POKE 7659,F2                                :rem 69
35Ø POKE 766Ø,F1                                :rem 61
4ØØ INPUT"PROGRAM NAME";N$                      :rem 78
41Ø NL=LEN(N$)                                  :rem 14
42Ø IF NL<1Ø THEN 46Ø                           :rem 37
43Ø PRINT"NAME TOO LONG"                        :rem 171
44Ø GOTO 4ØØ                                    :rem 1Ø1
46Ø POKE 7648,NL                                :rem 1Ø4
47Ø FOR I=1 TO NL                               :rem 118
48Ø POKE 7648+I,ASC(MID$(N$,I,1))               :rem 255
49Ø NEXT I                                      :rem 37
5ØØ IF D$="D" THEN PRINT "PRESS ANY KEY TO SAVE"
                                                :rem 129
5Ø5 IF D$="T" THEN PRINT"REWIND TAPE AND PRESS ANY
     KEY"                                       :rem 138
51Ø GET A$                                      :rem 219
52Ø IF A$="" THEN 51Ø                           :rem 2Ø9
53Ø SYS 7472                                    :rem 1Ø7
56Ø END                                         :rem 114
1ØØØ DATA 169,192,32,144,255,173,237,29,174,238,29
     ,172,239,29,32,186,255,173                 :rem 193
11ØØ DATA 224,29,162,225,16Ø,29,32,189,255,96,234,
     234,234,234                                :rem 197
12ØØ DATA 169,Ø,32,144,255,96,234,234,234,234,234,
     234,234,234,234,234                        :rem 68
13ØØ DATA 32,Ø,29,169,251,174,235,29,172,236,29,32
     ,216,255,32,32,29,Ø                        :rem 66
```

Appendices

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. Most of the programs published in this book are written in a computer language called BASIC. BASIC is easy to learn and is built into all Commodore 64s.

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.

Braces and Special Characters

The exception to this typing rule is when you see the braces, such as {DOWN}. Anything within a set of braces is a special character or characters that cannot easily be listed on a printer. When you come across such a special statement, refer to Appendix B, "How to Type In Programs."

About DATA Statements

Some programs contain a section or sections of DATA statements. These lines provide information needed by the program. Some DATA statements contain actual programs (called machine language); others contain graphics codes. These lines 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 and STOP key 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. Do you know how to enter reverse video, lowercase, and control characters? It's all explained in your computer's manuals.

A Quick Review

1. Type in the program a line at a time, in order. Press RETURN at the end of each line. Use backspace or the back arrow to correct mistakes.
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.

How to Type In Programs

To make it easy to know exactly what to type when entering one of these programs into your computer, we have established the following listing conventions.

Generally, Commodore 64 program listings will contain words within braces which spell out any special characters: {DOWN} would mean to press the cursor down key. {5 SPACES} would mean to press the space bar five times.

To indicate that a key should be *shifted* (hold down the SHIFT key while pressing the other key), the key would be underlined in our listings. For example, S would mean to type the S key while holding the SHIFT key. This would appear on your screen as a heart symbol. If you find an underlined key enclosed in braces (e.g., {10 N}) , you should type the key as many times as indicated (in our example, you would enter ten shifted N's).

If a key is enclosed in special brackets, < >, you should hold down the *Commodore key* while pressing the key inside the special brackets. (The Commodore key is the key in the lower left corner of the keyboard.) Again, if the key is preceded by a number, you should press the key as many times as necessary.

Rarely, you'll see a solitary letter of the alphabet enclosed in braces. These characters can be entered by holding down the CTRL key while typing the letter in the braces. For example, {A} would indicate that you should press CTRL-A.

About the *quote mode:* You know that you can move the cursor around the screen with the CRSR keys. Sometimes a programmer will want to move the cursor under program control. That's why you see all the {LEFT}'s, {HOME}'s, and {BLU}'s in our programs. The only way the computer can tell the difference between direct and programmed cursor control is the quote mode.

Once you press the quote (the double quote, SHIFT-2), you are in the quote mode. If you type something and then try to change it by moving the cursor left, you'll only get a bunch of reverse-video lines. These are the symbols for cursor left. The only editing key that isn't programmable is the DEL key;

you can still use DEL to back up and edit the line. Once you type another quote, you are out of quote mode.

You also go into quote mode when you INSerT spaces into a line. In any case, the easiest way to get out of quote mode is to just press RETURN. You'll then be out of quote mode and you can cursor up to the mistyped line and fix it.

Use the following table when entering cursor and color control keys:

When You Read:	Press:	See:	When You Read:	Press:	See:
{CLR}	SHIFT CLR/HOME		Ɛ 1 Ǝ	COMMODORE 1	
{HOME}	CLR/HOME		Ɛ 2 Ǝ	COMMODORE 2	
{UP}	SHIFT ↑ CRSR ↓		Ɛ 3 Ǝ	COMMODORE 3	
{DOWN}	↑ CRSR ↓		Ɛ 4 Ǝ	COMMODORE 4	
{LEFT}	SHIFT ← CRSR →		Ɛ 5 Ǝ	COMMODORE 5	
{RIGHT}	← CRSR →		Ɛ 6 Ǝ	COMMODORE 6	
{RVS}	CTRL 9		Ɛ 7 Ǝ	COMMODORE 7	
{OFF}	CTRL 0		Ɛ 8 Ǝ	COMMODORE 8	
{BLK}	CTRL 1		{ F1 }	f1	
{WHT}	CTRL 2		{ F2 }	SHIFT f1	
{RED}	CTRL 3		{ F3 }	f3	
{CYN}	CTRL 4		{ F4 }	SHIFT f3	
{PUR}	CTRL 5		{ F5 }	f5	
{GRN}	CTRL 6		{ F6 }	SHIFT f5	
{BLU}	CTRL 7		{ F7 }	f7	
{YEL}	CTRL 8		{ F8 }	SHIFT f7	

The Automatic Proofreader

Charles Brannon

"The Automatic Proofreader" will help you type in program listings without typing mistakes. It is a short error-checking program that hides itself in memory. When activated, it lets you know immediately after typing a line from a program listing if you have made a mistake. Please read these instructions carefully before typing any programs in this book.

Preparing the Proofreader

1. Using the listing below, type in the Proofreader. Be very careful when entering the DATA statements—don't type an l instead of a 1, an O instead of a 0, extra commas, and so on.

2. Save the Proofreader on tape or disk at least twice *before running it for the first time.* This is very important because the Proofreader erases part of itself when you first type RUN.

3. After the Proofreader is saved, type RUN. It will check itself for typing errors in the DATA statements and warn you if there's a mistake. Correct any errors and save the corrected version. Keep a copy in a safe place—you'll need it again and again, every time you enter a program from this book, *COMPUTE!'s Gazette,* or *COMPUTE!* magazine.

4. When a correct version of the Proofreader is run, it activates itself. You are now ready to enter a program listing. If you press RUN/STOP–RESTORE, the Proofreader is disabled. To reactivate it, just type the command SYS 886 and press RETURN.

Using the Proofreader

All listings in this book have a checksum number appended to the end of each line. An example is *":rem 123". Don't enter this statement when typing in a program.* It is just for your information. The rem makes the number harmless if someone does type it in. It will, however, use up memory if you enter it, and it will confuse the Proofreader, even if you entered the rest of the line correctly.

When you type in a line from a program listing and press RETURN, the Proofreader displays a number at the top of your screen. *This checksum number must match the checksum number in the printed listing.* If it doesn't, it means you typed the line differently than the way it is listed. Immediately recheck your typing. Remember, don't type the rem statement with the checksum number; it is published only so you can check it against the number which appears on your screen.

The Proofreader is not picky with spaces. It will not notice extra spaces or missing ones. This is for your convenience, since spacing is generally not important. But occasionally proper spacing *is* important, so be extra careful with spaces, since the Proofreader will catch practically everything else that can go wrong.

There's another thing to watch out for: If you enter the line by using abbreviations for commands, the checksum will not match up. But there is a way to make the Proofreader check it. After entering the line, LIST it. This eliminates the abbreviations. Then move the cursor up to the line and press RETURN. It should now match the checksum. You can check whole groups of lines this way.

Special Tape SAVE Instructions

When you're done typing a listing, you must disable the Proofreader before saving the program on tape. Disable the Proofreader by pressing RUN/STOP–RESTORE (hold down the RUN/STOP key and sharply hit the RESTORE key). This procedure is not necessary for disk, *but you must disable the Proofreader this way before a tape SAVE.*

A SAVE to tape erases the Proofreader from memory, so you'll have to load and run it again if you want to type another listing. A SAVE to disk does not erase the Proofreader.

Hidden Perils

The Proofreader's home in the 64 is not a very safe haven. Since the cassette buffer is wiped out during tape operations, you need to disable the Proofreader with RUN/STOP–RESTORE before you save your program. This applies only to tape use. Disk users have nothing to worry about.

Not so for 64 owners with tape drives. What if you type in a program in several sittings? The next day, you come to your computer, load and run the Proofreader, then try to load the partially completed program so you can add to it. But since the Proofreader is trying to hide in the cassette buffer, it's wiped out!

What you need is a way to load the Proofreader after you've loaded the partial program. The problem is, a tape LOAD to the buffer destroys what it's supposed to load.

After you've typed in and run the Proofreader, enter the following lines in direct mode (without line numbers) exactly as shown:

```
A$="PROOFREADER.T":B$="{10 SPACES}":FORX=1TO4:A$=A
  $+B$:NEXTX
FORX=886TO1018:A$=A$+CHR$(PEEK(X)):NEXTX
OPEN1,1,1,A$:CLOSE1
```

After you enter the last line, you will be asked to press record and play on your cassette recorder. Put this program at the beginning of a new tape. This gives you a new way to load the Proofreader. Anytime you want to bring the Proofreader into memory without disturbing anything else, put the cassette in the tape drive, rewind, and enter:

```
OPEN1:CLOSE1
```

You can now start the Proofreader by typing SYS 886. To test this, PRINT PEEK (886) should return the number 173. If it does not, repeat the steps above, making sure that A$ ("PROOFREADER.T") contains 13 characters and that B$ contains 10 spaces.

The Proofreader will load itself into the cassette buffer whenever you type OPEN1:CLOSE1 and PROOFREADER.T is the next program on your tape. It does not disturb the contents of BASIC memory.

Replace Original Proofreader

If you typed in the original version of the Proofreader from the October 1983 issue of *COMPUTE!'s Gazette*, you should replace it with the improved version below.

Automatic Proofreader

```
100 PRINT"{CLR}PLEASE WAIT...":FORI=886TO1018:READ
    A:CK=CK+A:POKEI,A:NEXT                    :rem 86
110 IF CK<>17539 THEN PRINT"{DOWN}YOU MADE AN ERRO
    R":PRINT"IN DATA STATEMENTS.":END        :rem 115
120 SYS886:PRINT"{CLR}{2 DOWN}PROOFREADER ACTIVATE
    D.":NEW                                   :rem 24
```

```
886 DATA 173,036,003,201,150,208                     :rem 38
892 DATA 001,096,141,151,003,173                     :rem 36
898 DATA 037,003,141,152,003,169                     :rem 45
904 DATA 150,141,036,003,169,003                     :rem 30
910 DATA 141,037,003,169,000,133                     :rem 26
916 DATA 254,096,032,087,241,133                     :rem 50
922 DATA 251,134,252,132,253,008                     :rem 36
928 DATA 201,013,240,017,201,032                     :rem 22
934 DATA 240,005,024,101,254,133                     :rem 27
940 DATA 254,165,251,166,252,164                     :rem 51
946 DATA 253,040,096,169,013,032                     :rem 47
952 DATA 210,255,165,214,141,251                     :rem 38
958 DATA 003,206,251,003,169,000                     :rem 34
964 DATA 133,216,169,019,032,210                     :rem 43
970 DATA 255,169,018,032,210,255                     :rem 47
976 DATA 169,058,032,210,255,166                     :rem 58
982 DATA 254,169,000,133,254,172                     :rem 48
988 DATA 151,003,192,087,208,006                     :rem 52
994 DATA 032,205,189,076,235,003                     :rem 52
1000 DATA 032,205,221,169,032,032                    :rem 66
1006 DATA 210,255,032,210,255,173                    :rem 75
1012 DATA 251,003,133,214,076,173                    :rem 75
1018 DATA 003                                        :rem 119
```

Using the Machine Language Editor: MLX

Charles Brannon

Remember the last time you typed in the BASIC loader for a long machine language program? You typed in hundreds of numbers and commas. Even then, you couldn't be sure if you typed it in right. So you went back, proofread, tried to run the program, crashed, went back and proofread again, corrected a few typing errors, ran again, crashed again, rechecked your typing. . . . Frustrating, wasn't it?

Until now, though, that has been the best way to get machine language into your computer. Unless you happen to have an assembler and are willing to tangle with machine language on the assembly level, it is much easier to enter a BASIC program that reads DATA statements and POKEs the numbers into memory.

Some of these "BASIC loaders" use a checksum to see if you've typed the numbers correctly. The simplest checksum is just the sum of all the numbers in the DATA statements. If you make an error, your checksum does not match up with the total. Some programmers make your task easier by including checksums every few lines, so you can locate your errors more easily.

Now, MLX comes to the rescue. MLX is a great way to enter all those long machine language programs with a minimum of fuss. MLX lets you enter the numbers from a special list that looks similar to DATA statements. It checks your typing on a line-by-line basis. It won't let you enter illegal characters when you should be typing numbers. It won't let you enter numbers greater than 255 (forbidden in ML). It will prevent you from entering the numbers on the wrong line. In short, MLX makes proofreading obsolete.

Tape or Disk Copies

In addition, MLX generates a ready-to-use copy of your machine language program on tape or disk. You can then use the

LOAD command to read the program into the computer, as with any other program. Specifically, you enter:

LOAD *"program name"*,8,1(for disk)

or

LOAD *"program name"*,1,1(for tape)

To start the program, you need to enter a SYS command that transfers control from BASIC to your machine language program. The starting SYS is always listed in the article which presents the machine language program in MLX format.

Using MLX

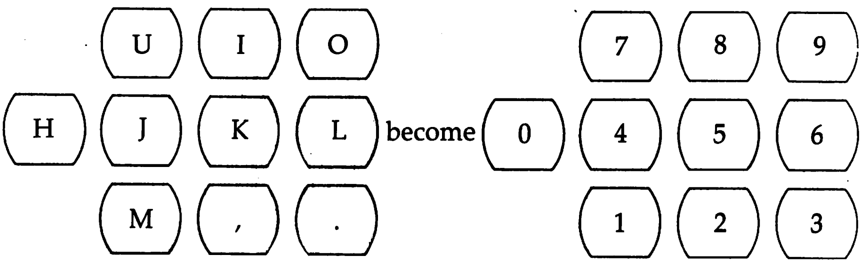
Type in and save MLX (you'll want to use it in the future). When you're ready to type in the machine language program, run MLX. MLX asks you for two numbers: the starting address and the ending address. These numbers are given in the article accompanying the ML program you're typing. For example, the addresses for "Screen-80" should be 49152 and 52811 respectively.

You'll see a prompt. The prompt is the current line you are entering from the MLX-format listing. It increases by six each time you enter a line. That's because each line has seven numbers—six actual data numbers plus a checksum number. The checksum verifies that you typed the previous six numbers correctly. If you enter any of the six numbers wrong, or enter the checksum wrong, the 64 sounds a buzzer and prompts you to reenter the line. If you enter the line correctly, a bell tone sounds and you continue to the next line.

A Special Editor

You are not using the normal 64 BASIC editor with MLX. For example, it will accept only numbers as input. If you make a typing error, press the INST/DEL key; the entire number is deleted. You can press it as many times as necessary, back to the start of the line. If you enter three-digit numbers as listed, the computer automatically prints the comma and goes on to accept the next number. If you enter less than three digits, you can press either the space bar or RETURN key to advance to the next number. The checksum automatically appears in reverse video for emphasis.

To make it even easier to enter these numbers, MLX re-defines part of the keyboard as a numeric keypad (lines 581–584).



When testing it, I've found MLX to be an extremely easy way to enter long listings. With the audio cues provided, you don't even have to look at the screen if you're a touch-typist.

Done at Last!

When you get through typing, assuming you type your machine language program all in one session, you can then save the completed and bug-free program to tape or disk. Follow the instructions displayed on the screen. If you get any error messages while saving, you probably have a bad disk, or the disk is full, or you made a typo when entering the MLX program. (Sorry, MLX can't check itself!)

Command Control

You don't have to enter the whole ML program in one sitting. MLX lets you enter as much as you want, save it, and then reload the file from tape or disk later. MLX recognizes these commands:

SHIFT-S: Save
SHIFT-L: Load
SHIFT-N: New Address
SHIFT-D: Display

Hold down SHIFT while you press the appropriate key. MLX jumps out of the line you've been typing, so I recommend you do it at a prompt. Use the Save command to store what you've been working on. It will save on tape or disk as if you've finished, but the tape or disk won't work, of course, until you finish typing. Remember what address you stopped on. The next time you run MLX, answer all the prompts as you did before, then insert the disk or tape containing the stored file. When you get the entry prompt, press SHIFT-L to reload the partly completed file into memory. Then use the New Address command (SHIFT-N) to resume typing.

New Address and Display
After you press SHIFT-N, enter the address where you previously stopped. The prompt will change, and you can then continue typing. Always enter a New Address that matches up with one of the line numbers in the special listing, or else the checksums won't match up. You can use the Display command to display a section of your typing. After you press SHIFT-D, enter two addresses within the line number range of the listing. You can abort the listing by pressing any key.

Tricky Stuff
The special commands may seem a little confusing, but as you work with MLX, they will become valuable. For example, what if you forgot where you stopped typing? Use the Display command to scan memory from the beginning to the end of the program. When you reach the end of your typing, the lines will contain a random pattern of numbers, quite different from what should be there. When you see the end of your typing, press any key to stop the listing. Use the New Address command to continue typing from the proper location.

You can use the Save and Load commands to make copies of the complete machine language program. Use the Load command to reload the tape or disk, then insert a new tape or disk and use the Save command to create a new copy. When resaving on disk, it is best to use a different filename each time you save. For example, I like to number my work and use filenames such as ASTRO1, ASTRO2, ASTRO3, and so on.

One quirk about tapes made with the MLX Save command: when you load them, the message *FOUND program* may appear twice. The tape will load just fine, however.

I think you'll find MLX to be a true labor-saving program. Since it has been tested by entering actual programs, you can count on it as an aid for generating bug-free machine language. Be sure to save MLX; it will be used for future applications in COMPUTE! books, *COMPUTE!* magazine, and *COMPUTE!'s Gazette.*

Machine Language Editor: MLX

For mistake-proof program entry, be sure to use "The Automatic Proofreader," Appendix C.

```
10 REM LINES CHANGED FROM MLX VERSION 2.00 ARE 750
   ,765,770 AND 860                               :rem 50
20 REM LINE CHANGED FROM MLX VERSION 2.01 IS 300
                                                  :rem 147
100 PRINT"{CLR}[6]";CHR$(142);CHR$(8);:POKE53281,1
    :POKE53280,1                                  :rem 67
101 POKE 788,52:REM DISABLE RUN/STOP             :rem 119
110 PRINT"{RVS}{39 SPACES}";                     :rem 176
120 PRINT"{RVS}{14 SPACES}{RIGHT}{OFF}[*]£{RVS}
    {RIGHT} {RIGHT}{2 SPACES}[*]{OFF}[*]£{RVS}£
    {RVS}{14 SPACES}";                           :rem 250
130 PRINT"{RVS}{14 SPACES}{RIGHT} [G]{RIGHT}
    {2 RIGHT} {OFF}£{RVS}£[*]{OFF}[*]{RVS}
    {14 SPACES}";                                 :rem 35
140 PRINT"{RVS}{41 SPACES}"                      :rem 120
200 PRINT"{2 DOWN}{PUR}{BLK} MACHINE LANGUAGE EDIT
    OR VERSION 2.02{5 DOWN}"                     :rem 238
210 PRINT"[5]{2 UP}STARTING ADDRESS?{8 SPACES}
    {9 LEFT}";                                   :rem 143
215 INPUTS:F=1-F:C$=CHR$(31+119*F)               :rem 166
220 IFS<256OR(S>40960ANDS<49152)ORS>53247THENGOSUB
    3000:GOTO210                                 :rem 235
225 PRINT:PRINT:PRINT                            :rem 180
230 PRINT"[5]{2 UP}ENDING ADDRESS?{8 SPACES}
    {9 LEFT}";:INPUTE:F=1-F:C$=CHR$(31+119*F)
                                                  :rem 20
240 IFE<256OR(E>40960ANDE<49152)ORE>53247THENGOSUB
    3000:GOTO230                                 :rem 183
250 IFE<STHENPRINTC$;"{RVS}ENDING < START
    {2 SPACES}":GOSUB1000:GOTO 230               :rem 176
260 PRINT:PRINT:PRINT                            :rem 179
300 PRINT"{CLR}";CHR$(14):AD=S                   :rem 56
310 A=1:PRINTRIGHT$("0000"+MID$(STR$(AD),2),5);":"
    ;                                             :rem 33
315 FORJ=ATO6                                     :rem 33
320 GOSUB570:IFN=-1THENJ=J+N:GOTO320             :rem 228
390 IFN=-211THEN 710                              :rem 62
400 IFN=-204THEN 790                              :rem 64
410 IFN=-206THENPRINT:INPUT"{DOWN}ENTER NEW ADDRES
    S";ZZ                                         :rem 44
415 IFN=-206THENIFZZ<SORZZ>ETHENPRINT"{RVS}OUT OF
    {SPACE}RANGE":GOSUB1000:GOTO410              :rem 225
417 IFN=-206THENAD=ZZ:PRINT:GOTO310              :rem 238
420 IF N<>-196 THEN 480                           :rem 133
430 PRINT:INPUT"DISPLAY:FROM";F:PRINT,"TO";:INPUTT
                                                  :rem 234
```

```
440 IFF<SORF>EORT<SORT>ETHENPRINT"AT LEAST";S;"
    {LEFT}, NOT MORE THAN";E:GOTO430            :rem 159
450 FORI=FTOTSTEP6:PRINT:PRINTRIGHT$("ØØØØ"+MID$(S
    TR$(I),2),5);":";                            :rem 3Ø
451 FORK=ØTO5:N=PEEK(I+K):PRINTRIGHT$("ØØ"+MID$(ST
    R$(N),2),3);",";                             :rem 66
46Ø GETA$:IFA$>""THENPRINT:PRINT:GOTO31Ø        :rem 25
47Ø NEXTK:PRINTCHR$(2Ø);:NEXTI:PRINT:PRINT:GOTO31Ø
                                                 :rem 5Ø
48Ø IFN<Ø THEN PRINT:GOTO31Ø                    :rem 168
49Ø A(J)=N:NEXTJ                                 :rem 199
5ØØ CKSUM=AD-INT(AD/256)*256:FORI=1TO6:CKSUM=(CKSU
    M+A(I))AND255:NEXT                           :rem 2ØØ
51Ø PRINTCHR$(18);:GOSUB57Ø:PRINTCHR$(146);:rem 94
511 IFN=-1THENA=6:GOTO315                        :rem 254
515 PRINTCHR$(2Ø):IFN=CKSUMTHEN53Ø              :rem 122
52Ø PRINT:PRINT"LINE ENTERED WRONG : RE-ENTER":PRI
    NT:GOSUB1ØØØ:GOTO31Ø                         :rem 176
53Ø GOSUB2ØØØ                                    :rem 218
54Ø FORI=1TO6:POKEAD+I-1,A(I):NEXT:POKE54272,Ø:POK
    E54273,Ø                                     :rem 227
55Ø AD=AD+6:IF AD<E THEN 31Ø                     :rem 212
56Ø GOTO 71Ø                                     :rem 1Ø8
57Ø N=Ø:Z=Ø                                      :rem 88
58Ø PRINT"{£}";                                  :rem 81
581 GETA$:IFA$=""THEN581                         :rem 95
582 AV=-(A$="M")-2*(A$=",")-3*(A$=".")-4*(A$="J")-
    5*(A$="K")-6*(A$="L")                        :rem 41
583 AV=AV-7*(A$="U")-8*(A$="I")-9*(A$="O"):IFA$="H
    "THENA$="Ø"                                  :rem 134
584 IFAV>ØTHENA$=CHR$(48+AV)                     :rem 134
585 PRINTCHR$(2Ø);:A=ASC(A$):IFA=13ORA=44ORA=32THE
    N67Ø                                         :rem 229
59Ø IFA>128THENN=-A:RETURN                       :rem 137
6ØØ IFA<>2Ø THEN 63Ø                             :rem 1Ø
61Ø GOSUB69Ø:IFI=1ANDT=44THENN=-1:PRINT"{OFF}
    {LEFT} {LEFT}";:GOTO69Ø                      :rem 62
62Ø GOTO57Ø                                      :rem 1Ø9
63Ø IFA<48ORA>57THEN58Ø                          :rem 1Ø5
64Ø PRINTA$;:N=N*1Ø+A-48                         :rem 1Ø6
65Ø IFN>255 THEN A=2Ø:GOSUB1ØØØ:GOTO6ØØ          :rem 229
66Ø Z=Z+1:IFZ<3THEN58Ø                           :rem 71
67Ø IFZ=ØTHENGOSUB1ØØØ:GOTO57Ø                   :rem 114
68Ø PRINT",";:RETURN                             :rem 24Ø
69Ø S%=PEEK(2Ø9)+256*PEEK(21Ø)+PEEK(211)        :rem 149
691 FORI=1TO3:T=PEEK(S%-I)                       :rem 67
695 IFT<>44ANDT<>58THENPOKES%-I,32:NEXT          :rem 2Ø5
```

```
700 PRINTLEFT$("{3 LEFT}",I-1);:RETURN        :rem 7
710 PRINT"{CLR}{RVS}*** SAVE ***{3 DOWN}" :rem 236
715 PRINT"{2 DOWN}(PRESS {RVS}RETURN{OFF} ALONE TO
     CANCEL SAVE){DOWN}"                 :rem 106
720 F$="":INPUT"{DOWN} FILENAME";F$:IFF$=""THENPRI
    NT:PRINT:GOTO310                      :rem 71
730 PRINT:PRINT"{2 DOWN}{RVS}T{OFF}APE OR {RVS}D
    {OFF}ISK: (T/D)"                      :rem 228
740 GETA$:IFA$<>"T"ANDA$<>"D"THEN740      :rem 36
750 DV=1-7*(A$="D"):IFDV=8THENF$="0:"+F$:OPEN15,8,
    15,"S"+F$:CLOSE15                     :rem 212
760 T$=F$:ZK=PEEK(53)+256*PEEK(54)-LEN(T$):POKE782
    ,ZK/256                               :rem 3
762 POKE781,ZK-PEEK(782)*256:POKE780,LEN(T$):SYS65
    469                                   :rem 109
763 POKE780,1:POKE781,DV:POKE782,1:SYS65466:rem 69
765 K=S:POKE254,K/256:POKE253,K-PEEK(254)*256:POKE
    780,253                               :rem 17
766 K=E+1:POKE782,K/256:POKE781,K-PEEK(782)*256:SY
    S65496                                :rem 235
770 IF(PEEK(783)AND1)OR(191ANDST)THEN780  :rem 111
775 PRINT"{DOWN}DONE.{DOWN}":GOTO310      :rem 113
780 PRINT"{DOWN}ERROR ON SAVE.{2 SPACES}TRY AGAIN.
    ":IFDV=1THEN720                       :rem 171
781 OPEN15,8,15:INPUT#15,E1$,E2$:PRINTE1$;E2$:CLOS
    E15:GOTO720                           :rem 103
790 PRINT"{CLR}{RVS}*** LOAD ***{2 DOWN}" :rem 212
795 PRINT"{2 DOWN}(PRESS {RVS}RETURN{OFF} ALONE TO
     CANCEL LOAD)"                        :rem 82
800 F$="":INPUT"{2 DOWN} FILENAME";F$:IFF$=""THENP
    RINT:GOTO310                          :rem 144
810 PRINT:PRINT"{2 DOWN}{RVS}T{OFF}APE OR {RVS}D
    {OFF}ISK: (T/D)"                      :rem 227
820 GETA$:IFA$<>"T"ANDA$<>"D"THEN820      :rem 34
830 DV=1-7*(A$="D"):IFDV=8THENF$="0:"+F$  :rem 157
840 T$=F$:ZK=PEEK(53)+256*PEEK(54)-LEN(T$):POKE782
    ,ZK/256                               :rem 2
841 POKE781,ZK-PEEK(782)*256:POKE780,LEN(T$):SYS65
    469                                   :rem 107
845 POKE780,1:POKE781,DV:POKE782,1:SYS65466:rem 70
850 POKE780,0:SYS65493                    :rem 11
860 IF(PEEK(783)AND1)OR(191ANDST)THEN870  :rem 111
865 PRINT"{DOWN}DONE.":GOTO310            :rem 96
870 PRINT"{DOWN}ERROR ON LOAD.{2 SPACES}TRY AGAIN.
    {DOWN}":IFDV=1THEN800                 :rem 172
```

```
880 OPEN15,8,15:INPUT#15,E1$,E2$:PRINTE1$;E2$:CLOS
    E15:GOTO800                                :rem 102
1000 REM BUZZER                                :rem 135
1001 POKE54296,15:POKE54277,45:POKE54278,165
                                               :rem 207
1002 POKE54276,33:POKE 54273,6:POKE54272,5 :rem 42
1003 FORT=1TO200:NEXT:POKE54276,32:POKE54273,0:POK
     E54272,0:RETURN                           :rem 202
2000 REM BELL SOUND                             :rem 78
2001 POKE54296,15:POKE54277,0:POKE54278,247
                                               :rem 152
2002 POKE 54276,17:POKE54273,40:POKE54272,0:rem 86
2003 FORT=1TO100:NEXT:POKE54276,16:RETURN   :rem 57
3000 PRINTC$;"{RVS}NOT ZERO PAGE OR ROM":GOTO1000
                                                :rem 89
```

Index

Keeping the Tradition

COMPUTE!'s Third Book of Commodore 64 continues the tradition of its best-selling predecessors in presenting a wide-ranging collection of programs and articles exclusively for the Commodore 64. Programs of the highest quality and articles that clearly illustrate both simple and complex programming techniques provide something for every 64 user.

Although many of the programs and articles were originally published in *COMPUTE!* magazine or *COMPUTE!'s Gazette,* several have been modified or even completely rewritten. Others have never before been published. Dozens of ready-to-type-in programs are included, as well as utilities that insure error-free BASIC and machine language programs.

Some of the programs and articles in *COMPUTE!'s Third Book of Commodore 64* are:

- "Screen-80," a program which turns your monitor or television set into an 80-column display.
- "Family Tree," a genealogical storage and retrieval program.
- Educational games for children, which are fun to play and which teach math and language skills at the same time.
- "Turtle Graphics Interpreter," an excellent introduction to turtle graphics, an easy way to show children how to use a computer.
- "JEP," a machine language program which allows joystick-controlled BASIC programming. COMPUTE!'s first utility designed by and for the physically handicapped.
- Arcade and adventure games which turn you into a tank gunner or a clever sleuth.
- Utilities that crunch programs, save machine language routines, or automatically load any progam.
- "64 Paintbox," which turns your 64 into an Atari-like graphics computer.
- Articles detailing efficient programming style, variable use, and easily read programs.

These are the best from COMPUTE! Publications. You'll find every program and article useful, entertaining, or informative. And all are in the tradition you've come to expect from a COMPUTE! book. You won't be disappointed.

ISBN 0-942386-72-8

COMPUTE!'s Third Book of **Commodore 64**

COMPUTE! Books