COMPUTE!'s SECOND BOOK OF COMMODORE 64

Games, applications, utilities, tutorials, and information for users of the Commodore 64® home computer. From processing words to creating dazzling graphics, the best from COMPUTE! Publications.

A **COMPUTE! Books** Publication

$12.95

COMPUTE!'s SECOND BOOK OF COMMODORE 64

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

Commodore 64 is a trademark of Commodore Electronics Limited.

The following articles were originally published in *COMPUTE!* magazine, copyright 1983, COMPUTE! Publications, Inc.:
"Working with SID" (October).
"SuperBASIC 64" (December).

The following articles were originally published in *COMPUTE!'s Gazette* magazine, copyright 1983, COMPUTE! Publications, Inc.:
"VIC/64 Mailing List" (August).
"Wordspell" (August).
"Using the Function Keys" (September).
"Merging Programs on the 64" (November).
"VIC/64 Program Lifesaver" (November).
"Martian Prisoner" (November).
"Munchmath" (November).
"Introduction to Custom Characters for VIC and 64" (November).
"How to Make Custom Characters on the 64" (November).
"Spike" (December).
"The Note Name Game" (December).
"Sprites Made Easy" (December).
"Educational Games: A Kid's View" (December)
"SpeedScript" (January 1984).

The following article was originally published in *COMPUTE!'s Gazette* magazine, copyright 1984, COMPUTE! Publications, Inc.: "How to Use Arrays" (February).

Copyright 1984, COMPUTE! Publications, Inc. All rights reserved.

Reproduction or translation of any part of this work beyond that permitted by Sections 107 and 108 of the United States Copyright Act without the permission of the copyright owner is unlawful.

Printed in the United States of America.

ISBN 0-942386-44-2

10 9 8 7 6 5 4 3 2

COMPUTE! Publications, Inc., Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is one of the ABC Publishing Companies, and is not associated with any manufacturer of personal computers. Commodore 64 is a registered trademark of Commodore Electronics Limited.

Contents

Foreword v

Chapter 1: Recreations and Applications 1

SpeedScript
Charles Brannon 3

Spike
Eric Brandon 41

Martian Prisoner
Alan Poole 60

64 Mailing List
Joseph J. Shaughnessy 66

64 Spreadsheet
Michael Tinglof 72

Chapter 2: Kid Stuff—Educational Games 85

Educational Games: A Kid's View
Kevin Dewey 87

Wordspell
Richard Herrmann 92

Munchmath
Gerald R. Anderson 96

Chapter 3: Sound 103

Working with SID
Jerry M. Jaco 105

Sound Editor 64
Daniel L. Riegal 119

SYS Sound
Michael Steed 126

The Note Name Game
Jeff Behrens 132

Chapter 4: Colors, Characters, and Motion 137

Introduction to Custom Characters for the 64
Tom R. Halfhill 139

How to Make Custom Characters on the 64
Gary Davis 146
SuperBASIC Sprite Editor
Martin C. Kees 155
Sprites Made Easy
Paul F. Schatz 166

Chapter 5: Inside Your 64 177
Using the Function Keys: A BASIC Tutorial
Charles Brannon 179
How to Use Arrays
Don Stauffer 185
Adding New Keywords to BASIC
Sheldon Leemon 194

Chapter 6: Utilities 213
SuperBASIC 64
Martin C. Kees 215
Copyfile
Gregor Larson 250
Merging Programs on the 64
John A. Winnie 253
64 Program Lifesaver
Vern Buis 256

Appendices 259
A: A Beginner's Guide to Typing In Programs 261
B: How to Type In Programs 263
C: Screen Location Table 265
D: Screen Color Memory Table 266
E: Screen Color Codes 267
F: ASCII Codes 269
G: Screen Codes 273
H: Commodore 64 Keycodes 275
I: Using the Maching Language Editor: MLX
Charles Brannon 276
J: The Automatic Proofreader
Charles Brannon 283

Foreword

Since its introduction in the fall of 1982, the Commodore 64 computer has become one of the most popular home computers available. Hundreds of thousands of beginning and experienced programmers have learned to use its remarkable graphics, sound, and game-playing capabilities. And more and more professional programmers have been writing software for the 64, giving users a wider choice of applications for their computer.

COMPUTE! Books has been active in its support of the Commodore 64. *COMPUTE!'s First Book of Commodore 64* continues to be a bestseller among computer books. Maintaining this tradition of presenting high quality programs and detailed information, *COMPUTE!'s Second Book of Commodore 64* offers a wide range of games, applications, tutorials, and utilities. Of the articles and programs which originally appeared in *COMPUTE!* magazine or *COMPUTE!'s Gazette*, many have been enhanced since first published. Other articles and programs appear here for the first time anywhere.

There's something for computer users at any level of experience in this book. If you're just starting to use custom characters, sprites, or arrays, you'll find excellent articles which will introduce you to those techniques. If you're a more experienced programmer, you'll enjoy the articles on how to create new BASIC keywords, how to best use the 64's SID chip, and how to merge, copy, or retrieve programs easily.

Do you like to write? Then "SpeedScript," a machine language word processor, is something you'll find particularly impressive. Interested in writing games on the 64? There are utilities and information here which will help you as you program. From a sprite editor to a sound editor, you'll find what you need in *Second Book of Commodore 64*.

If you like to just sit back and play arcade-quality games on your computer, then you'll find "Spike," "Martian Prisoner," and others well worth the time it takes to type them in. There's even a complete section of games written especially for children. Not only are these games fun to play, but they'll teach your children something at the same time. "Wordspell"

offers practice in spelling, while "Munchmath" makes a game out of mathematics.

No matter what your programming experience or interest, you're certain to find that *Second Book of Commodore 64* has plenty to offer.

1

Recreations and Applications

SpeedScript

Charles Brannon

"SpeedScript," is a word processing program written entirely in machine language. Fast, powerful, and easy to use, it includes almost all the major features found in professional word processor programs for personal computers. It approaches commercial-quality programs costing $50 or more. It runs on the Commodore 64, leaving a huge 45K free for text. SpeedScript will considerably amplify the utility of your computer.

A current advertising campaign extols the virtues of a ball-point pen that can erase like a pencil, dubbing it the "portable, personal word processor." It can even plot graphics. Like a word processor, the pen can edit, change, and erase. It can produce flawless hard copy. And, indeed, you can draw circles, squares, and bar graphs. But can the pen move paragraphs? Put a 100-page book on a 5¼ inch disk? Turn a rough draft into final copy with only a few changes? Can it truly edit without a trace of correction and produce formatted, double-spaced, automatically page-numbered text?

Maybe we're not being fair to the erasable pen, but it should be made clear that a word processor is more than just a computerized typewriter. Such a "word processor" would be a few lines long:

```
10 OPEN 1,4
20 INPUT A$
30 PRINT #1,A$
40 GOTO 20
```

When RUN, the program flashes the cursor and waits for a line to be typed. When you hit RETURN, the line is sent to the printer. You can move the cursor left and overstrike or use the DEL key to make changes to the line before you hit RETURN and print it out. But once it's on paper, it's committed.

Too late to make any changes.

With a true word processor, you type everything in first, then print the whole thing out. Before you print, you can make as many changes as you want. A good word processor lets you change any line, swap paragraphs, and manipulate your text in numerous other ways. You can buy such a word processing program for your 64 for $40 to more than $100, depending on the features.

Or you can type in "SpeedScript." Even if you already own a commercial word processor for your 64, you'll be pleasantly surprised. SpeedScript offers all the standard features, plus others you may not have seen before.

Entering SpeedScript

First, you'll need to type in SpeedScript. Programs 1 and 2 look long, but they are only about 4.5K, shorter than most BASIC games. The mass of numbers are machine language. Only with machine language do you get such power, speed, and compactness. Unfortunately, machine language isn't as easy to enter as a BASIC program. To aid with all the typing, we've developed MLX, the machine language editor. Be sure to read and understand Appendix I before you begin typing in SpeedScript.

Type in and SAVE the MLX program. When you are ready to enter SpeedScript, turn your machine off and on (to clear it out), then enter this line before you load MLX:

```
POKE 44,27:POKE 6912,0:NEW
```

You can then load MLX from tape or disk, and enter RUN. MLX will ask for the starting and ending addresses. The starting address is the first number in the listing, 2049. The ending address is the last number plus five, or 6860. After you enter this, follow the instructions in Appendix I to enter the listing. It takes only a few hours (you can stop, save your work, and continue typing in several sessions). No matter what your typing speed is, rest assured that it will be well worth your effort.

Getting Started

After you enter SpeedScript with MLX, you can just LOAD it like a BASIC program. As a matter of fact, you can make copies of it with the SAVE command, as usual (SAVE "SPEEDSCRIPT" or SAVE "SPEEDSCRIPT",8 for disk). After

you LOAD, enter RUN.

The screen will be light gray or white with black (or dark gray) lettering. The top line of the screen is highlighted.

The blinking cursor shows you where text will appear when you begin typing. You cannot type on the top line of the screen. This is the command window, and is used by SpeedScript to ask questions and display messages. When a message is displayed, it will remain until you begin typing again.

To get started, just begin typing. If a word you're typing won't fit on the screen line, the word and the cursor are moved to the next line. This is called word wrap, or parsing. It makes your text much easier to read on the screen, as words are never split across the margin. Another thing to notice is that a back-arrow appears if you press RETURN. This marks the end of a paragraph or line. It is not necessary to press RETURN at the end of each screen line, as you must do when reaching the end of a line on a typewriter.

Most of us, being human, are not infallible, so you may need to correct your typing mistakes. This is a big advantage of a word processor. You fix your errors before you print, so there's no messy fluids or special ribbons. (Did you ever have to manually erase on a typewriter?—ugh!)

If you want to backspace, press the INST/DEL key in the unSHIFTed position. The cursor backs up and erases the last letter you typed. You can press it as many times as necessary to back up to the error, then retype the rest of the sentence.

But this is clearly not the best way to do things. Instead, you can move the cursor nondestructively. The cursor control keys are in the lower-right corner of the keyboard (see Figure 1: Keyboard Map). The CRSR left/right key moves the cursor to the right, and when SHIFTed moves the cursor left. Before you can correct the error, you have to move the cursor to the word in question. For example, to correct this line:

```
Now is the rime for all good men
```

The cursor is moved to the "r" (cursor-left 21 times):

```
Now is the rime for all good men
```

The letter "t" is typed:

```
Now is the time for all good men
```

And the cursor is moved to the end:

```
Now is the time for all good men
```

Resume typing:

```
Now is the time for all good men to
come to the aid of they're country.
```

Another error! We typed "they're" instead of "their." No problem.

In the above example, of course, you don't have to press the cursor-left key 21 times. You can just hold down the cursor-left key. It will repeat, and keep moving until you let go.

English Cursor Controls

You can also move the cursor in ways that make sense in plain English. For example, if you hold down SHIFT and press the f1 function key, (which is how you get f2), the cursor jumps back to the previous word. To correct the error in the first example above, just press f2 five times. You can then press f1 five times to go back to the end of the sentence and resume typing. Here is a list of what the function keys do:

f1: Move cursor to next word.
f2: Move cursor to previous word.
f3: Move cursor to start of next sentence.
f4: Move cursor to start of previous sentence.
f5: Move cursor to start of next paragraph.
f6: Move cursor to start of previous paragraph.

SpeedScript recognizes a sentence by the ending punctuation (. or ? or !), or by a RETURN mark (back-arrow). A paragraph is any sequence of characters that ends in a RETURN mark. (Refer to Figure 2, after the program listing, for a clip-out function key overlay.)

Since you're working with English, the cursor up-down keys do not move up or down exactly one screen line. Instead, they act like f3 and f4. Cursor-down moves to the next sentence, and cursor-up moves to the previous sentence. This is easier to understand for many people, but it takes some getting used to for others.

As you begin to move the cursor around, you'll notice that you cannot move the cursor past the end of text. There is an invisible marker, sometimes called End Of File (EOF) at the

end of the document. You can add text to the end of your document, but you cannot move past it, since there's nothing there. Very rarely, you may see some text past the end of file, but you can't move to it, so ignore it.

Many of the other keys behave predictably. The CLR/HOME key in the unSHIFTed position moves the cursor to the top of the screen. If you press it twice, it brings you to the top of your document (in case the document is longer than one screen). The insert key (SHIFT-INST/DEL) inserts a space at the cursor position. You can press it as many times as necessary to make space for inserting a word. You can also go into insert mode, where every letter you type is automatically inserted. In insert mode, it is not possible to overstrike. You enter or leave insert mode by pressing CTRL-I.

Normally when you type a key, that letter or symbol appears. Certain keys, such as CLR/HOME, however, perform a function. SpeedScript extends this idea and places all the command keys in an easy-to-remember order. For example, insert mode is turned on or off by pressing CTRL-I. (To use a control key, hold down CTRL while you type the other key.)

When you enter insert mode, the command window changes color to remind you. If you press CTRL-I again, you're back in normal overstrike mode, and the command window reverts to its usual color.

CTRL-Z moves you to the bottom of your document (end of file). It's useful for adding text to the end. If you want to check how much memory you have left for typing, press CTRL and the equals (=) key. You have about 45K of text memory on the 64. SpeedScript takes advantage of all the available RAM on the 64.

To accommodate personal taste and video clarity, you can change the screen and text colors to any combination you want. CTRL-B (think "background") changes the screen color. You can keep pressing it until a color you like comes up. CTRL-L ("letters") changes the text color. If you have a color monitor, you can get some really interesting combinations.

The RUN/STOP key is like a TAB key. It inserts five spaces at the cursor position. You can use it for indenting, or to add indentation to a paragraph previously typed.

If you want to change the case of a letter or word, position the cursor on the letter and press CTRL-A. It will switch from lower- to uppercase or vice versa. CTRL-A moves the

cursor to the right, so you can hold it down to change more than one letter. Another handy command is CTRL-X, or Transpose. It will switch two adjacent letters. My most common typing mistake is to wsitch (switch) two letters while I'm typing fast. With CTRL-X, it's easy to exchange the two letters without overstriking (which is useful in insert mode).

Text Deletion

With a typewriter, if you don't like what you've typed, you can tear the paper out, crumple it up, and dunk it into "file 13." With a word processor, this satisfying act is accomplished with but a few keystrokes.

With the DEL key, you can erase the last letter typed. If you're in the middle of text and press it, you'll notice that the character the cursor is sitting on is pulled on top of the previous character, and the rest of the text follows along. It sounds a little confusing, but it's easy:

```
The quick brown fox juunmped over
```

Cursor is moved to error:

```
The quick brown fox juun[m]ped over
```

DEL is struck twice, deleting the erroneous characters:

```
The quick brown fox juu[m]ped over
The quick brown fox ju[m]ped over
```

If you don't want the text to be pulled back, you can press the back-arrow key. It will just backspace and blank out the previous character without pulling the adjacent characters backward. Another way to delete is with CTRL-back-arrow. The cursor does not move, but the following text is "sucked into" the cursor. It is like a tiny black hole.

If you want to strike out a whole word, sentence, or paragraph, it's time for a more drastic command: CTRL-E. When you press CTRL-E, the command window turns red (to instill fear and awe). You see the message:

```
Erase (S,W,P): RETURN to exit
```

Each time you press one of the three keys, a sentence, word, or paragraph is pulled toward the cursor and deleted. You can keep pressing S, W, or P until all the text you want to

remove is gone. Then press RETURN to exit the Erase function and resume writing. Erase will remove text to the *right* of the cursor. If you are at the end of a sentence, word, or paragraph, you can use Delete (CTRL-D) to erase *backward.* CTRL-D displays:

```
Delete (S,W,P)
```

and immediately returns to the normal mode after its work is done. CTRL-Delete is like the DEL key, and CTRL-Erase is like CTRL-back-arrow.

What if you press one key too many in the Erase command? What if you change your mind? Oh, no! What if you accidentally erase the wrong paragraph? On most word processors, you're out of luck. But with SpeedScript, you can retrieve the crumpled-up piece of paper and "uncrumple" it. Within certain limitations, SpeedScript remembers and stores the text you Erase or Delete. If you change your mind, just press CTRL-R.

Here's how it works. When you Erase text, the text is moved from the main screen into a *failsafe buffer,* a reserved area of memory. SpeedScript reserves 12K for the failsafe buffer.

There's another valuable use for the buffer, too. You can move text by putting it in the buffer and recalling it at the destination. Just Erase the paragraphs, words, or sentences you want to move, then place the cursor where you want to insert the text and press CTRL-R (think "Restore," "Retrieve," or "Recall"). In a flash, the text is inserted. If you want to copy (rather than move) a word, sentence, or paragraph, you can restore the deleted text with CTRL-R, then move the cursor and press CTRL-R to insert the deleted text again. You can retrieve the buffer contents as often as you like. For example, if you use a long word or phrase often, just type it once, Erase it, then use CTRL-R to have the computer type it out for you.

You should be aware that CTRL-E and CTRL-D will clear the previous buffer contents. When you move one paragraph, then go back to move another, you don't want to have both paragraphs merged together the second time. Also, if CTRL-Delete added text to the buffer instead of replacing the buffer, CTRL-R would insert the text entries out of order, since CTRL-D deletes "backward."

If you want to move two paragraphs at the same time

instead of separately, you can override the replacement and cause CTRL-Erase to add to the end of the buffer. Just hold down SHIFT with CTRL as you press E. If you want to force the buffer to be cleared, you can use CTRL-K (Kill) to clear the buffer. If you try to delete more than the length of the buffer (12K on the 64), you'll see "Buffer Full". Stop and move the text, or use CTRL-K to clear the buffer to erase some more.

Finally, if you really want to wipe out all your text, there is a way. (Beware: You cannot recover from a total clear.) Press SHIFT-CLR/HOME. You will see:

```
ERASE ALL TEXT: Are you sure? (Y/N)
```

If you really want to erase all the text, press Y. Any other key, including N, will return you to your text unharmed. You should use this command only when you want to start a new document, as it is one of the few ways to lose text beyond recovery.

Search Feature

When you are lost in the middle of a big document and want to find a particular word or phrase, the Hunt command comes in handy. Press CTRL-H and you'll see:

```
Hunt for:
```

Enter the word or phrase you want to find, then press RETURN. SpeedScript will locate the word and place the cursor on it, scrolling if necessary. If the phrase is not found, you'll see a "Not Found" message in the command window.

The first time you use Hunt, SpeedScript will search for the phrase from the top of the document. Pressing CTRL-H again will find the next occurrence of the search phrase after the cursor position. You can search for a new phrase without waiting to get "Not Found" for the previous phrase by holding down SHIFT while you press CTRL-H.

There are some tricks to using Hunt. For example, if you search for the word "if," SpeedScript will match it with the embedded "if" in a word like "specific." Should you just want to find the word "if," search for "if" followed by a space. Also, searching for "if" will not match with the capitalized "If."

Saving and Loading

What makes a word processor truly great is that you can save your typing to tape or disk. Say you're writing a term paper. You type it in and save it to disk. Your teacher returns the rough draft with suggested corrections. Without retyping the entire paper, you just load the original, make some changes, and print it out. A 5¼ inch disk can hold more writing than a briefcase! You can also write in stages: save your work as you go along, then come back to it at another time. Saving and loading alone elevates word processing far above any other means of writing.

To save your work, press f8 (SHIFT-f7). You will see:

```
Save:
```

Enter the name you want to use for the document. Follow the standard Commodore filename rules, such as keeping the name to 16 characters or less. Press RETURN, then press either T or D, answering the prompt TAPE OR DISK?

After the Save is completed, you'll see NO ERRORS (hopefully). If there was an error during the save, such as no disk in the drive, or a disk full error, SpeedScript will read the error channel and display the error message. You'll get the error "file exists" if you try to save using a name that's already on the disk. If you want to replace the file, prefix the name with the characters "@:", such as "@:Document". This is called "Save with Replace." You can also press CTRL-↑ (up arrow, explained below) and scratch the file before you save.

Press f7 to load a file. You may want to use SHIFT-CLR/HOME to erase the current text first. The Load feature will append text starting wherever the cursor is positioned. This lets you merge several files from tape or disk into memory. If the cursor is not at the top of the file, the command window will change color to warn you that you are performing an append. You should add text only to the end of the file, as the end-of-file marker is put wherever the load stops. Also, beware that you can crash SpeedScript if you try to load a file and don't have enough room (a file longer than available memory). You can use CTRL-= (equals sign) to check the available memory space before merging files to avoid a crash.

You can use CTRL-V to Verify a saved file. Verify works like Load, but compares the file with what's in memory. It's most useful with tape, but you can use it with disk files, too.

SpeedScript files appear on the directory as PRG, program files. The documents certainly aren't programs, but since the operating system has convenient Save and Load routines, the text files are just dumped from memory. This is also more reliable for tape. You can load files created on some other word processors, such as *WordPro* or *PaperClip*, but you may have to do some reformatting. If the upper- and lowercase come out reversed, you can hold down CTRL-A to transform the entire file.

Other Disk Commands

Use CTRL-4 (think CTRL-$, as in LOAD"$",8 from BASIC) to look at the disk directory. You will not lose whatever text you have in memory. While the directory is being printed on the screen, you can press CTRL to slow down the printing, or the space bar to freeze the listing (press the space bar again to continue).

You can send any other disk command with CTRL-↑ (up-arrow). It may not seem easy to remember, but I think of the arrow as pointing to the disk drive. The command window shows a greater-than sign (>). Type in the disk command and press RETURN. By referring to your disk drive manual, you can do anything the commands permit, such as Initialize, New, Copy, Rename, Scratch, etc. If you press RETURN without entering a disk command, SpeedScript displays the disk error message (if any). (Table 1, near the end of this article, is a clip-out reference card for all the editing commands.)

PRINT!

At last, we get to the whole point of word processing—the printout. Actually, you can use SpeedScript without a printer. If you and a friend each have a copy of SpeedScript, you can exchange letters on tape or disk, ready to load and view. You can get a lot of text on one tape or disk. And if you have a friend with a printer and a 64, you can bring SpeedScript and your files.

Before your text can be printed, it must be formatted. The text must be broken into lines with margins, and there has to be a way to divide the output into pages. For those with pinfeed paper, we also need to skip over the perforation. Of course, it would be nice to be able to automatically number all pages. And why not let the computer center lines for you, or block them edge right? You should be able to change the left

and right margin anytime, as well as line spacing. Headers and footers at the top and bottom of each page would add a really nice touch.

SpeedScript does all that and more. But with that power comes the responsibility to learn more commands. These commands do not act directly on the text, but control how the text is printed out. Some commands do things like change the left margin, while others let you do things with the text like centering or underlining. Remember, the formatting commands will not change how the text on the screen looks. They affect only the hard copy (what's on paper).

Thanks to several default settings, you can print right away without using any printer commands. If you press CTRL-P, SpeedScript will make several assumptions and begin to print. A few of these assumptions are: left margin of five spaces, right margin at 75 (meaning a line length of 70 characters), and double spacing. If you want to change these settings, you'll need to use the formatting commands.

Entering Format Commands

The format commands are single letters or characters that appear on the screen in reverse video. To get a reverse video letter, press CTRL and the English pound sign (next to the CLR/HOME key). The command window will prompt "Key:". Now press one of the format letters, such as "r" for right margin, or "c" for center. That letter will appear in reverse video (within a "box," with its colors switched). SpeedScript recognizes only lowercase letters and some symbols as commands.

Changing Printer Variables

The printer variables are values such as left margin, right margin, line spacing, top and bottom margins, and so on. They are called variables because they can change. For example, to quote a passage within your text, you may indent it by increasing the left margin, and also change to single spacing to set it apart. You would then want to switch back to normal margins and double spacing for the rest of the paper.

To change a printer variable, just follow the reverse video letter with a number. Do not leave a space between a letter and a number. You can put the format commands anywhere in text, though I prefer to group them together on a line of their own. Here is an example setting:

```
[l10[r60[s1[t10[b50←
```

To set off these format commands, I'll show here that they are in reverse video by enclosing them in brackets. You'll enter them with CTRL-English pound sign.

[s] Spacing, default 2. Line spacing. This is set to 2 to designate double spacing. For single spacing, enter 1, for triple spacing, enter 3, and so on.

[l] Left margin, default 5. The left margin is the number of spaces to indent for each line.

[r] Right margin, default 75. This must be a number less than 80, which is the number of characters that can fit on a line. Add the line length you want to the left margin to get the right margin.

[t] Top margin, default 5. How many blank lines to skip from the top of the page to the first line of printing. Should be at least 5.

[b] Bottom margin, default 58. A number less than 66, which is the number of lines on an 8½ inch x 11 inch sheet of paper or pinfeed paper. Do not use a bottom margin more than 58

[h] Define header. The header is printed at the top of each page, if you specify one. To define the header, begin a line with [h], enter the header text, then press RETURN. Example:

```
[hAccounting Procedures←
```

You can embed a format [c] after the [h] to center the header, a format [e] to block the header edge right, and a format [#] any place you want a page number to appear. Examples:

A centered page number with a dash on each side:

```
[hcPage -[#-←
```

The header used when this article was written:

```
[heBrannon/"SpeedScript/[#←
```

[f] Define footer. Just like header, but appears at the bottom of each page. A centered page number within parentheses:

```
[fc(#)←
```

[n] Next page. This command forces the printer to skip to the next page, regardless of the position on the current page.

Other Commands

These commands do not change printer variables, so they are usually embedded within a line.

[u] Underline—place on each side of a word or phrase to underline. It works by backspacing and overstriking an underline symbol on top of each character. Some printers, including the VIC 1525, do not support the backspace command, so underlining will not work on these printers.

[c] Center—place this at the start of a line you wish to center. Remember to end the line with RETURN.

[e] Edge right—like center, but will block the line to the edge of the right margin.

[#] Page number—When SpeedScript encounters this symbol, it prints the current page number.

User-Definable Codes

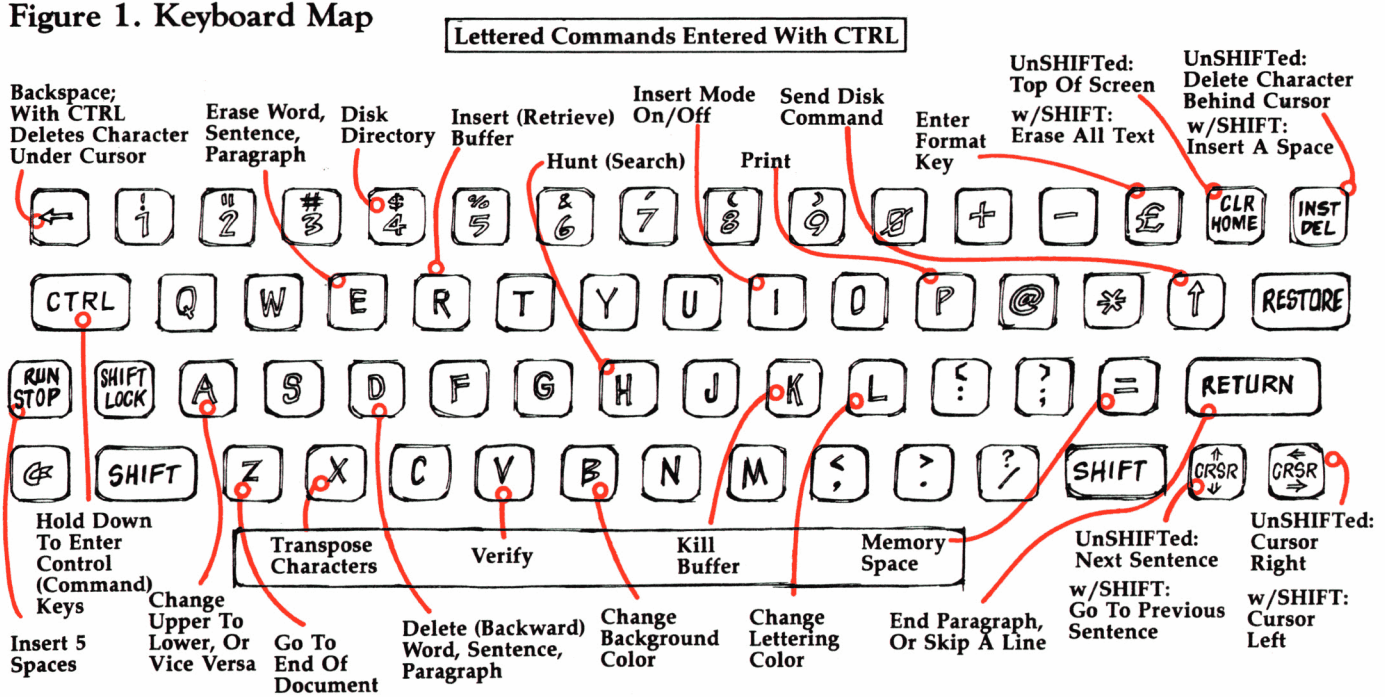
Many printers use special so-called escape sequences to control printer functions such as automatic underlining, boldface, italics, super/subscripting, elongated, condensed, etc. These codes are either ASCII numbers less than 32 (control codes) or are triggered by an ESCape character, CHR$(27), followed by a letter or symbol. For example, for the Epson MX-80 with Graftrax, italics is turned on with ESC 4. You should study your manuals to learn how to use these codes. Since most of the control codes and the escape character are not available from the keyboard, SpeedScript lets you define the format commands 1–9.

If you enter [1]=65, then every time the reverse video [1] is encountered during printing, that character (65 is the letter A in ASCII) is sent to the printer. For example, SpeedScript uses the back-arrow for a carriage return mark, so you can't directly cause a back-arrow to print on the printer. Instead, you can look up the ASCII value of the back-arrow, which is 95. You would enter [1]=95, say, at the top of your document. Then, any place you want to print a back-arrow, just embed a [1] in your text. Refer to Appendix F, "ASCII Codes", for the ASCII values of the 64's characters and graphics symbols. The first four numbers are predefined so that you don't have to set them, but you can change their definition:

[1]=27 (escape), [2]=14 (elongated, most printers), [3]=15 (elongated off), [4]=18 (condensed).

A fascinating possibility is to trigger the bit graphics

Figure 1. Keyboard Map



capability of your printer. For example, you could define special characters. On the VIC 1525, you could send a graphic box (for a checklist perhaps) with:

```
1=82=153=2554=193
13444444432 Toothpaste
```

This would appear on the printer as:

```
□ Toothpaste
```

Printer Compatibility

SpeedScript works best, of course, with a standard Commodore printer. However, we have used it with several other printers such as the Epson MX-80, an Okidata Microline 82A, and the Leading Edge Prowriter (NEC 8023), via an appropriate interface. The interfaces I've used are the Cardco Card/Print and the Tymac Connection. Any interface that works through the Commodore serial port should be fine. SpeedScript will probably not work with an RS-232 printer attached to the modem/user port. SpeedScript may operate with some interfaces which emulate a Centronics port on the user port via software, as long as the software does not conflict with SpeedScript. If you can get your printer to work fine with CTRL-P, skip the next few paragraphs to avoid confusion.

The Commodore printers and most interfaces use a device number of 4. (Other device numbers are 1 for the tape drive and 8 for the disk drive). If you have more than one printer attached with different device numbers, you can enter this number by holding down SHIFT while you press CTRL-P. You'll be asked to enter the device number and the secondary address. Incidentally, you can get a rough idea of page breaks before printing by using a device number of 3, which causes output to go to the screen.

The secondary address is a command number for the printer. For Commodore printers or interfaces which emulate the Commodore printer, the secondary address should be 7, which signifies lowercase mode. The default device number, 4, and the default secondary address, 7, are automatic when you press CTRL-P without holding down SHIFT.

If your interface cannot even partially emulate a Commodore printer, you will have a few problems. First of all, the numbers Commodore uses to describe characters, called

PETASCII by some, do not correspond with standard ASCII, which most non-Commodore printers use. The result is usually that upper- and lowercase come out switched. SpeedScript lets you get around this if you place a format [a] at the top of your file.

You also need to use the [a] if you want to bypass the emulation offered by the interface. You may do this to be able to activate your printer's special function codes which are often intercepted and interpreted by the interface. You will also have to use a different secondary address. I'll have to bow out and suggest you scrutinize both your printer's manual and that of the interface.

Pinfeed Versus Single Sheet

The pinfeed or tractor feed is the cheapest and most common paper delivery system for printers. Some printers, however, have a platen like a typewriter and can accept single sheets of paper, such as stationery or company letterhead paper. Normally, SpeedScript prints continuously, skipping over the perforation that divides continuous pinfeed paper.

If you are using single sheets of paper, you need SpeedScript to stop at the end of each page, tell you to insert a new sheet, then continue. If you place a reverse video [w] (for Wait) at the top of your file (again, use CTRL-English pound sign to do this), SpeedScript will do just that. When you get to the end of the page, insert a new sheet, then press RETURN to continue printing.

Table 2, after the program listing, provides a quick-reference card for all formatting commands.

As you can tell, SpeedScript is a truly comprehensive word processor. Although it's ultimately easy to use, it may take you a while to master all the features and variations. I hope your adventure will prove to be fascinating and fruitful.

SpeedScript

```
2049 :011,008,010,000,158,050,238
2055 :048,054,049,000,000,000,158
2061 :032,103,009,076,193,009,179
2067 :165,251,141,051,008,165,032
2073 :252,141,052,008,165,253,128
2079 :141,054,008,165,254,141,026
2085 :055,008,166,181,240,032,207
```

```
2091 :169,000,141,195,026,160,222
2097 :000,185,014,039,153,013,197
2103 :039,200,204,195,026,208,159
2109 :244,238,052,008,238,055,128
2115 :008,224,000,240,007,202,236
2121 :208,224,165,180,208,222,000
2127 :096,165,181,170,005,180,108
2133 :208,001,096,024,138,101,141
2139 :252,141,123,008,165,251,007
2145 :141,122,008,024,138,101,119
2151 :254,141,126,008,165,253,026
2157 :141,125,008,232,164,180,191
2163 :208,004,240,013,160,255,227
2169 :185,000,028,153,001,028,004
2175 :136,192,255,208,245,206,089
2181 :123,008,206,126,008,202,038
2187 :208,234,096,169,040,133,251
2193 :195,133,020,169,004,133,031
2199 :196,169,216,133,021,173,035
2205 :191,026,133,155,173,192,003
2211 :026,133,156,162,001,173,046
2217 :194,026,133,012,173,204,143
2223 :026,141,032,208,160,000,230
2229 :173,203,026,145,020,177,157
2235 :155,153,205,026,200,041,199
2241 :127,201,031,240,019,192,235
2247 :040,208,235,136,177,155,126
2253 :041,127,201,032,240,005,083
2259 :136,208,245,160,039,200,175
2265 :132,167,136,185,205,026,044
2271 :145,195,136,016,248,164,103
2277 :167,024,152,101,155,133,193
2283 :155,165,156,105,000,133,181
2289 :156,152,157,060,003,192,193
2295 :040,240,008,169,032,145,113
2301 :195,200,076,246,008,024,234
2307 :165,195,105,040,133,195,068
2313 :133,020,144,004,230,196,224
2319 :230,021,232,224,025,240,219
2325 :003,076,179,008,165,155,095
2331 :141,201,026,165,156,141,089
2337 :202,026,096,169,000,133,147
2343 :155,141,191,026,141,197,122
2349 :026,133,038,169,029,133,061
2355 :156,141,192,026,141,198,137
2361 :026,133,039,169,032,162,106
2367 :179,160,255,198,156,145,132
2373 :155,200,230,156,145,155,086
2379 :200,208,251,230,156,202,042
2385 :208,246,145,155,096,133,040
```

```
2391 :167,132,168,160,000,177,123
2397 :167,240,006,032,210,255,235
2403 :200,208,246,096,169,012,006
2409 :141,204,026,169,038,133,048
2415 :001,169,011,141,203,026,150
2421 :032,036,009,169,000,141,248
2427 :194,026,032,115,015,169,162
2433 :255,141,138,002,032,245,174
2439 :012,032,150,009,169,109,104
2445 :160,025,032,086,009,238,179
2451 :193,026,096,032,166,009,157
2457 :169,090,160,025,032,086,203
2463 :009,169,000,141,193,026,185
2469 :096,162,039,169,032,157,052
2475 :000,004,202,016,250,169,044
2481 :019,076,210,255,072,041,082
2487 :128,074,133,167,104,041,062
2493 :063,005,167,096,160,000,168
2499 :177,038,133,002,160,000,193
2505 :177,038,073,128,145,038,032
2511 :032,142,008,032,228,255,136
2517 :208,013,165,162,041,016,050
2523 :240,245,169,000,133,162,144
2529 :076,199,009,170,160,000,071
2535 :165,002,145,038,224,095,132
2541 :208,012,032,160,011,169,061
2547 :032,160,000,145,038,076,182
2553 :193,009,173,193,026,240,059
2559 :007,138,072,032,150,009,151
2565 :104,170,138,201,013,208,071
2571 :002,162,095,138,041,127,064
2577 :201,032,144,070,224,160,080
2583 :208,002,162,032,138,072,125
2589 :173,194,026,240,003,032,185
2595 :140,014,104,032,181,009,003
2601 :160,000,145,038,032,142,046
2607 :008,056,165,038,237,197,236
2613 :026,133,167,165,039,237,052
2619 :198,026,005,167,144,014,101
2625 :165,038,105,000,141,197,199
2631 :026,165,039,105,000,141,035
2637 :198,026,230,038,208,002,011
2643 :230,039,032,231,010,076,189
2649 :193,009,138,174,125,010,226
2655 :221,125,010,240,006,202,131
2661 :208,248,076,193,009,202,013
2667 :138,010,170,169,009,072,163
2673 :169,192,072,189,162,010,139
2679 :072,189,161,010,072,096,207
2685 :035,029,157,137,133,002,106
```

```
2691 :012,138,134,020,148,004,075
2697 :019,009,147,135,139,005,079
2703 :136,140,022,145,017,159,250
2709 :018,024,026,016,028,030,035
2715 :006,001,011,008,031,003,215
2721 :150,011,159,011,170,011,161
2727 :227,011,054,012,066,012,037
2733 :080,012,179,012,231,013,188
2739 :139,014,014,014,083,014,201
2745 :201,014,225,014,253,014,138
2751 :024,015,185,015,222,017,157
2757 :205,016,043,018,080,012,059
2763 :179,012,111,018,118,019,148
2769 :023,020,028,012,108,020,164
2775 :186,017,112,023,002,014,057
2781 :039,020,244,012,215,023,006
2787 :057,025,122,014,032,071,036
2793 :011,056,165,038,237,191,163
2799 :026,133,167,165,039,237,238
2805 :192,026,005,167,176,030,073
2811 :056,173,191,026,233,000,162
2817 :133,167,173,192,026,233,157
2823 :029,005,167,240,013,165,114
2829 :038,141,191,026,165,039,101
2835 :141,192,026,032,142,008,048
2841 :056,173,201,026,229,038,236
2847 :133,155,173,202,026,229,181
2853 :039,133,156,005,155,240,253
2859 :002,176,024,024,173,191,121
2865 :026,109,061,003,141,191,068
2871 :026,173,192,026,105,000,065
2877 :141,192,026,032,142,008,090
2883 :076,025,011,096,056,173,248
2889 :197,026,233,000,133,167,061
2895 :173,198,026,233,207,005,153
2901 :167,144,010,169,000,141,204
2907 :197,026,169,207,141,198,005
2913 :026,056,165,038,233,000,103
2919 :133,167,165,039,233,029,101
2925 :005,167,176,009,169,000,123
2931 :133,038,169,029,133,039,144
2937 :096,056,165,038,237,197,142
2943 :026,133,167,165,039,237,126
2949 :198,026,005,167,176,001,194
2955 :096,173,197,026,133,038,034
2961 :173,198,026,133,039,096,042
2967 :230,038,208,002,230,039,130
2973 :076,231,010,165,038,208,117
2979 :002,198,039,198,038,076,202
```

```
2985 :231,010,165,038,133,155,133
2991 :165,039,133,156,198,156,254
2997 :160,255,177,155,201,032,137
3003 :240,004,201,031,208,003,106
3009 :136,208,243,177,155,201,033
3015 :032,240,008,201,031,240,183
3021 :004,136,208,243,096,132,000
3027 :167,056,165,155,101,167,254
3033 :133,038,165,156,105,000,046
3039 :133,039,076,231,010,160,104
3045 :000,177,038,201,032,240,149
3051 :008,201,031,240,004,200,151
3057 :208,243,096,200,240,025,229
3063 :177,038,201,032,240,247,158
3069 :201,031,240,243,024,152,120
3075 :101,038,133,038,165,039,005
3081 :105,000,133,039,076,231,081
3087 :010,173,197,026,133,038,080
3093 :173,198,026,133,039,076,154
3099 :013,012,169,000,141,191,041
3105 :026,173,198,026,056,233,233
3111 :004,201,029,176,002,169,108
3117 :029,141,192,026,032,142,095
3123 :008,076,016,012,238,204,093
3129 :026,173,204,026,041,015,030
3135 :141,204,026,096,238,203,203
3141 :026,173,203,026,041,015,041
3147 :141,203,026,076,142,008,159
3153 :165,038,133,155,165,039,008
3159 :133,156,198,156,160,255,121
3165 :177,155,201,046,240,012,156
3171 :201,033,240,008,201,063,077
3177 :240,004,201,031,208,004,025
3183 :136,208,235,096,177,155,094
3189 :201,046,240,026,201,033,096
3195 :240,022,201,063,240,018,139
3201 :201,031,240,014,136,208,191
3207 :235,198,156,165,156,201,222
3213 :000,176,227,076,169,012,033
3219 :132,167,198,167,200,240,227
3225 :010,177,155,201,032,240,200
3231 :247,136,076,210,011,164,235
3237 :167,076,115,012,169,000,192
3243 :133,038,169,029,133,039,200
3249 :076,231,010,160,000,177,063
3255 :038,201,046,240,029,201,170
3261 :033,240,025,201,063,240,223
3267 :021,201,031,240,017,200,137
3273 :208,235,230,039,165,039,093
3279 :205,198,026,240,226,144,222
```

```
3285 :224,076,016,012,200,240,213
3291 :250,177,038,201,032,240,133
3297 :247,201,046,240,243,201,123
3303 :033,240,239,201,063,240,223
3309 :235,201,031,240,231,076,227
3315 :001,012,169,000,141,059,113
3321 :028,169,208,141,060,028,115
3327 :032,166,009,169,129,160,152
3333 :025,032,086,009,169,001,071
3339 :141,193,026,096,056,165,176
3345 :038,233,000,133,167,165,241
3351 :039,233,029,005,167,208,192
3357 :003,104,104,096,165,038,027
3363 :133,251,165,039,133,252,240
3369 :096,056,165,038,133,253,014
3375 :073,255,101,251,141,063,163
3381 :028,165,039,133,254,073,233
3387 :255,101,252,141,064,028,132
3393 :165,251,141,065,028,165,112
3399 :252,141,066,028,165,253,208
3405 :141,067,028,133,251,165,094
3411 :254,141,068,028,133,252,191
3417 :024,173,064,028,109,060,035
3423 :028,201,255,144,020,032,007
3429 :166,009,169,144,160,025,006
3435 :032,086,009,169,001,141,033
3441 :193,026,169,000,133,198,064
3447 :096,173,059,028,133,253,093
3453 :173,060,028,133,254,173,178
3459 :063,028,133,180,024,109,156
3465 :059,028,141,059,028,173,113
3471 :064,028,133,181,109,060,206
3477 :028,141,060,028,169,000,063
3483 :141,026,208,169,032,133,096
3489 :001,032,019,008,169,038,172
3495 :133,001,169,001,141,026,126
3501 :208,173,065,028,133,251,007
3507 :173,066,028,133,252,173,236
3513 :067,028,133,253,173,068,139
3519 :028,133,254,056,173,197,008
3525 :026,229,253,133,180,173,167
3531 :198,026,229,254,133,181,200
3537 :032,019,008,056,173,197,182
3543 :026,237,063,028,141,197,139
3549 :026,173,198,026,237,064,177
3555 :028,141,198,026,096,032,236
3561 :015,013,032,160,011,032,240
3567 :042,013,056,173,059,028,098
3573 :233,001,141,059,028,173,112
3579 :060,028,233,000,141,060,005
```

```
3585 :028,096,032,151,011,032,095
3591 :015,013,032,160,011,076,058
3597 :042,013,032,245,012,169,014
3603 :002,133,012,032,166,009,117
3609 :169,156,160,025,032,086,141
3615 :009,032,228,255,240,251,022
3621 :072,032,150,009,104,041,189
3627 :191,201,023,208,009,032,195
3633 :015,013,032,171,011,076,111
3639 :042,013,201,019,208,009,035
3645 :032,015,013,032,081,012,246
3651 :076,042,013,201,016,208,111
3657 :009,032,015,013,032,025,199
3663 :015,076,042,013,096,056,121
3669 :165,038,237,191,026,133,107
3675 :167,165,039,237,192,026,149
3681 :005,167,240,011,173,191,116
3687 :026,133,038,173,192,026,179
3693 :133,039,096,169,000,133,167
3699 :038,169,029,133,039,076,087
3705 :231,010,160,005,140,085,240
3711 :028,032,140,014,172,085,086
3717 :028,136,208,244,076,228,029
3723 :011,024,165,038,133,251,249
3729 :105,001,133,253,165,039,073
3735 :133,252,105,000,133,254,004
3741 :056,173,197,026,229,253,067
3747 :133,180,173,198,026,229,078
3753 :254,133,181,201,255,208,121
3759 :006,169,001,133,180,230,126
3765 :181,032,080,008,160,000,130
3771 :169,032,145,038,238,197,238
3777 :026,208,003,238,198,026,124
3783 :076,013,012,173,194,026,181
3789 :073,014,141,194,026,096,237
3795 :169,171,160,025,032,086,086
3801 :009,032,228,255,240,251,208
3807 :201,089,096,169,002,133,145
3813 :012,032,166,009,169,194,043
3819 :160,025,032,086,009,032,067
3825 :211,014,240,003,076,150,167
3831 :009,162,255,154,076,013,148
3837 :008,160,000,177,038,201,069
3843 :031,240,015,200,208,247,176
3849 :230,039,165,039,205,198,117
3855 :026,144,238,076,016,012,015
3861 :200,076,001,012,165,038,001
3867 :133,155,165,039,133,156,040
3873 :198,156,160,255,177,155,110
```

```
3879 :201,031,240,016,136,192,087
3885 :255,208,245,198,156,165,248
3891 :156,201,029,176,237,076,158
3897 :169,012,056,152,101,155,190
3903 :133,155,169,000,101,156,009
3909 :133,156,056,165,155,229,195
3915 :038,133,167,165,156,229,195
3921 :039,005,167,208,018,132,138
3927 :167,024,165,155,229,167,226
3933 :133,155,165,156,233,000,167
3939 :133,156,076,043,015,165,175
3945 :155,133,038,165,156,133,117
3951 :039,076,231,010,120,169,244
3957 :127,141,013,220,169,027,046
3963 :141,017,208,169,146,141,177
3969 :020,003,169,015,141,021,242
3975 :003,169,001,141,026,208,171
3981 :141,018,208,088,096,169,093
3987 :058,164,012,205,018,208,044
3993 :208,005,169,001,172,204,144
3999 :026,140,033,208,141,018,213
4005 :208,201,001,240,008,169,224
4011 :001,141,025,208,076,188,042
4017 :254,169,001,141,025,208,207
4023 :076,049,234,173,141,002,090
4029 :041,001,208,003,032,245,207
4035 :012,032,166,009,169,209,024
4041 :160,025,032,086,009,160,161
4047 :000,177,038,073,128,145,000
4053 :038,032,142,008,160,000,081
4059 :177,038,073,128,145,038,050
4065 :169,002,133,012,032,228,033
4071 :255,240,251,009,064,201,227
4077 :087,208,009,032,022,016,099
4083 :032,228,011,076,037,016,131
4089 :201,083,208,009,032,022,036
4095 :016,032,180,012,076,037,096
4101 :016,201,080,208,009,032,039
4107 :022,016,032,254,014,076,169
4113 :037,016,076,150,009,165,214
4119 :038,133,253,141,054,027,157
4125 :165,039,133,254,141,055,048
4131 :027,096,056,165,038,133,038
4137 :251,237,054,027,141,063,046
4143 :028,165,039,133,252,237,133
4149 :055,027,141,064,028,032,144
4155 :065,013,173,054,027,133,012
4161 :038,173,055,027,133,039,018
4167 :032,142,008,076,206,015,038
4173 :169,038,229,211,141,199,040
```

```
4179 :026,169,000,141,088,028,023
4185 :160,000,169,156,032,210,048
4191 :255,169,018,032,210,255,010
4197 :169,032,032,210,255,169,200
4203 :157,032,210,255,140,200,077
4209 :026,032,228,255,240,251,121
4215 :172,200,026,133,167,169,218
4221 :146,032,210,255,169,032,201
4227 :032,210,255,169,157,032,218
4233 :210,255,169,155,032,210,144
4239 :255,165,167,201,013,240,160
4245 :046,201,020,208,015,136,007
4251 :016,004,200,076,091,016,046
4257 :169,157,032,210,255,076,036
4263 :091,016,041,127,201,032,163
4269 :144,172,204,199,026,240,134
4275 :167,165,167,153,245,026,078
4281 :032,210,255,169,000,133,216
4287 :212,200,076,091,016,032,050
4293 :210,255,169,000,153,245,205
4299 :026,152,096,032,166,009,172
4305 :169,246,160,025,032,086,159
4311 :009,032,051,017,176,031,019
4317 :169,000,133,155,169,029,108
4323 :133,156,174,197,026,172,061
4329 :198,026,169,155,032,216,005
4335 :255,176,010,032,183,255,126
4341 :041,191,208,003,076,028,024
4347 :018,240,039,173,050,017,020
4353 :201,008,144,006,032,174,054
4359 :023,076,028,017,173,050,118
4365 :017,201,001,240,249,032,241
4371 :166,009,169,252,160,025,032
4377 :032,086,009,032,115,015,058
4383 :169,001,141,193,026,096,145
4389 :032,166,009,169,007,160,068
4395 :026,032,086,009,076,028,044
4401 :017,008,032,077,016,240,183
4407 :024,169,038,160,026,032,248
4413 :086,009,032,228,255,240,143
4419 :251,162,008,201,068,240,229
4425 :012,162,001,201,084,240,005
4431 :006,032,150,009,104,104,228
4437 :096,142,050,017,169,001,048
4443 :160,000,032,186,255,160,116
4449 :000,224,001,240,042,185,021
4455 :245,026,201,064,208,007,086
4461 :185,246,026,201,058,240,041
4467 :028,169,048,141,029,027,045
4473 :169,058,141,030,027,185,219
```

```
4479 :245,026,153,031,027,200,041
4485 :204,200,026,144,244,240,167
4491 :242,200,076,156,017,185,247
4497 :245,026,153,029,027,200,057
4503 :204,200,026,208,244,140,149
4509 :053,027,032,166,009,169,101
4515 :245,160,026,032,086,009,209
4521 :173,053,027,162,029,160,005
4527 :027,032,189,255,169,013,092
4533 :032,210,255,076,086,018,090
4539 :032,166,009,169,241,160,196
4545 :025,032,086,009,032,228,093
4551 :255,240,251,032,181,009,143
4557 :009,128,072,173,194,026,039
4563 :240,003,032,140,014,032,160
4569 :150,009,104,076,041,010,095
4575 :056,165,038,233,000,133,080
4581 :167,165,039,233,029,005,099
4587 :167,240,004,169,005,133,185
4593 :012,032,166,009,169,058,175
4599 :160,026,032,086,009,032,080
4605 :051,017,165,012,201,005,192
4611 :240,003,032,036,009,169,236
4617 :000,166,038,164,039,032,192
4623 :213,255,144,003,076,252,190
4629 :016,142,197,026,140,198,228
4635 :026,032,231,255,032,166,001
4641 :009,169,028,160,026,032,201
4647 :086,009,076,028,017,032,031
4653 :166,009,169,064,160,026,127
4659 :032,086,009,032,051,017,022
4665 :169,001,162,000,160,029,066
4671 :032,213,255,032,183,255,009
4677 :041,191,240,211,032,166,182
4683 :009,169,015,160,026,032,230
4689 :086,009,076,028,017,120,161
4695 :169,000,141,026,208,141,004
4701 :033,208,169,049,141,020,201
4707 :003,169,234,141,021,003,158
4713 :169,255,141,013,220,088,223
4719 :096,169,147,032,210,255,252
4725 :169,013,032,210,255,032,060
4731 :086,018,032,162,018,169,096
4737 :013,032,210,255,169,072,112
4743 :160,026,032,086,009,032,224
4749 :228,255,201,013,208,249,015
4755 :032,115,015,076,150,009,032
4761 :032,204,255,169,001,032,078
4767 :195,255,096,032,231,255,199
4773 :169,001,162,008,160,000,153
```

```
4779 :032,186,255,169,002,162,209
4785 :087,160,026,032,189,255,158
4791 :032,192,255,176,221,162,197
4797 :001,032,198,255,032,207,146
4803 :255,032,207,255,032,207,159
4809 :255,032,183,255,208,202,056
4815 :032,207,255,240,197,032,146
4821 :204,255,032,228,255,201,108
4827 :032,208,005,032,228,255,211
4833 :240,251,162,001,032,198,085
4839 :255,032,207,255,072,032,060
4845 :207,255,168,104,170,152,013
4851 :160,055,132,001,032,205,060
4857 :189,160,054,132,001,169,186
4863 :032,032,210,255,032,207,255
4869 :255,240,006,032,210,255,235
4875 :076,003,019,169,013,032,067
4881 :210,255,076,199,018,162,169
4887 :000,142,056,027,142,057,191
4893 :027,142,058,027,056,177,004
4899 :155,233,048,144,042,201,090
4905 :010,176,038,014,056,027,106
4911 :046,057,027,014,056,027,018
4917 :046,057,027,014,056,027,024
4923 :046,057,027,014,056,027,030
4929 :046,057,027,013,056,027,035
4935 :141,056,027,200,208,212,147
4941 :230,156,076,033,019,248,071
4947 :173,056,027,013,057,027,180
4953 :240,023,056,173,056,027,152
4959 :233,001,141,056,027,173,214
4965 :057,027,233,000,141,057,104
4971 :027,238,058,027,076,083,104
4977 :019,173,058,027,216,096,190
4983 :056,173,059,028,233,000,156
4989 :141,061,028,173,060,028,104
4995 :233,208,141,062,028,013,048
5001 :061,028,208,016,032,166,136
5007 :009,169,097,160,026,032,124
5013 :086,009,169,001,141,193,236
5019 :026,096,024,165,038,133,125
5025 :251,109,061,028,133,253,228
5031 :165,039,133,252,109,062,159
5037 :028,133,254,056,173,197,246
5043 :026,229,251,133,180,173,147
5049 :198,026,229,252,133,181,180
5055 :024,101,254,201,207,144,098
5061 :016,032,166,009,169,089,166
5067 :160,026,032,086,009,169,173
5073 :001,141,193,026,096,032,186
```

```
5079 :080,008,024,173,061,028,077
5085 :133,180,109,197,026,141,239
5091 :197,026,173,062,028,133,078
5097 :181,109,198,026,141,198,062
5103 :026,165,038,133,253,165,251
5109 :039,133,254,169,000,133,205
5115 :251,169,208,133,252,169,153
5121 :000,141,026,208,169,032,065
5127 :133,001,032,019,008,169,113
5133 :038,133,001,169,001,141,240
5139 :026,208,076,231,010,160,218
5145 :000,177,038,170,200,177,019
5151 :038,136,145,038,200,138,214
5157 :145,038,096,160,000,177,141
5163 :038,041,063,240,010,201,124
5169 :027,176,006,177,038,073,034
5175 :064,145,038,076,151,011,028
5181 :133,167,041,064,010,005,225
5187 :167,041,191,133,167,041,039
5193 :032,073,032,010,005,167,136
5199 :096,005,075,066,005,058,128
5205 :002,001,027,255,015,018,147
5211 :000,000,000,000,000,032,123
5217 :166,009,169,141,160,026,000
5223 :076,086,009,076,129,021,244
5229 :169,004,141,089,028,160,188
5235 :007,173,141,002,041,001,224
5241 :240,054,032,166,009,169,023
5247 :116,160,026,032,086,009,044
5253 :032,228,255,240,251,056,171
5259 :233,048,201,003,144,217,217
5265 :201,008,176,213,141,089,205
5271 :028,032,166,009,169,125,168
5277 :160,026,032,086,009,032,246
5283 :228,255,240,251,056,233,146
5289 :048,048,190,201,010,176,074
5295 :186,168,169,001,174,089,194
5301 :028,032,186,255,169,000,083
5307 :032,189,255,032,096,020,043
5313 :032,192,255,162,001,032,099
5319 :201,255,144,003,076,129,239
5325 :021,169,000,133,155,169,084
5331 :029,133,156,162,000,142,065
5337 :070,028,142,069,028,142,184
5343 :090,028,142,091,028,142,232
5349 :083,028,189,080,020,157,018
5355 :075,028,232,224,007,208,241
5361 :245,169,255,141,084,028,139
5367 :160,000,177,155,016,003,246
5373 :076,068,022,201,031,240,123
```

```
5379 :034,153,059,027,200,238,202
5385 :082,028,173,082,028,205,095
5391 :076,028,144,230,136,140,001
5397 :196,026,177,155,201,032,040
5403 :240,009,206,082,028,136,216
5409 :208,244,172,196,026,140,251
5415 :196,026,152,056,101,155,213
5421 :133,155,165,156,105,000,247
5427 :133,156,160,000,173,084,245
5433 :028,201,255,208,003,032,016
5439 :236,021,032,033,022,173,068
5445 :196,026,141,195,026,169,054
5451 :059,133,169,169,027,133,253
5457 :170,032,109,024,032,048,240
5463 :022,173,084,028,205,079,166
5469 :028,144,003,032,148,021,213
5475 :056,165,155,237,197,026,167
5481 :133,167,165,156,237,198,137
5487 :026,005,167,240,029,144,210
5493 :027,169,000,141,069,028,039
5499 :141,078,028,032,148,021,059
5505 :032,225,255,240,251,169,021
5511 :001,032,195,255,032,204,086
5517 :255,076,150,009,076,247,186
5523 :020,056,173,077,028,237,226
5529 :084,028,168,136,136,240,177
5535 :010,048,008,169,013,032,183
5541 :210,255,136,208,248,173,115
5547 :070,028,240,019,141,195,096
5553 :026,173,073,028,133,169,011
5559 :173,074,028,133,170,032,025
5565 :033,022,032,109,024,169,066
5571 :013,032,210,255,032,210,179
5577 :255,032,210,255,173,081,183
5583 :028,208,026,032,204,255,192
5589 :032,166,009,169,150,160,131
5595 :026,032,086,009,032,228,120
5601 :255,240,251,032,096,020,095
5607 :162,001,032,201,255,238,096
5613 :083,028,173,069,028,240,090
5619 :019,141,195,026,173,071,100
5625 :028,133,169,173,072,028,084
5631 :133,170,032,033,022,032,165
5637 :109,024,169,013,032,210,050
5643 :255,172,078,028,140,084,000
5649 :028,136,136,240,010,048,103
5655 :008,169,013,032,210,255,198
5661 :136,208,248,096,169,032,150
5667 :172,075,028,140,082,028,048
5673 :032,210,255,136,208,250,108
```

```
5679 :096,172,080,028,024,152,087
5685 :109,084,028,141,084,028,015
5691 :169,013,032,210,255,136,106
5697 :208,250,096,141,087,028,107
5703 :041,127,032,061,020,201,041
5709 :049,144,007,201,058,176,200
5715 :003,076,080,023,174,158,085
5721 :022,221,158,022,240,012,252
5727 :202,208,248,206,082,028,045
5733 :173,087,028,076,000,021,230
5739 :202,138,010,170,140,085,084
5745 :028,169,022,072,169,128,189
5751 :072,189,170,022,072,189,065
5757 :169,022,072,096,056,173,201
5763 :085,028,101,155,133,155,020
5769 :165,156,105,000,133,156,084
5775 :076,247,020,200,177,155,250
5781 :201,031,240,001,136,140,130
5787 :085,028,096,010,087,065,014
5793 :076,082,084,066,083,078,118
5799 :072,070,188,022,197,022,226
5805 :205,022,215,022,225,022,116
5811 :235,022,245,022,255,022,212
5817 :014,023,051,023,169,000,209
5823 :141,081,028,200,076,146,095
5829 :022,169,001,141,090,028,136
5835 :076,146,022,200,032,022,189
5841 :019,141,075,028,076,146,182
5847 :022,200,032,022,019,141,139
5853 :076,028,076,146,022,200,001
5859 :032,022,019,141,078,028,035
5865 :076,146,022,200,032,022,219
5871 :019,141,079,028,076,146,216
5877 :022,200,032,022,019,141,169
5883 :080,028,076,146,022,140,231
5889 :086,028,032,148,021,172,232
5895 :086,028,140,085,028,076,194
5901 :146,022,056,152,101,155,133
5907 :141,071,028,165,156,105,173
5913 :000,141,072,028,032,043,085
5919 :023,056,152,237,085,028,100
5925 :141,069,028,076,146,022,007
5931 :200,177,155,201,031,208,247
5937 :249,136,096,056,152,101,071
5943 :155,141,073,028,165,156,005
5949 :105,000,141,074,028,032,185
5955 :043,023,056,152,237,085,151
5961 :028,141,070,028,076,146,050
5967 :022,200,177,155,201,061,127
5973 :240,004,136,076,101,022,152
```

```
5979 :200,032,022,019,072,173,097
5985 :087,028,041,015,170,202,128
5991 :104,157,087,020,032,146,137
5997 :022,076,129,022,032,231,109
6003 :255,169,000,032,189,255,247
6009 :169,015,162,008,160,015,138
6015 :032,186,255,032,192,255,055
6021 :144,001,096,032,166,009,069
6027 :169,062,032,210,255,032,131
6033 :077,016,240,025,162,015,168
6039 :032,201,255,176,012,169,228
6045 :245,160,026,032,086,009,203
6051 :169,013,032,210,255,032,106
6057 :231,255,076,150,009,032,154
6063 :231,255,169,000,032,189,027
6069 :255,169,015,162,008,160,182
6075 :015,032,186,255,032,192,131
6081 :255,176,228,032,166,009,035
6087 :162,015,032,198,255,032,125
6093 :077,016,032,231,255,169,217
6099 :001,141,193,026,096,173,073
6105 :141,002,201,005,240,005,043
6111 :173,088,028,208,037,032,021
6117 :166,009,169,171,160,026,162
6123 :032,086,009,032,077,016,231
6129 :208,003,076,150,009,169,088
6135 :001,141,088,028,141,193,071
6141 :026,169,000,133,155,169,137
6147 :029,133,156,076,022,024,187
6153 :165,038,133,155,165,039,192
6159 :133,156,160,001,076,024,053
6165 :024,160,000,162,000,189,044
6171 :245,026,032,181,009,209,217
6177 :155,240,002,162,255,200,023
6183 :208,011,230,156,165,156,197
6189 :205,198,026,240,002,176,124
6195 :035,232,236,200,026,208,220
6201 :224,024,152,101,155,133,078
6207 :038,165,156,105,000,133,148
6213 :039,056,165,038,237,200,036
6219 :026,133,038,165,039,233,197
6225 :000,133,039,076,231,010,058
6231 :032,166,009,169,181,160,036
6237 :026,032,086,009,169,001,160
6243 :141,193,026,169,000,141,001
6249 :088,028,096,096,160,000,061
6255 :204,195,026,240,248,177,177
```

```
6261 :169,048,038,032,061,020,229
6267 :032,026,025,032,210,255,191
6273 :173,091,028,240,010,169,072
6279 :008,032,210,255,169,095,136
6285 :032,210,255,032,225,255,126
6291 :208,005,104,104,076,129,005
6297 :021,200,076,111,024,140,213
6303 :085,028,041,127,032,061,021
6309 :020,201,049,144,017,201,029
6315 :058,176,013,041,015,170,132
6321 :202,189,087,020,032,210,149
6327 :255,076,154,024,201,067,192
6333 :208,026,056,169,080,237,197
6339 :195,026,074,056,237,075,090
6345 :028,168,169,032,032,210,072
6351 :255,136,208,250,172,085,033
6357 :028,076,154,024,201,069,253
6363 :208,017,056,173,076,028,009
6369 :237,195,026,056,237,075,027
6375 :028,168,169,032,076,205,141
6381 :024,201,085,208,008,173,168
6387 :091,028,073,001,141,091,156
6393 :028,201,035,240,003,076,064
6399 :154,024,140,085,028,174,092
6405 :083,028,169,000,160,055,244
6411 :132,001,032,205,189,160,218
6417 :054,132,001,172,085,028,233
6423 :076,154,024,174,090,028,057
6429 :240,026,133,167,041,127,251
6435 :201,065,144,018,201,091,243
6441 :176,014,170,165,167,041,006
6447 :128,073,128,074,074,133,145
6453 :167,138,005,167,096,032,146
6459 :166,009,056,169,000,237,184
6465 :197,026,170,169,207,237,047
6471 :198,026,160,055,132,001,131
6477 :032,205,189,160,054,132,081
6483 :001,169,001,141,193,026,102
6489 :096,014,008,155,211,080,141
6495 :069,069,068,211,067,082,149
6501 :073,080,084,032,049,046,209
6507 :049,000,032,066,089,032,119
6513 :195,072,065,082,076,069,160
6519 :083,032,194,082,065,078,141
6525 :078,079,078,000,194,085,127
6531 :070,070,069,082,032,195,137
6537 :076,069,065,082,069,068,054
```

```
6543 :000,194,085,070,070,069,119
6549 :082,032,198,085,076,076,186
6555 :000,196,069,076,069,084,137
6561 :069,032,040,211,044,215,004
6567 :044,208,041,000,058,032,038
6573 :193,082,069,032,089,079,205
6579 :085,032,083,085,082,069,103
6585 :063,032,040,217,047,206,022
6591 :041,058,000,197,210,193,122
6597 :211,197,032,193,204,204,214
6603 :032,212,197,216,212,000,048
6609 :197,082,065,083,069,032,225
6615 :040,211,044,215,044,208,209
6621 :041,058,032,018,210,197,009
6627 :212,213,210,206,146,032,222
6633 :084,079,032,069,088,073,146
6639 :084,000,203,069,089,058,230
6645 :000,211,065,086,069,058,222
6651 :000,212,065,080,069,032,197
6657 :197,210,210,207,210,000,011
6663 :211,084,079,080,080,069,098
6669 :068,000,214,069,082,073,007
6675 :070,089,032,197,082,082,059
6681 :079,082,000,206,079,032,247
6687 :069,082,082,079,082,083,252
6693 :000,147,032,018,212,146,080
6699 :065,080,069,032,079,082,194
6705 :032,018,196,146,073,083,085
6711 :075,063,000,204,079,065,029
6717 :068,058,000,214,069,082,040
6723 :073,070,089,058,000,208,053
6729 :082,069,083,083,032,018,184
6735 :210,197,212,213,210,206,047
6741 :146,000,036,048,206,079,088
6747 :032,210,079,079,077,000,056
6753 :206,079,032,084,069,088,143
6759 :084,032,073,078,032,066,212
6765 :085,070,070,069,082,046,019
6771 :000,196,069,086,073,067,094
6777 :069,032,035,000,211,069,025
6783 :067,079,078,068,046,032,241
6789 :193,068,068,082,046,032,110
6795 :035,000,208,082,073,078,103
6801 :084,073,078,071,000,206,145
6807 :069,088,084,032,083,072,067
```

```
6813 :069,069,084,044,032,018,217
6819 :210,197,212,213,210,206,131
6825 :146,000,200,085,078,084,250
6831 :032,070,079,082,058,000,240
6837 :206,079,084,032,198,079,091
6843 :085,078,068,000,000,029,191
6849 :000,000,004,104,005,029,079
6855 :036,013,192,032,011,012,239
```

Table 1. Clip-Out Quick Reference Card—Editing Commands

CTRL-A: Change case
CTRL-B: Change background color
CTRL-D: Delete
CTRL-E: Erase
CTRL-H: Hunt
CTRL-I: Insert Mode
CTRL-K: Clear buffer
CTRL-L: Change lettering color
CTRL-P: Print
CTRL-R: Recall buffer
CTRL-V: Verify
CTRL-X: Transpose characters
CTRL-Z: End of document
CTRL-4: Disk directory
CTRL-↑: Send DOS command
CTRL-£: Enter format key
CTRL-=: Free memory
f1: Next word
f2: Previous word
f3: Previous sentence
f5: Next paragraph
f6: Previous paragraph
f7: Load
f8: Save
Cursor Up: Previous sentence
Cursor Down: Next Sentence
Cursor Left/Right: As implied
SHIFT-CLR/HOME: Erase all
CLR/HOME: Top of screen/top of document
Back-arrow: Backspace
CTRL-Back-arrow: Delete character
RUN/STOP: Insert 5 spaces

Table 2. Clip-Out Quick Reference Card—Format Commands

Format commands in column one are entered with CTRL-£.

Cmd	Description	Default
l	left margin	5
r	right margin	75
t	top margin	5
b	bottom margin	58
h	define header	none
f	define footer	none
w	wait for next sheet	no wait
a	true ASCII	
u	underline toggle	
c	center line	
e	edge right	
s	line spacing	2
n	go to next page	
#	page number	
1–9	user-definable keys (see text)	

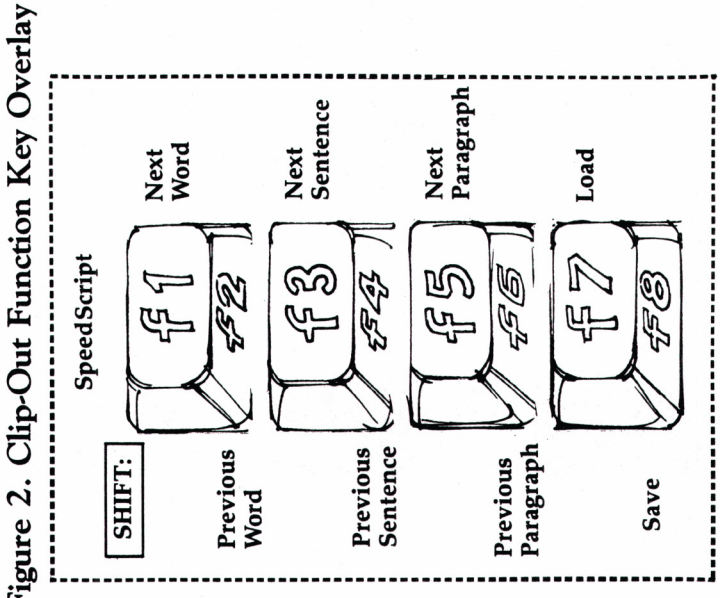


Figure 2. Clip-Out Function Key Overlay

SPIKE

Eric Brandon

An all-machine-language game, "Spike" pits you against deadly power spikes on the Grid *as you search for your hidden Commodore 64. The game is fast-paced, and approaches commercial-quality software—a game you might expect to pay $30 or more for.*

It is a dark and stormy night, and you are diligently typing games into your Commodore 64.

Suddenly, just outside, you see a dazzling flash of light and almost at once hear the deafening retort of thunder. The lights dim, flicker, and wink out. A wave of dizziness overcomes you.

When you regain consciousness, you cannot recognize your surroundings. "This isn't my computer room," you think. A thousand theories about your situation fly through your head, but none is even close to the terrible truth.

You are trapped inside the Power Grid.

To return to your own world, you must find and encircle your Commodore 64 computer. It's not visible from where you are, but you know it is hidden inside one of the many grid nodes. Fortunately, you are carrying your pocket sonar, which always tells you how far from the 64 you are. The shorter the line displayed by your sonar, the closer you are to escaping.

You soon discover that the Grid is a dangerous place to be. Deadly power spikes travel up and down the wires. Touching one of the spikes results in a terrible shock. These shocks, though powerful, are very short, so you can endure up to four collisions with the spikes and still stand a chance to make it home.

Unfortunately, should you successfully reach your 64, you will find that the magnetic disturbance which trapped you on the Grid in the first place is worse than ever. You end up on the Grid again, but now it is coursed by even more power spikes.

Is there no escape?

Playing Spike

The recommended way to travel on the Power Grid is with a joystick in port two. The joystick may seem a bit awkward at first: Since the Grid is tilted 45 degrees, the four cardinal directions (up, down, left, right) are likewise tilted. To better orient yourself, it may help to turn the joystick base to the same angle.

When "Spike" first starts, you will have to make some decisions. You must decide the speed of the game and whether you want the Easy or Hard option. Pressing the RETURN key or the joystick button automatically chooses the Hard option and a speed of 5. If you want some other option, press the number of the speed you want (1 to 9) and the *E* key for an Easy game.

Another handy feature of Spike is the pause option. Pressing a SHIFT key pauses the action. Pressing SHIFT/LOCK freezes the game until SHIFT/LOCK is released.

You start each game with five lives. An indicator at the top of the screen, labeled STAMINA, keeps track of your remaining lives, not counting the one currently in play.

Another indicator, SONAR, shows your proximity to your invisible goal, the hidden Commodore 64 computer. The shorter the line, the closer you are to the 64.

The LEVEL indicator displays flags to show how many times you've found the 64 and advanced to a more difficult power grid.

When you start a new game, the Grid is patrolled by two power spikes. Another spike joins them on each succeeding level, up to a maxmium of seven spikes.

To develop a winning strategy, it's vital to understand how the scoring works. The screen is divided into 112 grid nodes (diamond-shaped blocks). Your goal, the Commodore 64, is hidden in one of them, leaving 111 empty nodes. You gain survival points for traversing the Grid—ten points for each new side of a node you cross. If you box in a node by leaving your trail along all four of its sides, the node is colored blue. You'll want to box in as few nodes as possible, because it costs you bonus points later.

When you find the Commodore 64 by locating it with your sonar and encircling its node, you win bonus points and advance to the next level. The bonus is figured by multiplying the number of unboxed nodes times the bonus value for the

current level. The bonus value starts at 40 for level one and increases by five for each additional level. For instance, if you find the 64 on level three after boxing in 11 nodes, you would win 5000 bonus points (100 unboxed nodes x bonus value of 50 = 5000). This would be added to the survival points you gained while searching the Grid.

A HIGH SCORE indicator keeps track of the best game played during the current sitting.

Typing Spike

Unavoidably, Spike is a long program—more than 4K of pure machine language. Normally, it is very difficult to type in such a program without making a mistake. Also, in the past, a machine language monitor was necessary to enter such a program from a published listing in a book or magazine.

However, to make the typing as easy and as foolproof as possible, "MLX," a machine language entry program, was written by Program Editor Charles Brannon to greatly simplify the task of typing ML programs from listings. It includes an instant checksum feature which does not let you continue until you've typed a line correctly. It also automatically types commas and lets you break up the job into several sittings.

Please read the directions in Appendix I for using MLX. And be sure to save MLX, because it will be needed for other machine language programs in this book.

You'll need to type in a POKE statement in direct mode (without a line number) before you begin entering Spike. This line will move down the top of memory to below the Spike program so that BASIC will not write over Spike as you type it in.

```
POKE 52,128:POKE 56,128:CLR
```

You'll use this statement *only* while you enter Spike using MLX. You *don't* need it when you enter other machine language programs.

This line must be entered *before* you load and run MLX. Then you can begin typing in the Spike program. If you enter Spike in several sessions, turning the computer off in between, you must type in the above POKE statement *each* time you begin entering Spike's data.

Here is the information you'll need to enter Spike with MLX:

Starting address—32768
Ending address—37295

Once Spike is saved on disk or tape, a special procedure is required to load the program.

For disk, enter:

```
LOAD"SPIKE",8,1
```

For tape, enter:

```
LOAD"",1,1
```

When the program is loaded, run it by entering SYS 32768.

Spike

```
32768 :169,005,141,190,207,169,113
32774 :072,141,180,207,032,019,145
32780 :144,169,007,141,201,207,113
32786 :169,040,141,200,207,169,176
32792 :012,141,199,207,169,000,240
32798 :141,039,208,162,024,157,249
32804 :000,212,202,224,255,208,113
32810 :248,169,070,141,254,207,107
32816 :169,120,141,253,207,169,083
32822 :255,141,015,212,141,182,232
32828 :207,169,128,141,018,212,167
32834 :169,064,141,136,002,169,235
32840 :001,141,246,207,169,019,087
32846 :032,210,255,169,000,141,117
32852 :032,208,173,014,220,041,004
32858 :254,141,014,220,165,001,117
32864 :041,251,133,001,160,000,170
32870 :185,000,208,153,000,080,216
32876 :185,000,209,153,000,081,224
32882 :185,000,210,153,000,082,232
32888 :185,000,211,153,000,083,240
32894 :185,000,212,153,000,084,248
32900 :185,000,213,153,000,085,000
32906 :185,000,214,153,000,086,008
32912 :185,000,215,153,000,087,016
32918 :200,208,205,165,001,009,170
32924 :004,133,001,173,014,220,189
32930 :009,001,141,014,220,169,204
32936 :198,141,000,221,169,008,137
32942 :141,024,208,032,183,128,122
32948 :076,219,128,120,169,127,251
32954 :141,013,220,169,001,141,103
32960 :026,208,169,000,141,018,242
```

```
32966 :208,173,017,208,041,127,204
32972 :141,017,208,169,119,141,231
32978 :020,003,169,140,141,021,192
32984 :003,088,096,032,225,128,020
32990 :076,249,128,169,089,133,042
32996 :252,160,000,133,251,169,169
33002 :000,145,251,200,208,251,009
33008 :230,252,166,252,224,128,212
33014 :208,243,096,169,016,160,114
33020 :000,153,000,064,153,000,110
33026 :065,153,000,066,153,000,183
33032 :067,200,208,241,169,022,147
33038 :141,248,067,169,006,153,030
33044 :000,216,153,000,217,153,247
33050 :000,218,153,000,219,200,048
33056 :208,241,032,182,137,032,096
33062 :043,129,076,187,129,032,122
33068 :133,139,169,001,133,002,109
33074 :169,050,141,255,207,172,020
33080 :255,207,162,000,032,239,183
33086 :139,232,224,151,240,005,029
33092 :136,192,030,208,243,173,026
33098 :255,207,024,105,020,141,058
33104 :255,207,201,200,144,225,032
33110 :169,010,141,255,207,174,018
33116 :255,207,160,200,032,239,161
33122 :139,136,232,224,151,208,164
33128 :247,173,255,207,024,105,091
33134 :020,141,255,207,201,151,061
33140 :144,229,169,190,141,255,220
33146 :207,172,255,207,162,000,101
33152 :032,239,139,232,224,151,121
33158 :240,005,200,192,200,208,155
33164 :243,173,255,207,056,233,027
33170 :020,141,255,207,201,022,224
33176 :176,225,169,020,141,255,114
33182 :207,174,255,207,160,030,167
33188 :032,239,139,200,232,224,206
33194 :151,208,247,173,255,207,131
33200 :024,105,020,141,255,207,160
33206 :201,151,144,229,096,169,148
33212 :096,133,252,169,032,133,235
33218 :254,160,000,133,251,133,101
33224 :253,177,251,145,253,200,199
33230 :208,249,230,252,230,254,093
33236 :166,252,224,127,208,239,148
33242 :177,251,145,253,200,192,156
33248 :064,208,247,032,155,139,045
33254 :032,166,135,032,145,143,115
```

```
33260 :169,007,141,021,208,173,187
33266 :030,208,076,212,140,173,057
33272 :000,220,141,252,207,041,085
33278 :001,208,043,032,030,139,195
33284 :240,003,032,186,138,173,008
33290 :253,207,201,030,208,003,144
33296 :076,173,130,173,254,207,005
33302 :201,150,208,003,076,173,065
33308 :130,238,254,207,206,253,036
33314 :207,173,252,207,141,249,239
33320 :207,076,173,130,173,252,027
33326 :207,041,002,208,037,032,061
33332 :030,139,240,003,032,186,170
33338 :138,173,253,207,201,200,206
33344 :240,107,173,254,207,201,222
33350 :000,240,100,238,253,207,084
33356 :206,254,207,173,252,207,095
33362 :141,249,207,076,173,130,034
33368 :173,252,207,041,004,208,205
33374 :037,032,050,139,240,003,083
33380 :032,186,138,173,253,207,065
33386 :201,030,240,063,173,254,043
33392 :207,201,000,240,056,206,254
33398 :254,207,206,253,207,173,138
33404 :252,207,141,249,207,076,232
33410 :173,130,173,252,207,041,082
33416 :008,208,034,032,050,139,095
33422 :240,003,032,186,138,173,146
33428 :253,207,201,200,240,019,244
33434 :173,254,207,201,150,240,099
33440 :012,238,254,207,238,253,082
33446 :207,173,252,207,141,249,115
33452 :207,032,155,139,162,255,098
33458 :160,000,200,208,253,232,207
33464 :208,250,032,024,136,032,098
33470 :144,136,032,036,137,032,195
33476 :029,135,032,030,139,208,001
33482 :008,032,050,139,208,003,130
33488 :032,181,133,032,217,130,165
33494 :076,247,129,173,030,208,053
33500 :041,001,208,001,096,206,005
33506 :199,207,104,104,032,200,048
33512 :143,174,199,207,232,169,076
33518 :032,157,040,064,076,212,051
33524 :140,173,241,207,010,141,132
33530 :207,207,176,008,169,000,249
33536 :141,206,207,076,011,131,004
33542 :169,001,141,206,207,173,135
33548 :207,207,024,105,013,141,197
```

```
33554 :014,208,173,206,207,105,163
33560 :000,024,106,106,141,206,095
33566 :207,173,016,208,041,127,034
33572 :013,206,207,141,016,208,059
33578 :173,240,207,024,105,041,064
33584 :141,015,208,169,001,141,211
33590 :046,208,169,023,141,255,128
33596 :067,162,254,154,173,021,123
33602 :208,141,205,207,169,129,101
33608 :141,021,208,032,081,143,186
33614 :032,081,143,032,081,143,078
33620 :169,000,141,202,207,169,204
33626 :004,141,203,207,173,203,253
33632 :207,074,144,008,169,010,196
33638 :141,204,207,076,113,131,206
33644 :169,020,141,204,207,173,254
33650 :203,207,141,245,207,169,006
33656 :010,141,244,207,032,252,238
33662 :135,172,242,207,174,204,236
33668 :207,204,240,207,208,008,182
33674 :236,241,207,208,003,076,085
33680 :154,131,032,055,134,208,090
33686 :003,238,202,207,173,204,153
33692 :207,024,105,020,201,160,105
33698 :240,010,201,150,240,006,241
33704 :141,204,207,076,113,131,016
33710 :238,203,207,172,203,207,124
33716 :192,020,208,166,160,000,158
33722 :185,195,132,032,210,255,171
33728 :200,192,021,208,245,169,203
33734 :000,174,202,207,032,205,250
33740 :189,169,032,032,210,255,067
33746 :169,042,032,210,255,169,063
33752 :032,032,210,255,169,000,146
33758 :174,200,207,032,205,189,205
33764 :169,032,032,210,255,169,071
33770 :061,032,210,255,169,032,225
33776 :032,210,255,173,200,207,037
33782 :141,245,207,173,202,207,141
33788 :141,244,207,032,252,135,239
33794 :174,242,207,173,243,207,224
33800 :032,205,189,169,032,032,155
33806 :210,255,169,146,032,210,012
33812 :255,173,200,207,201,070,102
33818 :240,006,024,105,005,141,035
33824 :200,207,173,242,207,056,093
33830 :233,010,141,242,207,141,244
33836 :221,207,173,243,207,233,048
33842 :000,141,243,207,013,221,107
```

```
33848 :207,240,020,144,018,162,079
33854 :253,160,000,200,208,253,112
33860 :232,208,250,162,011,032,195
33866 :035,134,076,034,132,234,207
33872 :169,000,141,021,208,169,020
33878 :096,133,252,169,032,133,133
33884 :254,160,000,133,251,133,255
33890 :253,177,253,145,251,200,097
33896 :208,249,230,252,230,254,247
33902 :166,252,224,127,208,239,046
33908 :177,253,145,251,200,192,054
33914 :064,208,247,169,007,141,190
33920 :046,208,173,254,067,141,249
33926 :255,067,162,039,169,032,090
33932 :157,120,064,202,224,007,146
33938 :208,248,160,006,185,216,145
33944 :132,153,055,138,185,223,014
33950 :132,153,062,138,136,192,203
33956 :255,208,239,032,145,143,162
33962 :173,205,207,056,042,141,226
33968 :021,208,032,166,135,169,139
33974 :000,174,201,207,157,080,233
33980 :064,238,201,207,076,212,162
33986 :140,159,019,017,017,017,051
33992 :029,029,029,029,029,029,118
33998 :029,029,029,018,066,079,200
34004 :078,085,083,032,010,030,018
34010 :040,060,080,110,130,040,166
34016 :080,170,050,090,140,160,146
34022 :142,217,207,140,216,207,079
34028 :200,032,055,134,201,003,093
34034 :240,093,232,032,055,134,004
34040 :201,002,208,085,202,202,124
34046 :032,055,134,201,002,208,118
34052 :076,173,216,207,024,105,037
34058 :019,168,032,055,134,201,107
34064 :002,208,062,232,232,032,016
34070 :055,134,201,002,208,053,163
34076 :173,217,207,024,105,009,251
34082 :170,173,216,207,024,105,161
34088 :009,168,032,055,134,201,127
34094 :002,208,032,200,200,032,208
34100 :055,134,201,002,208,023,163
34106 :173,217,207,056,233,009,185
34112 :170,032,055,134,201,002,146
34118 :208,009,136,136,032,055,134
34124 :134,201,002,240,001,096,238
34130 :174,217,207,172,216,207,251
34136 :236,241,207,208,012,152,120
```

```
34142 :024,105,010,205,240,207,117
34148 :208,003,076,245,130,162,156
34154 :011,032,035,134,169,003,234
34160 :133,002,169,255,141,246,034
34166 :207,141,214,207,172,216,251
34172 :207,200,238,246,207,173,115
34178 :246,207,201,019,240,044,063
34184 :201,010,144,006,206,214,149
34190 :207,076,149,133,238,214,135
34196 :207,173,217,207,024,109,061
34202 :214,207,141,215,207,173,031
34208 :217,207,056,237,214,207,018
34214 :170,202,232,032,239,139,156
34220 :236,215,207,208,247,076,081
34226 :125,133,096,172,253,207,140
34232 :174,254,207,192,040,240,011
34238 :016,192,030,240,012,224,136
34244 :150,240,008,152,056,233,011
34250 :020,168,032,230,132,172,188
34256 :253,207,174,254,207,192,215
34262 :030,240,021,224,000,240,201
34268 :017,224,010,240,013,152,108
34274 :056,233,010,168,138,056,119
34280 :233,010,170,032,230,132,015
34286 :172,253,207,174,254,207,225
34292 :192,030,240,021,224,150,077
34298 :240,017,224,140,240,013,100
34304 :152,056,233,010,168,138,245
34310 :024,105,010,170,032,230,065
34316 :132,172,253,207,174,254,180
34322 :207,192,200,240,011,192,036
34328 :190,240,007,224,150,240,051
34334 :003,032,230,132,096,189,200
34340 :000,064,201,057,240,004,090
34346 :254,000,064,096,169,048,161
34352 :157,000,064,202,076,035,070
34358 :134,152,072,138,072,169,023
34364 :096,133,252,169,000,133,075
34370 :251,138,072,074,074,170,077
34376 :152,072,074,074,074,168,174
34382 :202,224,255,240,014,165,154
34388 :251,024,105,008,133,251,088
34394 :144,242,230,252,076,078,088
34400 :134,136,192,255,240,016,045
34406 :165,251,024,105,064,133,076
34412 :251,165,252,105,001,133,247
34418 :252,076,097,134,104,041,050
34424 :007,168,136,192,255,240,094
34430 :009,230,251,208,247,230,021
```

```
34436 :252,076,122,134,104,041,093
34442 :003,170,169,192,141,247,036
34448 :207,202,224,255,240,009,001
34454 :078,247,207,078,247,207,190
34460 :076,145,134,200,173,247,107
34466 :207,049,251,141,221,207,214
34472 :173,247,207,041,001,208,021
34478 :015,078,247,207,078,247,022
34484 :207,078,221,207,078,221,168
34490 :207,076,168,134,104,170,021
34496 :104,168,173,221,207,096,137
34502 :169,000,141,220,207,169,080
34508 :255,141,219,207,174,220,140
34514 :207,232,236,219,207,240,015
34520 :061,173,220,207,024,109,242
34526 :219,207,106,141,218,207,040
34532 :141,245,207,141,244,207,133
34538 :032,252,135,173,242,207,251
34544 :056,237,222,207,141,221,044
34550 :207,173,243,207,237,223,000
34556 :207,013,221,207,240,026,142
34562 :144,009,173,218,207,141,126
34568 :219,207,076,208,134,173,001
34574 :218,207,141,220,207,076,059
34580 :208,134,173,220,207,141,079
34586 :218,207,096,173,241,207,144
34592 :056,237,254,207,144,003,165
34598 :076,048,135,173,254,207,163
34604 :056,237,241,207,141,245,147
34610 :207,141,244,207,032,252,109
34616 :135,173,242,207,141,236,166
34622 :207,173,243,207,141,237,246
34628 :207,173,240,207,056,237,164
34634 :253,207,144,003,076,088,077
34640 :135,173,253,207,056,237,117
34646 :240,207,141,245,207,141,243
34652 :244,207,032,252,135,173,111
34658 :242,207,024,109,236,207,099
34664 :141,222,207,173,243,207,017
34670 :109,237,207,141,223,207,210
34676 :032,198,134,173,218,207,054
34682 :074,074,074,170,168,169,083
34688 :160,157,127,064,202,224,038
34694 :255,208,248,173,218,207,163
34700 :041,007,170,189,158,135,072
34706 :153,128,064,169,032,153,077
34712 :129,064,153,130,064,096,020
34718 :101,116,117,097,246,234,045
34724 :231,160,173,027,212,041,240
```

```
34730 :015,201,013,176,247,024,078
34736 :105,001,141,241,207,173,020
34742 :027,212,041,007,024,105,086
34748 :001,141,245,207,169,020,203
34754 :141,244,207,032,252,135,181
34760 :173,242,207,141,240,207,130
34766 :173,241,207,041,001,208,053
34772 :009,173,240,207,056,233,106
34778 :010,141,240,207,173,241,206
34784 :207,141,245,207,169,010,179
34790 :141,244,207,032,252,135,217
34796 :173,242,207,141,241,207,167
34802 :173,240,207,024,105,030,253
34808 :141,240,207,096,169,000,077
34814 :141,242,207,162,008,078,068
34820 :245,207,144,004,024,109,225
34826 :244,207,106,110,242,207,102
34832 :202,208,240,141,243,207,233
34838 :096,096,162,006,032,086,244
34844 :136,208,049,032,106,136,183
34850 :208,044,173,027,212,201,131
34856 :064,176,008,169,014,157,116
34862 :183,207,076,080,136,201,161
34868 :128,176,008,169,013,157,191
34874 :183,207,076,080,136,201,173
34880 :192,176,008,169,011,157,009
34886 :183,207,076,080,136,169,153
34892 :007,157,183,207,202,224,032
34898 :255,208,197,096,189,055,058
34904 :138,024,125,062,138,141,204
34910 :250,207,008,104,041,001,193
34916 :141,251,207,076,085,139,231
34922 :189,055,138,056,253,062,091
34928 :138,144,011,141,250,207,235
34934 :169,000,141,251,207,076,194
34940 :085,139,189,062,138,056,025
34946 :253,055,138,141,250,207,150
34952 :169,000,141,251,207,076,212
34958 :085,139,238,182,207,173,142
34964 :182,207,205,181,207,240,090
34970 :001,096,169,255,141,182,230
34976 :207,162,006,189,183,207,090
34982 :041,001,208,029,189,062,184
34988 :138,201,030,208,003,076,060
34994 :030,137,189,055,138,201,160
35000 :150,208,003,076,030,137,020
35006 :254,055,138,222,062,138,035
35012 :076,030,137,189,183,207,250
35018 :041,002,208,023,189,062,215
```

```
35024 :138,201,200,240,073,189,225
35030 :055,138,201,000,240,066,146
35036 :254,062,138,222,055,138,065
35042 :076,030,137,189,183,207,024
35048 :041,004,208,023,189,062,247
35054 :138,201,030,240,043,189,055
35060 :055,138,201,000,240,036,146
35066 :222,055,138,222,062,138,063
35072 :076,030,137,189,183,207,054
35078 :041,008,208,020,189,062,022
35084 :138,201,200,240,013,189,225
35090 :055,138,201,150,240,006,040
35096 :254,055,138,254,062,138,157
35102 :202,224,255,208,128,096,119
35108 :162,006,169,128,141,247,121
35114 :207,189,055,138,010,176,049
35120 :059,105,014,008,072,138,188
35126 :010,170,104,157,002,208,193
35132 :138,074,170,040,173,247,134
35138 :207,073,255,045,016,208,102
35144 :144,003,013,247,207,141,059
35150 :016,208,189,062,138,024,203
35156 :105,041,072,138,010,170,108
35162 :104,157,003,208,138,074,006
35168 :170,078,247,207,202,224,200
35174 :255,208,194,076,134,137,082
35180 :024,105,014,072,138,010,215
35186 :170,104,157,002,208,138,125
35192 :074,170,173,016,208,013,006
35198 :247,207,141,016,208,076,253
35204 :080,137,206,246,207,208,192
35210 :035,169,010,141,246,207,178
35216 :173,248,207,201,021,240,210
35222 :008,169,021,141,248,207,176
35228 :076,164,137,169,020,141,095
35234 :248,207,162,006,157,249,167
35240 :067,202,224,255,208,248,092
35246 :173,141,002,201,001,240,164
35252 :249,096,162,006,169,007,101
35258 :157,040,208,169,020,157,169
35264 :249,067,202,224,255,208,117
35270 :241,162,036,169,003,157,198
35276 :120,216,202,224,255,208,149
35282 :248,162,039,169,032,157,249
35288 :000,064,157,040,064,157,186
35294 :120,064,157,080,064,202,141
35300 :224,255,208,239,160,000,034
35306 :185,069,138,032,210,255,099
35312 :200,192,011,208,245,160,232
```

```
35318 :000,185,088,138,032,210,131
35324 :255,200,192,043,208,245,115
35330 :160,007,185,080,138,153,213
35336 :000,080,136,192,255,208,111
35342 :245,200,185,131,138,032,177
35348 :210,255,200,192,040,208,101
35354 :245,160,000,185,170,138,156
35360 :032,210,255,200,192,016,169
35366 :208,245,160,000,185,080,148
35372 :142,153,000,069,200,208,048
35378 :247,173,030,208,096,010,046
35384 :030,040,060,080,110,130,250
35390 :040,080,170,050,090,140,120
35396 :160,158,019,017,017,017,200
35402 :083,079,078,065,082,058,007
35408 :126,126,126,126,096,096,008
35414 :096,096,154,019,017,017,229
35420 :076,069,086,069,076,058,014
35426 :032,032,032,032,032,032,034
35432 :032,032,032,032,032,032,040
35438 :032,032,032,032,032,032,046
35444 :032,032,032,032,032,032,052
35450 :032,032,032,032,032,032,058
35456 :032,032,032,005,019,083,075
35462 :067,079,082,069,058,032,009
35468 :048,048,048,048,048,048,172
35474 :032,032,032,032,032,032,082
35480 :072,073,071,072,032,083,043
35486 :067,079,082,069,058,032,033
35492 :048,048,048,048,048,048,196
35498 :153,019,017,083,084,065,079
35504 :077,073,078,065,058,032,047
35510 :218,218,218,218,173,249,196
35516 :207,041,001,208,009,238,124
35522 :254,207,206,253,207,076,117
35528 :247,138,173,249,207,041,231
35534 :002,208,009,238,253,207,099
35540 :206,254,207,076,247,138,060
35546 :173,249,207,041,004,208,076
35552 :009,206,254,207,206,253,079
35558 :207,076,247,138,173,249,040
35564 :207,041,008,208,006,238,176
35570 :254,207,238,253,207,032,153
35576 :155,139,162,255,160,000,095
35582 :200,208,253,232,208,250,069
35588 :032,024,136,032,144,136,252
35594 :032,036,137,032,217,130,082
35600 :032,030,139,208,165,032,110
35606 :050,139,208,160,032,181,024
```

```
35612 :133,096,173,254,207,024,147
35618 :109,253,207,141,250,207,177
35624 :008,104,041,001,141,251,074
35630 :207,076,085,139,173,254,212
35636 :207,056,237,253,207,144,132
35642 :011,141,250,207,169,000,068
35648 :141,251,207,076,085,139,195
35654 :173,253,207,056,237,254,226
35660 :207,141,250,207,169,000,026
35666 :141,251,207,173,250,207,031
35672 :056,233,010,141,250,207,217
35678 :173,251,207,233,000,141,075
35684 :251,207,048,028,173,250,033
35690 :207,013,251,207,240,020,020
35696 :173,250,207,056,233,020,027
35702 :141,250,207,173,251,207,067
35708 :233,000,141,251,207,076,008
35714 :102,139,096,169,002,133,003
35720 :002,162,159,160,031,032,170
35726 :239,139,200,192,200,208,040
35732 :248,202,224,150,208,241,141
35738 :096,173,254,207,010,176,046
35744 :029,105,015,141,000,208,146
35750 :173,016,208,041,254,144,234
35756 :002,009,001,141,016,208,037
35762 :173,253,207,024,105,041,213
35768 :141,001,208,076,207,139,188
35774 :024,105,015,141,000,208,171
35780 :173,016,208,009,001,141,232
35786 :016,208,076,178,139,169,220
35792 :002,133,002,174,254,207,212
35798 :172,253,207,032,055,134,043
35804 :201,001,208,005,162,012,041
35810 :032,035,134,174,254,207,038
35816 :172,253,207,032,239,139,250
35822 :096,072,152,072,138,072,072
35828 :169,096,133,252,169,000,039
35834 :133,251,138,072,074,074,224
35840 :170,152,072,074,074,074,104
35846 :168,202,224,255,240,014,085
35852 :165,251,024,105,008,133,186
35858 :251,144,242,230,252,076,189
35864 :007,140,136,192,255,240,226
35870 :016,165,251,024,105,064,143
35876 :133,251,165,252,105,001,175
35882 :133,252,076,026,140,104,005
35888 :041,007,168,136,192,255,079
35894 :240,009,230,251,208,247,215
35900 :230,252,076,051,140,104,145
```

```
35906 :041,003,170,169,063,141,141
35912 :247,207,165,002,010,010,201
35918 :010,010,010,010,202,224,032
35924 :255,240,012,074,074,056,027
35930 :110,247,207,110,247,207,194
35936 :076,082,140,200,072,173,071
35942 :247,207,049,251,145,251,228
35948 :104,017,251,145,251,104,212
35954 :170,104,168,104,096,173,161
35960 :025,208,141,025,208,041,000
35966 :001,240,077,169,012,160,017
35972 :059,162,024,142,022,208,237
35978 :141,033,208,140,017,208,117
35984 :141,024,208,173,018,208,148
35990 :201,081,144,016,169,000,249
35996 :141,018,208,173,017,208,153
36002 :041,127,141,017,208,076,004
36008 :206,140,169,000,141,033,089
36014 :208,169,200,141,022,208,098
36020 :169,027,141,017,208,169,143
36026 :004,141,024,208,169,081,045
36032 :141,018,208,173,017,208,189
36038 :041,127,141,017,208,076,040
36044 :049,234,104,168,104,170,009
36050 :104,064,169,070,141,254,244
36056 :207,169,120,141,253,207,033
36062 :162,039,169,032,157,120,133
36068 :064,202,224,007,208,248,157
36074 :032,155,139,032,029,135,244
36080 :173,199,207,201,007,240,243
36086 :057,032,248,141,032,024,012
36092 :136,032,144,136,032,036,000
36098 :137,162,255,160,000,200,148
36104 :208,253,232,208,250,162,041
36110 :249,160,000,200,208,253,060
36116 :232,208,250,173,000,220,079
36122 :041,016,208,220,162,013,174
36128 :169,032,157,040,064,232,214
36134 :224,039,208,248,173,030,192
36140 :208,076,247,129,162,039,137
36146 :169,013,157,040,216,202,079
36152 :224,255,208,248,160,000,127
36158 :162,000,189,041,064,157,163
36164 :040,064,232,224,037,208,105
36170 :245,185,037,142,141,077,133
36176 :064,152,072,160,000,162,178
36182 :000,200,208,253,232,208,163
36188 :250,104,168,173,000,220,239
36194 :041,016,240,010,200,192,029
```

```
36200 :043,208,211,160,000,076,034
36206 :062,141,162,000,189,007,159
36212 :064,221,031,064,240,018,242
36218 :144,021,162,005,189,007,138
36224 :064,157,031,064,202,224,102
36230 :255,208,245,076,145,141,180
36236 :232,224,006,208,225,162,173
36242 :005,189,031,064,157,192,016
36248 :207,202,224,255,208,245,213
36254 :173,000,220,041,016,240,080
36260 :249,032,073,145,032,182,109
36266 :137,162,005,189,192,207,038
36272 :157,031,064,202,224,255,085
36278 :208,245,169,096,133,252,005
36284 :169,032,133,254,160,000,168
36290 :133,251,133,253,177,253,114
36296 :145,251,200,208,249,230,203
36302 :252,230,254,166,252,224,048
36308 :127,208,239,177,253,145,081
36314 :251,200,192,064,208,247,100
36320 :032,145,143,169,007,141,093
36326 :021,208,141,201,207,169,153
36332 :040,141,200,207,169,012,237
36338 :141,199,207,076,212,140,193
36344 :162,000,189,006,142,032,011
36350 :210,255,232,224,032,208,135
36356 :245,096,005,019,017,029,159
36362 :029,029,029,029,029,029,184
36368 :029,029,029,029,029,029,190
36374 :029,029,029,080,082,069,084
36380 :083,083,032,066,085,084,205
36386 :084,079,078,135,129,141,168
36392 :133,160,143,150,133,146,137
36398 :032,046,046,032,016,018,236
36404 :005,019,019,032,002,021,150
36410 :020,020,015,014,032,020,179
36416 :015,032,016,012,001,025,165
36422 :032,001,007,001,009,014,134
36428 :032,046,046,032,000,000,232
36434 :000,000,000,000,000,000,082
36440 :000,000,000,000,000,000,088
36446 :000,000,000,000,000,000,094
36452 :000,000,048,000,000,048,196
36458 :000,000,252,000,000,252,098
36464 :000,000,048,000,000,048,208
36470 :000,000,000,000,000,000,118
36476 :000,000,000,000,000,000,124
36482 :000,000,000,000,000,000,130
36488 :000,000,000,000,000,000,136
```

```
36494 :000,000,000,000,000,000,142
36500 :000,000,000,000,000,000,148
36506 :000,000,000,000,000,000,154
36512 :000,000,000,000,000,001,161
36518 :140,000,000,216,000,000,010
36524 :112,000,000,112,000,000,140
36530 :216,000,001,140,000,000,023
36536 :000,000,000,000,000,000,184
36542 :000,000,000,000,000,000,190
36548 :000,000,000,000,000,000,196
36554 :000,000,000,000,000,000,202
36560 :000,000,000,000,000,000,208
36566 :000,000,000,000,000,000,214
36572 :000,000,000,000,000,000,220
36578 :000,000,000,000,096,000,066
36584 :000,240,000,001,248,000,209
36590 :000,240,000,000,096,000,062
36596 :000,000,000,000,000,000,244
36602 :000,000,000,000,000,000,250
36608 :000,000,000,000,000,000,000
36614 :000,000,000,000,000,000,006
36620 :000,000,000,000,000,000,012
36626 :000,000,000,000,000,000,018
36632 :000,000,000,000,015,255,038
36638 :255,023,224,049,016,000,085
36644 :001,035,102,051,102,219,034
36650 :051,075,108,103,199,254,064
36656 :102,128,000,014,255,255,034
36662 :252,255,255,248,000,000,040
36668 :000,000,000,000,000,000,060
36674 :000,000,000,000,000,000,066
36680 :000,000,000,000,000,000,072
36686 :000,000,165,162,007,169,069
36692 :000,157,000,212,202,224,111
36698 :255,208,248,169,141,141,228
36704 :024,212,169,005,141,005,140
36710 :212,169,218,141,006,212,036
36716 :169,150,141,001,212,169,182
36722 :139,141,000,212,169,017,024
36728 :141,004,212,160,140,162,171
36734 :000,232,208,253,200,208,203
36740 :250,169,016,141,004,212,156
36746 :232,208,253,200,208,250,209
36752 :096,160,000,169,000,153,210
36758 :000,212,200,192,008,208,202
36764 :246,169,143,141,024,212,067
36770 :169,008,141,005,212,169,098
36776 :243,141,006,212,169,129,044
36782 :141,004,212,162,255,142,066
```

```
36788 :001,212,202,160,255,136,122
36794 :192,001,208,251,224,001,039
36800 :208,241,169,128,141,004,059
36806 :212,096,162,000,169,000,069
36812 :157,000,212,232,224,008,013
36818 :208,248,169,143,141,024,119
36824 :212,169,017,141,005,212,204
36830 :169,213,141,006,212,169,108
36836 :002,141,003,212,169,100,087
36842 :141,002,212,169,005,141,136
36848 :001,212,169,135,141,000,130
36854 :212,169,065,141,004,212,025
36860 :169,252,160,000,162,000,227
36866 :200,208,253,232,208,250,073
36872 :024,105,001,208,245,169,248
36878 :064,141,004,212,096,162,181
36884 :000,142,033,208,142,032,065
36890 :208,189,206,144,240,007,252
36896 :032,210,255,232,076,027,096
36902 :144,234,173,190,207,105,067
36908 :048,032,210,255,169,157,147
36914 :032,210,255,032,228,255,038
36920 :208,010,173,000,220,041,196
36926 :016,208,244,076,091,144,073
36932 :201,013,240,019,201,049,023
36938 :144,233,201,058,176,229,091
36944 :072,056,233,048,141,190,052
36950 :207,104,032,210,255,162,032
36956 :000,160,000,232,208,253,177
36962 :200,208,250,173,000,220,125
36968 :041,016,240,249,189,050,121
36974 :145,240,007,032,210,255,231
36980 :232,076,108,144,173,180,005
36986 :207,032,210,255,169,157,128
36992 :032,210,255,032,228,255,116
36998 :208,010,173,000,220,041,018
37004 :016,208,244,076,167,144,227
37010 :201,013,240,017,201,069,119
37016 :240,007,201,072,240,003,147
37022 :076,131,144,141,180,207,013
37028 :032,210,255,234,056,169,096
37034 :010,237,190,207,010,073,129
37040 :255,024,105,002,141,177,112
37046 :130,141,251,138,141,004,219
37052 :141,173,180,207,056,233,154
37058 :069,074,073,001,141,181,221
37064 :207,169,001,133,204,096,242
37070 :014,147,017,017,159,018,066
37076 :029,029,029,029,029,029,130
```

```
37082 :029,032,032,032,032,032,151
37088 :032,032,017,157,157,157,008
37094 :157,157,157,157,032,211,077
37100 :208,201,203,197,032,146,199
37106 :032,045,032,005,032,194,070
37112 :089,032,197,082,073,067,020
37118 :032,194,082,065,078,068,005
37124 :079,078,013,029,029,029,005
37130 :029,029,029,029,159,018,047
37136 :032,032,032,032,032,032,208
37142 :032,013,013,013,013,013,119
37148 :029,029,029,029,029,029,202
37154 :158,211,080,069,069,068,177
37160 :032,040,049,045,057,041,048
37166 :063,032,159,000,013,013,070
37172 :013,029,029,029,029,029,210
37178 :029,158,197,065,083,089,167
37184 :047,200,065,082,068,063,077
37190 :032,159,000,120,173,013,055
37196 :220,009,129,141,013,220,040
37202 :169,000,141,026,208,169,027
37208 :234,141,021,003,169,049,193
37214 :141,020,003,088,169,021,024
37220 :141,024,208,169,027,141,042
37226 :017,208,169,199,141,000,072
37232 :221,169,004,141,136,002,017
37238 :169,000,141,021,208,032,177
37244 :019,144,169,064,141,136,029
37250 :002,169,198,141,000,221,093
37256 :169,008,141,024,208,169,087
37262 :216,133,252,160,000,132,011
37268 :251,169,006,145,251,200,146
37274 :208,251,230,252,166,252,233
37280 :224,220,208,243,032,183,246
37286 :128,169,255,141,182,207,224
37292 :096,000,255,013,013,013,050
```

Martian Prisoner

Alan Poole

"Martian Prisoner" is a mini-adventure game for the Commodore 64. If you've never played an adventure game before, this is a good introduction. Unlike most computer games, text adventures have no graphics and do not require fast reflexes—instead, they test the player's patience and cunning.

Without warning, the Martians have suddenly started a devastating war against Earth. They have captured you and are holding you prisoner in a cell on a Martian space cruiser headed toward Earth. The cruiser also carries a secret weapon that can neutralize all of Earth's defenses. Your task is to destroy the Martian ship and escape in a lifecraft before the Martians can complete their sinister mission.

Like Radio Dramas

"Martian Prisoner" is a mini-adventure game, using only a little more than 3K of memory. Adventure games require you to solve puzzles and explore a simulated world inside the computer. The computer will describe what you see and what happens, and you tell the computer what you want to do. Instead of using screen graphics, adventure games rely on text descriptions and your imagination. It's like the difference between old-time radio dramas and television; despite the visual impact of video, the mind can still imagine a scene more exciting than a camera can picture.

In Martian Prisoner, you start off in the prison cell of the Martian space cruiser. Besides the cell, the cruiser contains several other rooms. It's up to you to explore the rooms and find a way to destroy the ship. In each room, the computer will describe your surroundings and list the objects in the room. The computer then waits for you to type a command, consisting of one or two words.

For example, you would type GO NORTH to move north.

If there is a book in the room, you would type GET BOOK to pick it up. Type INVENTORY at any time to see a list of the objects you are carrying. All commands and nouns can be abbreviated to the first three letters. You can list your INVENTORY by typing INV, for instance.

The commands you can use, with the abbreviations capitalized, are:

North	INVentory
East	REAd
South	OPEn
West	WEAr
GO	EAT
GET	KILl
DROp	HIT

Although Martian Prisoner is a short adventure game, you must solve several puzzles to win. It's a good way to prepare for the more elaborate adventure games available commercially for Commodore 64s.

If you haven't played a text-adventure game before, it may be a good idea to draw a map of the cruiser as you explore its rooms. Using the map, you can easily backtrack if you run into a dead end or want to explore a side passage you earlier passed by.

Martian Prisoner doesn't award points for accomplishing tasks, as some other adventure games do. Because of its short length, you simply win by destroying the cruiser and escaping, or lose by getting your character killed. Of course, you can always try again!

Typing In

Take a look at the program listing for Martian Prisoner. You'll notice the characters :rem xxx on the far right of each line. These are *not* characters you will type in. They have to do with "Automatic Proofreader," in Appendix J, and are in effect checksums. Be sure to read Appendix J before you begin to type in Martian Prisoner. The Automatic Proofreader program will make mistake-proof entry a snap.

Martian Prisoner

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

```
10 GOSUB5000                                    :rem 166
100 GOSUB1000:IFR=6ANDU=0THENR=1:GOTO100        :rem 232
110 GOSUB2000:PRINT                             :rem 155
120 ONVGOSUB3000,3000,3000,3000,3100,3200,3300,340
    0,3500,3600,3700,3800,3900                   :rem 42
125 IFV=14THEN3900                               :rem 26
130 GOTO100                                      :rem 94
1000 PRINT:ONRGOSUB1100,1200,1300,1400,1500,1600,1
     700,1800                                    :rem 87
1003 IFW1=299THEN1010                           :rem 173
1005 IFI(4)=-1ANDI(6)=-1ANDW1=299THENPRINTNS$
                                                :rem 185
1010 PRINT:PRINT"{CYN}{6 RIGHT}OBJECTS:{WHT}"
                                                :rem 240
1020 FORL=1TO8:IFI(L)=RTHENPRINT"{6 RIGHT}";N$(L)
                                                :rem 117
1030 NEXT:PRINT:RETURN                          :rem 228
1100 PRINT"{6 RIGHT}YOU ARE IN A PRISON CELL."
                                                 :rem 23
1110 IFRND(1)>.25THENRETURN                     :rem 154
1120 G=1:PRINT"{6 RIGHT}A GUARD HAS TURNED OFF THE
      FORCE{8 SPACES}FIELD ";                    :rem 94
1125 PRINT"AND ENTERED THE CELL."               :rem 163
1130 C%(1,1)=2:RETURN                           :rem 149
1200 PRINT"{6 RIGHT}YOU ARE IN A N/S HALL.":RETURN
                                                :rem 40
1300 PRINT"{6 RIGHT}YOU ARE IN THE ENGINE ROOM.":R
     ETURN                                      :rem 203
1400 PRINT"{6 RIGHT}YOU ARE IN A SMALL ROOM. A LAR
     GE{8 SPACES}SIGN IS ON THE WALL."           :rem 42
1410 IFI(4)=-1THENPRINTNS$                       :rem 63
1420 IFI(6)=-1THEN4500                          :rem 185
1430 RETURN                                     :rem 168
1500 PRINT"{6 RIGHT}YOU ARE IN THE SUPPLY ROOM.":R
     ETURN                                       :rem 4
1600 PRINT"{6 RIGHT}YOU ARE IN THE NORTH{2 SPACES}
     SIDE OF THE{7 SPACES}HALL."                 :rem 8
1610 IFU=1THENPRINT"{6 RIGHT}THE GUARDS DON'T NOTI
     CE YOU."                                   :rem 184
1620 IFU=0THENPRINT"{6 RIGHT}THE GUARDS TAKE YOU B
     ACK TO THE{9 RIGHT}CELL.":G=0              :rem 103
1630 RETURN                                     :rem 170
1700 PRINT"{6 RIGHT}YOU ARE IN A LARGE ROOM.":RETU
     RN                                         :rem 228
1800 PRINT"{6 RIGHT}YOU ARE IN A STRANGE GARDEN WH
     ERE{7 SPACES}FOOD IS ";                     :rem 52
```

```
1805 PRINT"GROWN FOR THE CREW."                    :rem 83
1810 IFI(4)=-1THENPRINTNS$:W1=299                  :rem 230
1820 RETURN                                        :rem 171
2000 C$="":N=0:V=0:PRINT:INPUT"{6 RIGHT}COMMAND
     {GRN}";C$:PRINT"{WHT}":IFC$=""THEN2000
                                                   :rem 144
2015 P=0:IFLEN(C$)<2THEN2050                        :rem 73
2020 FORL=2TOLEN(C$)-1                             :rem 254
2030 IFMID$(C$,L,1)=" "THENP=L                     :rem 104
2040 NEXT                                           :rem 5
2050 IFP=0THENV$=C$:N$=""                          :rem 141
2060 IFP>0ANDP=LEN(C$)THENV$=C$:N$=""              :rem 134
2070 IFP>0ANDP<LEN(C$)THENV$=LEFT$(C$,P-1):N$=RIGH
     T$(C$,LEN(C$)-P)                               :rem 86
2080 FORL=1TO14:IFLEFT$(V$,3)=V$(L)THENV=L         :rem 23
2100 NEXT:FORL=1TO8:IFLEFT$(N$,3)=A$(L)THENN=L
                                                   :rem 55
2120 NEXT:IFN>0ANDV>0THENRETURN                     :rem 47
2130 IFN=0ANDV>0ANDN$=""THENRETURN                 :rem 124
2135 IFN=0ANDV=5THENRETURN                         :rem 191
2140 PRINT:PRINT"{6 RIGHT}I DON'T UNDERSTAND.":GOT
     O2000                                         :rem 13
3000 N$=V$:GOTO3110                                 :rem 36
3100 N$=LEFT$(N$,1)                                :rem 226
3110 IFR=1ANDN$="E"ANDG=0THENPRINT"{6 RIGHT}THE FO
     RCE FIELD STOPS YOU.":RETURN                  :rem 230
3120 IFR<>1ORN$<>"E"ORG=0THEN3130                  :rem 179
3125 PRINT"{6 RIGHT}AS YOU LEAVE THE CELL THE FORC
     E{9 SPACES}FIELD IS ACTIVATED, ";             :rem 149
3127 PRINT" TRAPPING THE{7 SPACES}GUARD."          :rem 133
3130 IFR=2ANDN$="E"ANDC%(2,1)=0ANDI(8)>-1THENPRINT
     "{6 RIGHT}DOOR LOCKED."                       :rem 173
3135 IFR=2ANDN$="E"ANDC%(2,1)=0ANDI(8)>-1THENRETUR
     N                                             :rem 255
3140 IFR<>2ORN$<>"E"ORC%(2,1)>0THEN3150            :rem 186
3145 PRINT"{6 RIGHT}YOU UNLOCK THE DOOR WITH THE K
     EY.":C%(2,1)=5:N$(7)="OPEN DOOR"              :rem 109
3150 IFN$="N"THEND=0                               :rem 121
3160 IFN$="E"THEND=1                               :rem 114
3165 IFN$="S"THEND=2                               :rem 134
3170 IFN$="W"THEND=3                               :rem 135
3175 IFC%(R,D)=0THENPRINTCN$:RETURN                :rem 210
3180 PRINT"{6 RIGHT}OK":R=C%(R,D):RETURN            :rem 67
3200 IFN=1ORN=2ORN=3ORN=7THENPRINT"{6 RIGHT}YOU CA
     N'T LIFT IT!":RETURN                          :rem 47
3203 IFI(N)<>RTHENPRINT"{6 RIGHT}IT'S NOT HERE.":R
     ETURN                                         :rem 45
3205 IFN=5THEN3720                                  :rem 20
3210 PRINT"{6 RIGHT}OK":I(N)=-1:RETURN             :rem 182
```

```
3300 PRINT"{6 RIGHT}OK":I(N)=R:RETURN      :rem 170
3400 PRINT"{6 RIGHT}YOU ARE CARRYING:"     :rem 180
3410 FORL=1TO8:IFI(L)=-1THENPRINT"{6 RIGHT}";N$(L)
                                           :rem 134
3420 NEXT:RETURN                           :rem 34
3500 IFN<>3ORR<>4THENPRINTCN$:RETURN       :rem 126
3510 PRINT"{6 RIGHT}ATOMIC FUEL NEARBY. DON'T BRIN
     G{9 SPACES}ANY RADIO-ACTIVE"          :rem 46
3520 PRINT"{6 RIGHT}MATERIALS INTO THIS ROOM.":RET
     URN                                   :rem 226
3600 IFN<>7ORR<>2ORI(8)>-1THENPRINTCN$:RETURN
                                           :rem 144
3610 N$="E":GOTO3145                       :rem 66
3700 IFI(5)<>RTHENPRINTCN$:RETURN          :rem 127
3720 PRINT"{6 RIGHT}YOU ARE NOW WEARING A UNIFORM.
     ":I(5)=-1:U=1:RETURN                  :rem 107
3800 IFN<>6THENPRINTRI$:RETURN             :rem 237
3810 PRINT"{6 RIGHT}YOU QUICKLY BECOME SICK AND DI
     E.":GOTO4600                          :rem 79
3900 PRINT"{6 RIGHT}THE GUARD SHOOTS YOU.":GOTO460
     0                                     :rem 231
3910 PRINTCN$:RETURN                       :rem 41
4500 PRINT                                 :rem 86
4505 PRINT"{6 RIGHT}THE RADIOACTIVE PLANT EMITS EN
     OUGH{6 SPACES}NEUTRONS TO START A"     :rem 16
4510 PRINT"{6 RIGHT}CHAIN REACTION. THE SHIP EXPLO
     DES."                                 :rem 214
4515 PRINT"{6 RIGHT}YOU ESCAPE IN A LIFE-CRAFT."
                                           :rem 191
4520 PRINT:PRINT"{PUR}{6 RIGHT}YOU WIN!":GOTO4610
                                           :rem 247
4600 PRINT:PRINT"{PUR}{6 RIGHT}YOU LOSE!" :rem 253
4610 PRINT:PRINT:PRINT"{6 RIGHT}{GRN}PLAY AGAIN?"
                                           :rem 203
4620 GETK$:IFK$="Y"THENRUN                 :rem 81
4630 IFK$="N"THENEND                       :rem 160
4640 GOTO4620                              :rem 211
5000 PRINT"{HOME}{CLR}":POKE36879,8:PRINT"{4 DOWN}
     {10 RIGHT}{GRN}{RVS}MARTIAN PRISONER{OFF}":PR
     INT                                   :rem 146
5080 DIMV$(14),C%(8,3),I(8),N$(8),A$(8)    :rem 146
5090 R=1:FORL=1TO14:READV$(L):NEXT          :rem 87
5100 FORL=1TO8:READC%(L,0),C%(L,1),C%(L,2),C%(L,3)
     :NEXT                                 :rem 31
5110 FORL=1TO8:READN$(L),A$(L),I(L):NEXT   :rem 97
5115 CN$="{6 SPACES}YOU CAN'T":RI$="{6 SPACES}DON'
     T BE SILLY!"                          :rem 87
5120 NS$="{6 SPACES}GEIGER COUNTER IS CLICKING.":R
     ETURN                                 :rem 9
```

```
6000 DATAN,E,S,W,GO,GET,DRO,INV,REA,OPE,WEA,EAT,KI
     L,HIT                                    :rem 217
6010 DATA0,0,0,0,6,0,3,0,2,4,0,0,0,0,0,3,0,0,0,2,7
     ,0,2,0,0,8,6,0,0,0,0,7                   :rem 103
6020 DATAFORCE FIELD,FOR,1,GUARDS,GUA,6,SIGN,SIG,4
     ,GEIGER COUNTER,COU,5,UNIFORM            :rem 13
6030 DATA UNI,5,PLANT,PLA,8,LOCKED DOOR,DOO,2,MAGN
     ETIC KEY,KEY,3                           :rem 151
```

64 Mailing List

Joseph J. Shaughnessy

Keeping track of your mailing list (or other kinds of files) is simple when you use this program on your Commodore 64. One alteration lets you use tape instead of a disk drive, and you can even print labels from your list.

"64 Mailing List" is a modified and expanded version of a utility program called "Addresses" from the Toronto PET Users Group. It was originally written in Dutch by Andy Finkel. The program has been translated into English and a printer option added. Using a Commodore printer, it can print the entire list or individual mailing labels.

The program is for the Commodore 64 and 1540/1541 disk drive, but it can easily be modified to operate with the cassette recorder by changing line 500 to:

```
500 PRINT"{CLR}":SAVER$:END
```

How to Use Mailing List

Once you've typed in and SAVEd the program, RUN it. You'll see a display of eight functions. They are:

1. Add Name. For adding new names and other information to your mailing list. After pressing the 1 key, you'll see another display. It will take you through a step-by-step process of entering information you want. Enter the person's name, last name first. Don't use commas to separate last name from first name. Next you enter the street address, city, state, zip code, and telephone number. If the person has a place of business, you can enter that also, along with the work telephone number. If any information requested is unknown, enter 0 for that category.

2. Removal. You can remove all information under one name using this function. Pressing the 2 key clears the screen and then asks for the item to be removed, or erased. Enter the data item number and the name, and all information under

that number will disappear from your mailing list.

3. Search. Use this to search through your entire list, looking for a particular name, address, zip code, or phone number. The screen clears after you press this key, and asks for the information you want the computer to search for. If it's in the mailing list, it will appear on the screen. If it's not in the list, the message "Field Not Found" displays.

4. Examine. You can look at any name and its accompanying information with this function. You can't change anything (see the next function), and when you're ready to move back to the main menu, hitting any key will return you there. This function is best used to check that information is entered correctly.

5. Change. Perhaps the function you'll use most often, this function allows you to update any name and its information, simply by specifying the item number. (You can locate the item number of the information you would like to change by first using the search function. Once the item is found, you'll see its item number at the top of the screen.) After you've specified the item number, the information will appear on the screen, in the order you entered the data. All you have to do is move down to the line you want to change by hitting the RETURN key the correct number of times. Make sure you use the RETURN key to move the cursor down, not the cursor control keys. If you use the latter, you may accidentally insert unwanted characters into the data. After making your changes, use the RETURN key to cursor down through the rest of the data. Hitting the RETURN key when the cursor is on the last line of data returns you to the main menu.

6. Save Update. When you're finished adding new data items or changing existing ones, simply press the 6 key. Make sure that you have a tape in the Datassette or a disk in the drive before you use this function, for the program will immediately begin to save out to tape or disk. Your new information is now included in the program.

7. Print. If you have a printer connected to your 64, you can use this function to print mailing labels from your list. You'll see the screen clear after you press the 7 key, and you'll be offered five choices, ranging from printing the entire list to printing individual mailing labels. You can return to the main menu by pressing the 5 key from this display.

8. End. Pressing the 8 key ends the program, showing the READY message on the screen. Make sure you SAVE any new information before you use this function.

Each address field is set up to receive eight items of information, as shown in lines 120 and 130. These items can be changed to anything you want (for instance, to set up a filing system instead of a mailing list), but you are limited to eight items because of the size of the keyboard buffer (line 230). Also, since the DATA statements are printed on the screen as part of the procedure for adding them to the program, you must be careful not to make your items of information so wordy that printing eight DATA statements would cause the first few lines to scroll off the screen and thereby be lost. Also, make sure that the statement DATA"END" follows *all* the name and information entries.

Search, Space, and Print

At one point, I had a version of this program that used upper- and lowercase letters, but I found this too inconvenient when using the "search" function. I often forgot to use appropriate capital letters either when entering the original information or when inputting the search value.

To aid in searching, names are entered and stored last name first, but they are sent to the printer first name first. Do not use commas when entering your mailing list items.

This program will easily fit into the 64. For instance, I have 65 names stored, and it takes about 12K of memory. If you need space for more names just add more DATA statements to the end of the program.

The program prints mailing labels in a single column. Further work could be done to print the labels two or three across the width of the paper, and the formatting could be changed to match the layout of adhesive labels.

I addressed my Christmas cards with this program (tape version) and found it a big timesaver, even though I had to use scissors and tape to put the labels on the envelopes.

A disk drive or printer will certainly enhance the program's usefulness, but neither is essential. The program can probably be modified to run on other computers, depending on the size of the keyboard buffer.

64 Mailing List

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

```
100 C=53280:REM 64 MAILING LIST PROGRAM-DISK VERSI
    ON                                          :rem 110
110 POKEC,5:POKEC+1,5:READR$,R:FORI=1TOR:READO$(I)
    :NEXT:DATA"64{2 SPACES}MAILING{2 SPACES}LIST
                                                :rem 222
120 DATA8,"NAME(LAST NAME FIRST)","STREET ADDRESS"
    ,"CITY","STATE","ZIPCODE"                    :rem 9
130 DATA"HOME PHONE NO.","COMPANY NAME","WORK PHON
    E NO."                                      :rem 0
140 PRINT"{CLR}{BLK}{9 RIGHT}{A} **********
    **{S}":PRINT"{9 SPACES}B "R$" -"            :rem 144

150 PRINT"{9 RIGHT}{Z} ********************{X}
                                                :rem 244

160 PRINT"{DOWN} 1. ADD NAME":PRINT"{DOWN} 2. REMO
    VAL{4 SPACES}"                              :rem 84
165 PRINT"{DOWN} 3. SEARCH":PRINT"{DOWN} 4. EXAMIN
    E                                           :rem 248
170 PRINT"{DOWN}{SHIFT-SPACE}5. CHANGE":PRINT"
    {DOWN} 6. SAVE UPDATE"                      :rem 149
175 PRINT"{DOWN} 7. PRINT OPTION":PRINT"{DOWN} 8.
    {SPACE}END                                  :rem 129
180 RESTORE:PRINT"{2 DOWN}WHICH DO YOU WANT?"
                                                :rem 102
190 GETA$:IFA$=""THEN190                         :rem 87
200 IFA$<"1"ORA$>"8"THEN190                     :rem 192
210 READB$:IFB$<>"{-}"THEN210                   :rem 160
220 A=VAL(A$):ONAGOTO240,290,320,420,490,500,520,5
    10                                          :rem 235
230 POKE198,10:FORI=0TO9:POKE631+I,13:NEXT:END
                                                 :rem 55
240 READA$:IFA$<>"{+}"THEN240                   :rem 110
250 READA:PRINT"{CLR}INPUT 0 FOR UNKNOWNS{DOWN}"
                                                :rem 205
260 PRINT"ITEM : "A"{DOWN}":FORI=1TOR:PRINTO$(I):I
    NPUTW$(I):PRINT:IFW$(1)=""THEN140           :rem 186
270 NEXT:W$(0)="XX"+CHR$(34)+","+STR$(A):Z=A*10+91
    0:K=0:PRINT"{CLR}{2 DOWN}"                  :rem 172
280 FORI=ZTOZ+R:PRINTI;"DATA"CHR$(34)W$(K):K=K+1:N
    EXT:PRINT"RUN{HOME}":GOTO230                 :rem 81
290 B$="":PRINT"{CLR}WHICH ITEM TO REMOVE ":INPUTB
    $:IFVAL(B$)=0THEN140                        :rem 247
300 PRINT"{CLR}{2 DOWN}":Z=VAL(B$)*10+910:PRINTZ"D
    ATA"CHR$(34)"{+}"CHR$(34)","VAL(B$)         :rem 185
310 FORI=Z+1TOZ+R:PRINTI:NEXT:PRINT"RUN{HOME}":GOT
    O230                                        :rem 213
```

```
320 INPUT"{CLR}SEARCH FOR ";B$:IFB$=""THEN140
                                          :rem 174
330 H=0:READA$                             :rem 6
335 IFA$="END"THENPRINT"{CLR}{RIGHT}{DOWN}FIELD NO
    T FOUND":FORT=0TO 2000:NEXT:GOTO140    :rem 65
340 IFA$="{+}"THENREADA:GOTO330            :rem 71
350 READA:FORI=1TOR:READA$(I):IFLEFT$(A$(I),LEN(B$
    ))=B$THENH=1                           :rem 201
360 NEXT:IFH=0THEN330                      :rem 27
370 PRINT"{CLR}ITEM :"A"{2 DOWN}":FORI=1TOR:PRINT"
    {2 SPACES}"A$(I):NEXT:IFW=1THENRETURN  :rem 251
380 PRINT"{2 DOWN}HIT ANY KEY TO PROCEED"  :rem 233
390 GETA$:IFA$=""THEN390                   :rem 91
400 IFQ=1THENRETURN                        :rem 241
410 GOTO140                                :rem 99
420 A$="":INPUT"{CLR}WHICH ITEM";A$:A=VAL(A$):IFA$
    =""ORA<1THEN140                        :rem 114
430 READA$:IFA$="END"THEN140               :rem 98
440 IFA$<>"XX"THEN430                      :rem 192
450 READA$:IFA<>VAL(A$)THEN430             :rem 253
460 READA$(1):IFA$(1)="{+}"THEN140         :rem 56
470 FORI=2TOR:READA$(I):NEXT:Q=1:GOSUB370:Q=0:IFW=
    1THENRETURN                            :rem 223
480 GOTO140                                :rem 106
490 W=1:GOSUB420:W=0:PRINT"{HOME}{2 DOWN}":FORI=1T
    OR:INPUTW$(I):GOTO270                  :rem 168
500 PRINT"{CLR}":SAVE"@0:"+R$,8:END        :rem 102
510 END                                    :rem 109
520 PRINT"{CLR}{2 DOWN}{RVS}{3 SPACES}PRINTER OPTI
    ONS{3 SPACES}"                         :rem 127
530 PRINT"{DOWN}{RVS}1{OFF} ENTIRE LIST"   :rem 82
540 PRINT"{DOWN}{RVS}2{OFF} MAILING LABELS" :rem 5
550 PRINT"{DOWN}{RVS}3{OFF} INDIVIDUAL DATA"
                                           :rem 86
560 PRINT"{DOWN}{RVS}4{OFF} SINGLE MAILING LABEL"
                                          :rem 120
570 PRINT"{DOWN}{RVS}5{OFF} RETURN TO PROGRAM"
                                          :rem 242
580 GETZ$:IFZ$=""THEN580                  :rem 143
590 Z=VAL(Z$)                             :rem 231
600 IFZ<1ORZ>5THEN520                      :rem 32
610 OPEN1,4:RESTORE                       :rem 184
620 ONZGOTO640,740,810,810                :rem 179
630 CLOSE1:GOTO140                         :rem 72
640 READB$:IFB$<>"{-}"THEN640             :rem 174
650 READB$:IFB$="{+}"THENCLOSE1:GOTO140    :rem 81
660 IFB$="XX"THENREADA:PRINT#1,CHR$(10)CHR$(10)"IT
    EM";A:GOSUB690:GOTO650                 :rem 81
670 PRINT#1,B$                             :rem 16
```

```
680 GOTO650                                          :rem 114
690 READB$:FORI=1TO50                                 :rem 1
700 IFMID$(B$,I,1)=" "THENX=I:I=50                   :rem 96
710 NEXTI                                            :rem 32
720 N2$=LEFT$(B$,X):N1$=RIGHT$(B$,LEN(B$)-X):PRINT
    #1                                               :rem 217
730 PRINT#1,N1$;" ";N2$:RETURN                       :rem 194
740 READB$:IFB$<>"{-}"THEN740                        :rem 176
750 READB$:IFB$="{+}"THENCLOSE1:GOTO140               :rem 82
760 IFB$<>"XX"THEN750                                :rem 203
770 READA:PRINT#1,CHR$(10):GOSUB690:GOSUB780:GOTO7
    50                                               :rem 182
780 FORI=1TO4:READA$(I):NEXT                         :rem 226
790 PRINT#1,A$(1):PRINT#1,A$(2);", ";A$(3);"
    {3 SPACES}";A$(4)                                 :rem 54
800 RETURN                                           :rem 120
810 INPUT"{CLR}{DOWN}WHICH ITEM";Q:RESTORE:rem 157
820 READB$:IFB$<>"{-}"THEN820                        :rem 174
830 READ B$                                           :rem 29
840 IFB$="{+}"THENPRINT"NO SUCH ITEM ON FILE":FO
    RX=0TO1500:NEXTX:CLOSE1:GOTO 140                 :rem 125
850 IFB$="XX"THENREADA:IFA=QTHEN870                  :rem 181
860 GOTO830                                          :rem 114
870 PRINT#1,CHR$(10)CHR$(10)"ITEM";A:GOSUB690
                                                      :rem 90
880 IFZ=4THEN900                                     :rem 194
890 FORX=1TO R-1:READB$:PRINT#1,B$:NEXT:CLOSE1:GOT
    O520                                             :rem 110
900 GOSUB780:CLOSE1:GOTO520                          :rem 163
910 DATA"{-}"                                        :rem 212
920 DATA"XX", 1                                       :rem 6
921 DATA"SMYTHE RANDY                                :rem 48
922 DATA"5000 STATE STREET                           :rem 246
923 DATA"SOMECITY                                     :rem 71
924 DATA"SOMESTATE                                   :rem 144
925 DATA"00000                                       :rem 204
926 DATA"111-555-5555                                 :rem 61
927 DATA"WIDGETS INC.                                :rem 253
928 DATA"111-555-5551                                 :rem 59
930 DATA"{+}", 2                                     :rem 254
940 DATA"{+}",3                                       :rem 0
950 DATA"{+}",4                                       :rem 2
960 DATA"{+}",5                                       :rem 4
970 DATA"{+}",6                                       :rem 6
980 DATA"{+}",7                                       :rem 8
990 DATA"{+}",8                                      :rem 10
1000 DATA"{+}",9                                     :rem 42
1010 DATA"{+}",10                                    :rem 83
1020 DATA"END"                                       :rem 248
```

64 Spreadsheet

Michael Tinglof

Ever wanted to calculate your return on various investments, each with several interest possibilities? Or tried to figure the best way to organize your tax deductions, or even your small business operation? If so, you'll find this spreadsheet program invaluable. And it's not expensive like some commercial software. Tape or disk can be used.

Spreadsheet analysis is one of the most common and useful microcomputer applications. With this powerful tool, you can easily evaluate your options and ask *what if?* questions. *VisiCalc* is one of the most widely used and well known spreadsheet programs. Many people have bought a computer just to use this kind of program.

If your budget doesn't enable you to purchase a comprehensive package like *VisiCalc*, but you'd like to do simple financial models, then you'll find this spreadsheet program will fit your needs. It's useful for small spreadsheet problems, such as for a small business or the home.

Spreadsheet Analysis

What is spreadsheet analysis? Basically, it's a program that enables you to set up a financial model in which you can simulate your options. This usually involves setting up a table of numbers with defined interrelationships. Once set up, you can experiment with *what if?* questions by altering the given values. Based on the defined relationships, the program automatically recalculates all the values in the table.

For example, consider this model of four different investments.

(Note that the yields of the respective options are simply approximations, and the total of the Yield column is meaningless.)

Spreadsheet Model

	Principal	Yield	Net	Earning
Passbook	$10,000	.06	$10,600	$600
Trs. Bill	$8,000	.09	$8,720	$720
U.S. Bond	$11,000	.11	$12,210	$1,120
All Saver	$5,000	.12	$5,600	$600
Total	$34,000	.38	$37,130	$3,130

In applying this model to the spreadsheet program, four columns are defined (Principal, Yield, Net, and Earning), and five rows (Passbook, Treasury Bills, U.S. Bonds, All Saver Certificate, and Total). Several mathematical relationships are defined as well: the Net column is defined as the product of the Yield and the Principal, added to the Principal; the Earnings column is defined as the product of the Yield and Principal; and the Total row is defined as the sum of the numbers in each column.

Once these definitions are set up, you can experiment by simply changing the given values. The spreadsheet program will then automatically recalculate the other values. For instance, if you were to change the Principal in this example, the program would then recalculate the Earnings, the Net, and the Totals.

Operating 64 Spreadsheet

Using the spreadsheet program is not difficult—in most cases the program prompts you for the data it needs. The only complicated part is setting up the model at the start. The best way to learn is by doing, so we'll demonstrate by setting up a working model of the example shown in the figure.

The first step is to type in LOAD and RUN the spreadsheet program. When run, the screen clears and a list of commands appears at the bottom of the screen. This is the normal operating mode. If a model were set up, it would be displayed above the command list, and you could experiment with it by changing values. However, if you're just starting out, or beginning a new model, you need to program a model.

Programming the Model. To do this you must enter the Program mode, which is the first option in the command list. Simply hit the *P* key. The screen again clears, a list of numbers appears on the left side of the screen, and a new list of commands appears in the bottom three rows. The list of numbers on the side are line numbers; on these lines the definition for the model will be stored.

To do this, use the Insert command (the first option now on the screen) by typing *I*. The computer will then ask which line the inserted text should be placed *before*. Type *1*—this line now contains an END statement which should be the last statement of the model definition. Next, the computer asks how many lines are to be inserted; in this case, enter *12*.

Row and Column. Beginning with line 1 and continuing to line 12, the computer will prompt you for each line. In response, type the following lines, which will be explained as we go along.

```
1 NAME RA=PASSBK
```

(The line numbers don't have to be typed; the program supplies them.) This line gives the first row of our example the name PASSBK, representing passbook savings, which will be displayed on the left side of the screen in the normal operating mode. In the designation *RA*, the *R* indicates that we are naming a row, and the *A* indicates the row A, or the first row. Only six characters or less are allowed for a row name. There are 20 rows, each designated by a single letter. Now enter:

```
2 NAME RB=TRSBIL
3 NAME RC=USBOND
4 NAME RD=ALSAV
5 NAME RE=TOTAL
```

These are the same as line 1, except they name rows B, C, D, and E. To name the columns, you could begin by typing:

```
6 NAME C1=PRNCPL,6
```

This names the first column, column 1, PRNCPL. In this case, the *C* indicates that it's a column, and the *1* indicates column 1. In addition to their names, columns must also be given a width—in this example, a width of 6. If no width is given, the program defaults to five characters. Every element in the column, including the column name itself, must have a length equal to or less than the given width.

```
7 NAME C2=YIELD,5
8 NAME C3=NET,6
9 NAME C4=EARN,6
```

These are similar to line 6 except that they name columns 2, 3, and 4, respectively.

DEFINE. The next line might be hard to understand at first. It's the first mathematical expression used, and its purpose is to set the NET column equal to the product of the PRINCIPAL column and the YIELD column added to the PRINCIPAL column. To understand the command, you must understand the designations used. Enter line 10 as follows:

```
1Ø DEFINE @3=@1*@2+@1
```

First of all, every element in the matrix of rows and columns has its own designation, which is simply the element's row letter followed by its column number. That is, element C2 is row C, column 2.

The DEFINE command then sets an element equal to an expression containing other elements, and possibly constants such as 2 or .56. Any of the four basic operations of addition, subtraction, multiplication, and division can be used. However, no hierarchy of operations is followed.

In a definition command, it might be useful to consider more than one element at a time. You can use the @ symbol to do this. It can be used to replace either the column or row designation. For example, in the DEFINE command above, @3 indicates all rows in column 3. Note that in the use of @, each element will still be considered separately. The whole statement can thus be translated to *each row in column three equals each row in column one times each row in column two, plus each row in column one.* Make sure the @ symbol *precedes* the column number and *follows* the row letter.

When this command is processed, the element to which the statement is being assigned is considered first. If an @ sign is present, a loop is executed so that each row or column, depending on the format, is considered one at a time. For example, when executing @3, the program takes each column in turn as the current column, evaluates the expression, and assigns the value to the current column. On the other hand, when an @ designation appears in the expression, it is replaced by the current value. For example, if @1 appeared in the expression, it would be replaced by the current column number. Although this designation might appear complicated

at first, it is really quite simple, yet allows complicated mathematical relationships to be constructed easily. If you don't understand it yet, don't worry. For now, you can make it easier by using only absolute designations, such as A2.

```
11 DEFINE @4=@1*@2
12 DEFINE E@=A@+B@+C@+D@
```

These follow the same rules as the first definition.

If you make a mistake, don't worry. During line entry, the delete key is functional. If you made a mistake on a previous line, use the Change command later to replace it; if you forgot a line, use Insert to put it in the program.

Note that the order in which elements are DEFINEd is important because one definition statement can involve values computed by other definitions.

To experiment with the model once it is set up, return to the normal operating mode by using the *E* key, for Exit option. The screen clears, the columns and rows named are displayed, and zeros are printed. The next step is, of course, to replace the zeros with applicable numbers.

Entering Data. The Change command allows you to change any value on the screen. To use it, simply type *C*. The program will then ask you for the row and column—use the same designations used in the program mode, such as A3, and so on. It will then ask you for the value to be entered into the matrix.

Next, the number will be displayed in the appropriate row and column. However, the other values are not recalculated. Because this spreadsheet program is written in BASIC, it is not very fast—so you can change/input as many values as necessary. When you are finished entering numbers and wish to see the results, use the Redraw screen command by hitting the *R* key. The spreadsheet program will then recompute all the values, as expressed in the DEFINE commands.

The first step is to enter the yield values in column 2. Use the Change command to enter .06 for row A, column 2; .09 for row B, column 2; and so on until all yield values seen in the figure have been entered. Then enter trial investment values into the Principal column (column 1) for each row. Finally, type *R* to have the computer calculate your earnings.

You've just created your first financial model. Experiment with it by changing values with the *C* command and redraw-

ing the screen. Design other models, and implement them using the spreadsheet program.

Saving. There is one final step before you finish using the spreadsheet—saving it. If you wish to use it again, use the Save worksheet command to store it. At a later date you can use the Load worksheet command to restore it.

When you want to save a worksheet, just enter a filename, which can be up to ten characters long. You don't have to specify the disk drive using ,8; just the filename. If you want to load a worksheet, simply type L, and then the filename when the prompt appears.

If you want to save and load worksheets using tape instead of disk, you will have to make a few changes in the 64 Spreadsheet program. Modify the following lines:

```
510 OPEN1,1,1,I$                          :rem 76
610 OPEN1,1,0,I$:IFST<>0THEN670           :rem 50
1610 OPEN1,1,1,I$:FORX=1TO100            :rem 153
1710 OPEN1,1,0,I$:IFST<>0THEN1750        :rem 148
```

Reference Guide

The following is a list of the instructions and a brief description for each.

Change Value. Allows you to change the value of any element of the screen. Uses the standard row/column designation to indicate the desired element.

Redraw Screen. Clears the screen and recalculates every value based on the DEFINE statements.

Save Worksheet. Saves the worksheet, including all entered data and the instructions set in program mode.

Load Worksheet. Loads a worksheet saved by the above command.

Exit. Exits the program to BASIC.

Program Mode. Enters the program mode for which the following commands are used:

- **Insert**—inserts a line(s) into the instruction list starting at the line you select.
- **List**—lists a specified part of the program. Accepts a line number, clears the screen, and lists from the given line to the given line plus 20.
- **Change**—accepts a line number and allows you to reenter that program line.

• **Save**—saves the instruction lines, but not the numbers in the worksheet.

• **Read**—reads an instruction set, saved as above, into memory.

• **Exit**—returns to the normal operating mode.

Instructions in Program Lines

• **NAME**—names a column or row, sets column length. ex: NAME C3=TEST,4

• **DEFINE**—sets an element equal to an expression. ex: DEFINE A3=C3+B1

Plus and Minus

This spreadsheet program will probably be most useful for home and small business applications. Although limited to a table with 20 rows and, depending on column width, 5 or 6 columns, it's a powerful tool. It allows experimentation with financial models, and replaces error-prone and time-consuming paper and pencil exercises. This type of experimentation and recalculation enables you to explore various options and select the best one.

Program Notes

This command will clear the worksheet:

- DEFINE @@=0
- Remember that constants can be used in expressions, as is the zero in the above statement.
- The Load command in the program mode can be used to load a program saved in the normal mode *without* loading the numeric data.
- If the program exits to BASIC type 'GOTO 100' to return without losing your data.

64 Spreadsheet

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

```
10 REM COMMODRE 64 SPREADSHEET                    :rem 61
20 REM                                            :rem 70
30 DIMPC%(110),PC$(110),WK(20,9),RN$(20),CN$(9),CW
   (9),CP(9)                                      :rem 43
40 X=0:SL=0:LL=0:L=0:CC=0:CR=0                    :rem 10
45 R$(1)="DEFINE":R$(2)="NAME":R$(3)="END":PC%(1)=
   3                                              :rem 210
50 S$="{39 SPACES}"                               :rem 93
```

```
60 C$="{HOME}{23 DOWN}"                          :rem 232
65 POKE53280,15:POKE53281,1:PRINT"{5}"  :rem 154
70 PRINT"{CLR}"LEFT$(C$,8)TAB(10)"SPREADSHEET ANAL
   YSIS"                                         :rem 159
80 REM                                           :rem 76
90 PRINTLEFT$(C$,24)TAB(8)"HIT RETURN TO CONTINUE
   {SPACE}";:GOSUB10000                          :rem 230
100 GOSUB5000                                    :rem 214
105 GOSUB6000                                    :rem 220
110 PRINT"{RVS}P{OFF}ROGRAM MODE {RVS}C{OFF}HANGE
    {SPACE}VALUE {RVS}R{OFF}EDRAW SCREEN"  :rem 52
120 PRINT"{RVS}S{OFF}AVE WORKSHEET {RVS}L{OFF}OAD
    {SPACE}WORKSHEET {RVS}E{OFF}XIT"             :rem 81
130 GOSUB10000                                    :rem 5
160 IFX$="R"THEN100                              :rem 53
170 IFX$="E"THENPRINT"{CLR}":END:GOTO100    :rem 17
180 IFX$="S"THEN500                              :rem 60
190 IFX$="L"THEN600                              :rem 55
200 IFX$="P"THEN1000                             :rem 94
210 IFX$="C"THEN400                              :rem 37
220 GOTO105                                      :rem 99
400 GOSUB6000:PRINT"CHANGE WHICH ROW:";:GOSUB10000
    :PRINT                                       :rem 221
410 CR=ASC(X$)-64:IFCR<1ORCR>20THEN105       :rem 8
415 PRINT"CHANGE WHICH COLUMN:";:GOSUB10000:PRINT
                                                 :rem 57
420 CC=VAL(X$):IFCC=0THEN105                     :rem 138
430 PRINT"INPUT VALUE:";:I=20:GOSUB9000:WK(CR,CC)=
    VAL(I$)                                      :rem 225
440 GOSUB5700:GOTO105                            :rem 237
500 GOSUB6000:PRINT"SAVE WORKSHEET AS:";:I=10:GOSU
    B9000:IFI$=""THEN105                         :rem 250
510 OPEN1,8,2,I$+",S,W"                          :rem 197
515 FORX=1TO100:PRINT#1,PC%(X)CHR$(13)CHR$(34)PC$(
    X)CHR$(13);                                  :rem 228
520 IFPC%(X)<>3THENNEXT                          :rem 162
530 FORCR=1TO20:FORCC=1TO9:PRINT#1,WK(CR,CC)CHR$(1
    3);:NEXT:NEXT                                :rem 220
540 PRINT#1,CHR$(13);:CLOSE1:GOTO100         :rem 124
600 GOSUB6000:PRINT"LOAD WORKSHEET:";:I=10:GOSUB90
    00:IFI$=""THEN105                             :rem 88
610 OPEN1,8,2,I$+",S,R":IFST<>0THEN670       :rem 167
620 FORX=1TO100:INPUT#1,X$,PC$(X):T=ST:PC%(X)=VAL(
    X$):IFT<>0THEN670                             :rem 88
630 IFPC%(X)<>3THENNEXT                          :rem 164
640 FORCR=1TO20:FORCC=1TO9:INPUT#1,X$:T=ST:WK(CR,C
    C)=VAL(X$):IFST<>0THEN670                    :rem 249
650 NEXT:NEXT:CLOSE1:RC=1:GOTO100            :rem 117
670 PRINT"TAPE ERROR.":PRINT"HIT ANY KEY TO CONTIN
    UE ";:GOSUB10000                             :rem 255
```

```
680 CLOSE1:RC=1:GOTO 100                                    :rem 134
1000 FL=1                                                   :rem 193
1010 PRINT"{HOME}";:FORX=FLTOFL+20:PRINTS$:PRINT"
     {UP}"X;R$(PC%(X))" "PC$(X):NEXT                        :rem 157
1020 GOSUB6000                                               :rem 9
1030 PRINT"{RVS}I{OFF}NSERT {RVS}L{OFF}IST {RVS}D
     {OFF}ELETE {RVS}C{OFF}HANGE {RVS}S{OFF}AVE
     {RVS}R{OFF}EAD"                                        :rem 34
1035 PRINT"{RVS}E{OFF}XIT"                                  :rem 120
1040 GOSUB10000                                              :rem 54
1050 IFX$="E"THENRC=1:GOTO100                               :rem 205
1060 IFX$="I"THEN1200                                       :rem 142
1070 IFX$="L"THEN1300                                       :rem 147
1080 IFX$="D"THEN1400                                       :rem 141
1090 IFX$="C"THEN1500                                       :rem 142
1100 IFX$="S"THEN1600                                       :rem 151
1110 IFX$="R"THEN1700                                       :rem 152
1120 GOTO1020                                               :rem 192
1200 GOSUB6000:PRINT"INSERT BEFORE LINE:";:GOSUB89
     90:SL=VAL(I$)                                          :rem 132
1210 IFSL=0THEN1020                                          :rem 81
1220 PRINT"NUMBER OF LINES:";:GOSUB8990:N=VAL(I$):
     N1=100-N:IFN=0THEN1020                                 :rem 86
1230 FORX=N1TOSLSTEP-1:PC%(X+N)=PC%(X):PC$(X+N)=PC
     $(X):NEXT                                              :rem 120
1235 N=N-1:FORL=SLTOSL+N:GOSUB8000:NEXT                      :rem 1
1240 GOTO1010                                               :rem 194
1300 GOSUB6000:PRINT"START AT WHICH LINE:";:GOSUB8
     990                                                    :rem 220
1320 IFVAL(I$)=0ORVAL(I$)>80THEN1020                         :rem 61
1330 FL=VAL(I$):GOTO1010                                    :rem 108
1400 GOSUB6000:PRINT"DELETE FROM LINE:";:GOSUB8990
     :SL=VAL(I$):IFSL=0THEN1020                             :rem 172
1410 PRINT"TO LINE:";:GOSUB8990:LL=VAL(I$):IFLL=0T
     HEN1020                                                :rem 219
1420 N=LL-SL+1:FORX=SLTO100-N:PC%(X)=PC%(X+N):PC$(
     X)=PC$(X+N):NEXT                                       :rem 241
1430 GOTO1010                                               :rem 195
1500 GOSUB6000:PRINT"CHANGE LINE NUMBER:";:GOSUB89
     90:L=VAL(I$)                                           :rem 27
1510 GOSUB8000:GOTO1010                                      :rem 68
1600 GOSUB6000:PRINT"SAVE PROGRAM AS:";:I=10:GOSUB
     9000:IFI$=""THEN1020                                   :rem 181
1610 OPEN1,8,2,I$+",S,W":FORX=1TO100                         :rem 18
1620 PRINT#1,PC%(X)CHR$(13)CHR$(34)PC$(X)CHR$(13);
                                                            :rem 247
1630 IFPC%(X)<>3THENNEXT                                    :rem 213
1640 CLOSE1:GOTO1000                                        :rem 166
```

```
1700 GOSUB6000:PRINT"LOAD PROGRAM:";:I=10:GOSUB900
     0:IFI$=""THEN1020                          :rem 19
1710 OPEN1,8,2,I$+",S,R":IFST<>0THEN1750        :rem 9
1720 FORX=1TO100:INPUT#1,X$,PC$(X):PC%(X)=VAL(X$)
                                                :rem 133
1730 IFPC%(X)<>3THENNEXT                        :rem 214
1740 CLOSE1:GOTO1000                            :rem 167
1750 PRINT"TAPE ERROR":PRINT"HIT ANY KEY TO CONTIN
     UE ";:GOSUB10000                           :rem 1
1760 CLOSE1:GOTO1000                            :rem 169
4999 END                                        :rem 182
5000 IFRC=0THEN5400                             :rem 78
5005 FORX=1TO9:RN$(X)="":CN$(X)="":CW(X)=0:CP(X)=0
     :NEXT                                      :rem 218
5007 FORX=10TO20:RN$(X)="":NEXT                 :rem 79
5010 RC=0:FORL=1TO100:IFPC%(L)=3THEN5310        :rem 93
5020 IFPC%(L)<>2THEN5300                        :rem 78
5030 X$=LEFT$(PC$(L),1):N$=MID$(PC$(L),2,1):rem 25
5040 IFX$="C"THEN5100                           :rem 141
5050 IFX$="R"THEN5200                           :rem 158
5060 GOTO5900                                   :rem 210
5100 FORX=4TOLEN(PC$(L)):IFMID$(PC$(L),X,1)<>","TH
     ENNEXT:LN=5:GOTO5120                       :rem 195
5110 LN=VAL(MID$(PC$(L),X+1))                   :rem 85
5120 N=VAL(N$):IFN=0THEN5900                    :rem 122
5130 CN$(N)=MID$(PC$(L),4,X-4):CW(N)=LN         :rem 140
5140 GOTO5300                                   :rem 203
5200 N=ASC(N$)-64:IFN<1ORN>20THEN5900           :rem 147
5210 RN$(N)=MID$(PC$(L),4)                      :rem 107
5300 NEXTL                                      :rem 83
5310 CP(1)=0:FORX=2TO9:CP(X)=CP(X-1)+CW(X-1)+ABS(C
     W(X)>0):NEXT                               :rem 164
5400 PRINT"{CLR}{RVS}";:FORX=1TO9:PRINTTAB(CP(X)+8
     )RIGHT$(S$+CN$(X),CW(X));                  :rem 104
5410 NEXT:PRINT                                 :rem 208
5420 FORX=1TO20:PRINT"{RVS}"LEFT$(RN$(X),6):NEXT
                                                :rem 130
5450 FORL=1TO100:IFPC%(L)=3THEN5700             :rem 44
5460 IFPC%(L)<>1THEN5690                        :rem 97
5470 X$=LEFT$(PC$(L),2):AC=VAL(RIGHT$(X$,1)):IFAC<
     1ORAC>9THENAC=0                            :rem 76
5480 AR=ASC(X$)-64:IFAR<1ORAR>20THENAR=0        :rem 168
5490 CC=AC:IFAC=0THENRC=1:FORCC=1TO9            :rem 246
5495 CR=AR:IFAR=0THENRR=1:FORCR=1TO20           :rem 111
5500 IFMID$(PC$(L),3,1)<>"="THEN5900            :rem 176
5510 X$=MID$(PC$(L),4):OP$="+":S=1:T=0          :rem 38
5520 FORX=STOLEN(X$):N$=MID$(X$,X,1):IFN$="+"ORN$=
     "-"ORN$="/"ORN$="*"THEN5550                :rem 166
5540 NEXTX                                      :rem 101
```

```
5550 N$=MID$(X$,S,X-S):S=X+1                    :rem 74
5560 IFN$="Ø"ORVAL(N$)<>ØTHENV=VAL(N$):GOTO561Ø
                                                 :rem 26
5565 IFLEN(N$)<>2THEN59ØØ                      :rem 175
5570 TR$=LEFT$(N$,1):TC$=RIGHT$(N$,1)           :rem 59
5580 TR=ASC(TR$)-64:IFTR<1ORTR>2ØTHENTR=CR:rem 168
5590 TC=VAL(TC$):IFTC<1ORTC>9THENTC=CC         :rem 155
5600 V=WK(TR,TC)                               :rem 186
5610 IFOP$="+"THENT=T+V                         :rem 95
5620 IFOP$="-"THENT=T-V                        :rem 100
5630 IFOP$="/"THENT=T/V                        :rem 105
5640 IFOP$="*"THENT=T*V                         :rem 96
5650 IFX<LEN(X$)THENOP$=MID$(X$,X,1):GOTO552Ø
                                               :rem 185
5660 WK(CR,CC)=T                               :rem 156
5670 IFRR=1THENNEXTCR                          :rem 118
5680 IFRC=1THENNEXTCC                           :rem 89
5690 RR=Ø:RC=Ø:NEXTL                           :rem 230
5700 PRINT"{HOME}{DOWN}";:FORCR=1TO2Ø:IFRN$(CR)=""
     THEN5795                                  :rem 34
5710 FORCC=1TO9:PRINTTAB(CP(CC)+8);            :rem 123
5720 X$=MID$(STR$(WK(CR,CC)),2)                :rem 220
5730 IFLEN(X$)<=CW(CC)THEN5790                 :rem 248
5740 FORX=1TOLEN(X$):IFMID$(X$,X,1)<>"."THENNEXT:G
     OTO5790                                   :rem 227
5750 N$=LEFT$(X$,X-1)                          :rem 126
5760 IFLEN(N$)<CW(CC)THENN$=N$+MID$(X$,X,CW(CC)-LE
     N(N$))                                    :rem 230
5780 X$=N$                                     :rem 255
5790 PRINTRIGHT$(S$+X$,CW(CC));:NEXTCC          :rem 74
5795 PRINT:NEXTCR                              :rem 117
5800 RETURN                                    :rem 173
5900 GOSUB6ØØØ:PRINT"SYNTAX ERROR IN LINE"L
                                               :rem 155
5910 PRINT"HIT ANY KEY TO CONTINUE ";:GOSUB1ØØØØ
                                                :rem 68
5920 RETURN                                    :rem 176
6000 PRINTLEFT$(C$,22)S$:PRINTS$:PRINTS$:PRINTLEFT
     $(C$,22);                                 :rem 118
6010 RETURN                                    :rem 167
8000 GOSUB6ØØØ:PRINTL;                          :rem 92
8010 I=35:GOSUB9ØØØ:IFI$=""THEN8ØØØ           :rem 232
8020 FORX=1TOLEN(I$):IFMID$(I$,X,1)<>" "THENNEXT
                                                :rem 73
8030 PC$(L)=MID$(I$,X+1):PC%(L)=Ø              :rem 244
8040 X$=LEFT$(I$,X-1):FORX=1TO3:IFX$=R$(X)THENPC%(
     L)=X                                      :rem 235
8050 NEXT:RETURN                                :rem 38
8990 I=3:GOSUB9ØØØ                              :rem 22
```

```
8992 IFVAL(I$)<1ORVAL(I$)>100THENI$=""     :rem 167
8995 RETURN                                :rem 191
8999 I=3                                   :rem 156
9000 I$="":POKE204,0:POKE205,20            :rem 160
9010 GETX$:IFX$=""THEN9010                 :rem 229
9020 C=ASC(X$):IFC=20THEN9060              :rem 152
9025 IFC=13THENPOKE204,1:PRINT" ":RETURN   :rem 185
9030 IF(C>31ANDC<95)OR(C>192ANDC<219)THEN9040
                                           :rem 74
9035 GOTO9010                              :rem 212
9040 IFLEN(I$)=ITHEN9010                   :rem 120
9050 PRINTX$;:I$=I$+X$:GOTO9010             :rem 71
9060 IFI$=""THEN9010                        :rem 69
9070 PRINTX$;:I$=LEFT$(I$,LEN(I$)-1):GOTO9010
                                           :rem 105
10000 POKE204,0:POKE205,20                 :rem 160
10010 GETX$:IFX$=""THEN10010                :rem 53
10020 POKE204,1:PRINTX$;:RETURN            :rem 173
```

2

Kid Stuff—Educational Games

Educational Games:

A Kid's View

Kevin Dewey

Here's a kid's-eye view of educational computer games—what they should do, how they should teach, and why they should entertain. The writer concludes his article by presenting "BLAM!," a game for the Commodore 64 that demonstrates his concepts. A joystick is required.

Have you ever tried to write an educational game? If you have, chances are you found it pretty hard. Sure, it's easy to make a simple addition and subtraction program, but education doesn't stop there.

There are many other areas to cover. I know. I'm only 12 and in the seventh grade. We have computers in our school and a variety of educational games. But, unfortunately, some of the games aren't too good. The main flaw that I see in them, and a lot of my classmates agree, is that they are too easy.

Take, for instance, a math program we had last year. There was only one skill level, and it was just basic multiplication with zeros on the end of the numbers to make it seem harder. The game itself had a very good concept but didn't teach you a thing (unless you're in the third grade, and the game was supposed to be sixth-grade level).

Educational Guidelines

Now, if that is what comes from experts, how are ordinary people supposed to write good educational games? Programmers should keep in mind the following things:

1. You should make your game one that teaches someone

something. After you've thought of your idea, ask yourself, "Is this *truly* educational or just a near miss?" This will help very much.

2. Your game should have varying skill levels. It should have levels to challenge the slowest to the fastest student.

3. Use good graphics so your game will be appealing to look at.

4. Have good sound effects. It's good for the player to get a rewarding sound or song if he or she is correct.

5. Most of all, make your game interesting and fun. How many kids want to sit and play a boring game, no matter how educational it is? Not many. It's good, in some cases, to make your game half-arcade and half-educational.

Those are the five essential elements of good educational games. Try to include them when writing one.

Now, here's a game I've made. I call it "BLAM!" It's educational and fun, and I hope you enjoy it.

Game Description

BLAM! is a half-arcade and half-educational game. You must maneuver your player around a building filled with bombs, while trying to disarm all the explosives. You move your player with the joystick and, once you've run into a bomb, disarm it with the keyboard.

You disarm bombs as follows: there is a number at the top of the screen next to the time clock. When you run into a bomb, another number appears at the bottom of the screen, under the blue line. You subtract this number from the one at the top and type your answer. If you're correct, the bomb disappears and you have one less bomb to disconnect. But if you subtract wrong, the bomb explodes! You can survive the explosions, but after three, the whole place falls apart. When you give a wrong answer, the correct answer appears at the top of the screen.

You get only five minutes to clear each story of bombs, because they are time bombs. When you clear a story, you go on to the next, which has ten more bombs than the one before. There are six stories in the building and, if you clear them all, you win the game.

There are also variable skill levels. At the beginning of the game, you choose a skill level from 1 to 100. Skill level 1 uses only numbers through 100, level 2 uses numbers through

200, and so on. Only very, very smart people should play on level 100.

Ways to Change BLAM!

You can raise the possible skill levels by changing the 100's in lines 5 and 6. You can vary the number of stories in the building by changing the 70 in line 131 to the number of stories you want multiplied by ten, plus ten. For example, to make a four-story building, change the 70 to 50.

Blam!

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

```
2 POKE53281,4:POKE53280,14                          :rem 192
3 GOTO500                                            :rem 1
4 SC=53281:BO=53280:POKESC,1:POKEBO,10:PRINT"{CLR}
  {9 DOWN}"TAB(15)"{RVS}{RED}SKILL LEVEL"           :rem 37
5 PRINT"{DOWN}"TAB(15)"(1-100) ";:INPUT A           :rem 111
6 IFA<1ORA>100THEN4                                 :rem 135
7 PRINT"{2 DOWN}"TAB(11)"USE JOYSTICK PORT 2":FORT
  =1TO2000:NEXT:Y=RND(0):B=A*100:H=10               :rem 151
8 W=54272:FORT=WTOW+24:POKET,0:NEXT:POKEW+24,15:PO
  KEW+5,17:POKEW+6,241:GOTO25                       :rem 75
9 J=INT(RND(1)*I):PRINT"{HOME}{22 DOWN}{9 RIGHT}("
  ;J;") BLAM NO. ";                                 :rem 229
10 POKE198,0:INPUTK$:K=VAL(K$)                      :rem 44
11 IFK+J=ITHENPRINT"{RVS}{DOWN}{15 RIGHT}CORRECT!!
   {OFF}";                                          :rem 46
12 IFK+J<>ITHENPRINT"{DOWN}{16 RIGHT}{RVS}WRONG...
   ";:GOTO80                                        :rem 149
13 POKEC,32:M=M+1:IFM=HTHEN110                      :rem 48
14 FORT=1TO25:POKEW,71:POKEW+1,71:POKEW+4,33:FORQ=
   1TO50:NEXT:POKEW+4,32:NEXT                       :rem 87
15 FORN=1910TO2015:POKEN,32:NEXTN                   :rem 29
16 GOTO38                                           :rem 11
25 C=1524:D=55796                                   :rem 126
27 PRINT"{CLR}":POKEBO,4:POKESC,1:FORF=1TOH:rem 67
28 G=INT(RND(1)*760)+40:V=PEEK(G+1024):IF(V<>32)OR
   (G=500)THEN28                                    :rem 85
29 POKEG+55296,0:POKEG+1024,66                      :rem 173
30 NEXTF:PRINT"{HOME}{20 DOWN}{BLU}DDDDDDDDDDDDDDD
   DDDDDDDDDDDDDDDDDDDDDDDDDDD";                    :rem 84
34 RESTORE                                          :rem 139
35 FORF=1TO30:READL,Q:POKEW,L:POKEW+1,Q:POKEW+4,17
   :FORT=1TO50:NEXT:POKEW+4,16                      :rem 194
36 POKEW+1,L-20:POKEW,Q:POKEW+4,17:FORT=1TO50:NEXT
   :POKEW+4,16:NEXT                                 :rem 209
37 TI$="000000":I=INT(RND(1)*B):PRINT"{HOME}
   {9 RIGHT}TIMER"                                  :rem 62
```

```
38 POKED,4:POKEC,65                                    :rem 129
40 JS=PEEK(56320):JS=15-(JSAND15):JS=JS+1:REM READ
    JOYSTK                                             :rem 173
41 KD=C:ONJSGOTO51,42,43,51,44,45,46,51,47,48,49
                                                       :rem 87
42 POKEKD,32:C=C-40:D=D-40:GOTO51:REM NORTH:rem 85
43 POKEKD,32:C=C+40:D=D+40:GOTO51:REM SOUTH:rem 90
44 POKEKD,32:C=C-1:D=D-1:GOTO51:REM WEST   :rem 169
45 POKEKD,32:C=C-41:D=D-41:GOTO51:REM NW   :rem 116
46 POKEKD,32:C=C+39:D=D+39:GOTO51:REM SW   :rem 132
47 POKEKD,32:C=C+1:D=D+1:GOTO51:REM EAST   :rem 146
48 POKEKD,32:C=C-39:D=D-39:GOTO51:REM NE   :rem 115
49 POKEKD,32:C=C+41:D=D+41:GOTO51:REM SE   :rem 103
50 POKEKD,32:C=C-40:D=D-40:REM NORTH       :rem 123
51 DV=DV+1:IFDV=10THENPOKEW+4,129:POKEW+4,128:DV=0
                                                       :rem 55
52 IFPEEK(C)=68THENC=C-160:D=D-160         :rem 211
53 IF C<1064 THEN C=C+40:D=D+40             :rem 80
54 IFPEEK(C)=66THEN9                       :rem 194
55 T$=RIGHT$(TI$,3):PRINT"{HOME}{15 RIGHT}";T$;"
   {10 RIGHT}";I                                       :rem 219
56 IFT$>"500"THEN200                        :rem 74
60 GOTO38                                   :rem 10
80 POKEC,67:FORT=100TO1STEP-2:POKEW+1,T:POKEW+4,12
   9:POKED,2                                           :rem 179
81 POKED,5:NEXTT:FORTT=1TO50:NEXTTT:PRINT"{HOME}
   {RVS}{2 RIGHT}CORRECT{OFF} BLAM!{RVS} NO.=";I-J
   :NN=NN+1                                            :rem 213
82 POKEW+4,128:IFNN=3THEN200               :rem 172
83 FORT=1TO4000:NEXTT:PRINT"{HOME}{31 SPACES}"
                                                       :rem 102
84 M=M+1:IFM=HTHEN110                      :rem 251
85 FORN=1910TO2015:POKEN,32:NEXTN:GOTO37     :rem 1
110 PRINT"{CLR}{DOWN}{14 RIGHT}GOOD WORK!!":M=0
                                                       :rem 63
119 ER=28                                  :rem 217
120 FORU=0TO3:POKEW+1,ER*U:POKEW,49:POKEW+4,17:FOR
    T=1TO1000:NEXT:NEXT                    :rem 181
130 POKEW+4,16:H=H+10                      :rem 159
131 IFH=70THEN600                          :rem 213
132 PRINT"{6 DOWN}{3 RIGHT}YOU GOT ALL THE BOMBS O
    UT OF THAT"                            :rem 246
133 PRINT"STORY, BUT THE TERRORISTS PUT EVEN MORE"
                                                       :rem 124
135 PRINT"IN THE NEXT.":PRINT"{4 DOWN}{13 RIGHT}SE
    E YA AGAIN!"                                        :rem 15
139 ER=28                                  :rem 219
140 FORU=0TO3:POKEW+1,ER*U:POKEW,49:POKEW+4,17:FOR
    T=1TO1000:NEXT:NEXT                    :rem 183
145 POKEW+4,16                                          :rem 18
```

```
150 FORI=1TO3000:NEXT:GOTO25                    :rem 237
200 FORT=100TO0STEP-2:POKEW+1,T:POKEW+4,129
                                                :rem 140
202 POKESC,INT(RND(1)*16):POKEBO,INT(RND(1)*16):PR
    INT"{CLR}"                                  :rem 175
204 NEXT:POKEW+4,128                            :rem 187
210 PRINT"{CLR}":POKEBO,0:POKESC,0:PRINT"{5 DOWN}
    {11 RIGHT}{WHT}THE PLACE BLEW UP!"          :rem 130
215 PRINTTAB(14)"GAME OVER!!"                   :rem 142
218 PRINT"{7 DOWN}"TAB(11)"PLAY AGAIN (Y/N)?"
                                                :rem 173
220 GETA$:IFA$=""THEN220                         :rem 75
225 IF A$="Y" THEN M=0:GOTO4                    :rem 247
230 SYS2048                                      :rem 98
400 DATA50,50,50,50,50,50,50,50,50,50,50,50,50,50,
    50,50,50,50,50,50,50,50                     :rem 248
403 DATA50,50,50,50,50,50,50,50                  :rem 13
404 DATA70,70,70,70,70,70,70,70,70,70,70,70,70,70,
    68,66,64,62,60,58,56,54                      :rem 67
405 DATA52,50,48,46,44,42,40,38                  :rem 38
500 PRINT"{CLR}{11 DOWN}"TAB(11)"{WHT}A FEW MOMENT
    S....."                                     :rem 179
505 POKE52,48:POKE56,48:CLR:POKE56334,PEEK(56334)A
    ND254                                       :rem 210
506 POKE1,PEEK(1)AND251:FORN=0TO2047:POKEN+12288,P
    EEK(N+53248):NEXTN                           :rem 84
510 FORF=1TO60:READX:NEXT:FORF=0TO31:READX:POKEF+1
    2808,X:NEXT                                 :rem 196
520 POKE 1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1
                                                :rem 134
523 POKE 53272,(PEEK(53272)AND240)+12           :rem 185
525 DATA56,56,144,254,58,56,40,108,28,16,56,124,25
    4,254,124,56                                 :rem 52
535 DATA215,254,124,255,255,223,147,161         :rem 184
540 DATA255,255,255,255,255,255,0,0             :rem 239
560 GOTO4                                         :rem 8
600 REM YOU WIN!                                :rem 134
605 PRINT"{6 DOWN}{3 RIGHT}YOU CLEARED THE BUILDIN
    G OF BOMBS."                                :rem 123
610 PRINT"{DOWN}{3 RIGHT}YOU ARE A VERY GREAT PERS
    ON.":PRINT"{3 DOWN}"TAB(13)"PLAY AGAIN?"
                                                :rem 135
612 GETA$:IFA$=""THEN612                         :rem 85
620 IF A$="Y" THEN PRINT"{CLR}":M=0:GOTO4       :rem 148
630 SYS 2048                                    :rem 102
```

Wordspell

Richard Herrmann

With your own list of words, you can use "Wordspell" to help your children practice spelling. This educational program for grades 1 through 9 can be used with tape or disk.

"Wordspell" makes good use of what is called the "dynamic keyboard" technique. This allows a program to modify itself as it is running. In Wordspell, the practice spelling words you enter become part of the program. At the beginning of the program, you are prompted to enter 20 words. After the words are entered, the dynamic keyboard routine merges them into the program as line-numbered DATA statements. This permits you to SAVE the program with the words included so they will not have to be reentered for the next practice session.

Once the spelling list is entered, it is presented one word at a time. The words are quickly spelled letter by letter and then disappear. You then type in the word, and you are told if it is correct—or you're shown the correct spelling if it is wrong. At the end of the program, a score is displayed, as well as a list of the misspelled words. The user now has the option of quitting the program, running the same words, or entering new words.

Notes on the Program

REM statements point out major routines.

DATA statements are created as lines 1, 5, 9, 13, and 17.

Main variables are:

A$( ) - DATA array
B$( ) - Create word list array
W$( ) - Misspelled words array
A$ - INPUT of user spelling

With a little work, the program could be modified to accept more or fewer than 20 words.

Timing loops (lines 37 and 46) for viewing letters and

responses may be easily altered to adapt Wordspell to different age groups. My nine-year-old son finds the default values suitable.

Wordspell

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

```
Ø PRINT"{CLR}":PRINTCHR$(14):POKE53280,7:POKE53281
  ,1:GOSUB61                                    :rem 202
21 PRINT"{CLR}{BLK}{4 DOWN}{3 RIGHT}{3 SPACES}CREA
   TE NEW LIST":INPUT"{2 DOWN}{4 RIGHT}{2 SPACES}(
   Y OR N)";R$:IFR$="Y"THEN5Ø                   :rem 103
22 IFR$<>"N"THEN21                               :rem 7
24 DIMA$(19),W$(19)                             :rem 194
25 FORP=ØTO19:READA$(P):NEXT                    :rem 237
26 FORP=ØTO19                                    :rem 25
27 PRINT"{CLR}"                                 :rem 205
28 PRINT"{9 DOWN}"                              :rem 212
29 PRINTTAB(INT(4Ø-LEN(A$(P)))/2)               :rem 132
3Ø GOSUB35                                       :rem 75
31 GOSUB4Ø                                       :rem 72
32 NEXT                                         :rem 164
33 GOTO71                                         :rem 7
34 REM PRINT OUT WORDS                           :rem 95
35 FORX=1TOLEN(A$(P))                           :rem 238
36 PRINTMID$(A$(P),X,1);                        :rem 103
37 FORT=1TO3ØØ:NEXT                             :rem 194
38 NEXT                                         :rem 17Ø
39 RETURN                                        :rem 76
4Ø PRINT"{CLR}":PRINT"{9 DOWN}"                 :rem 108
41 PRINTTAB((INT(4Ø-LEN(A$(P)))/2)-2):INPUTA$
                                                 :rem 93
42 IFA$=A$(P)THENPRINT"{CLR}":PRINTSPC(215)"
   {5 DOWN}CORRECT !":GOSUB9Ø:GOTO46             :rem 18
43 W$(P)="W":GOSUB81                            :rem 126
44 PRINT"{CLR}":PRINT"{4 DOWN}"SPC(17);"WRONG !":P
   RINT"{2 DOWN}"SPC(9)" CORRECT{SHIFT-SPACE}SPELL
   ING{SHIFT-SPACE}IS:"                         :rem 78
45 PRINT:PRINT:PRINT:PRINTTAB(INT(4Ø-LEN(A$(P)))/2
   )A$(P):K=K+1                                 :rem 7Ø
46 FORT=1TO2ØØØ:NEXT                            :rem 241
47 POKE 53280,7                                 :rem 255
48 RETURN                                        :rem 76
49 REM CREATE WORD DATA                          :rem 91
5Ø PRINT"{CLR}":DIMB$(19)                       :rem 254
51 FORI=ØTO19:PRINT"WORD";I+1;:INPUTB$(I):NEXT
                                                :rem 181
52 PRINT"{CLR}{2 DOWN}{WHT}"                    :rem 242
53 FORI=ØTO19STEP4                              :rem 13Ø
```

```
54 PRINT1+I;"DA"CHR$(34)B$(I)CHR$(34);","CHR$(34)B
   $(I+1)CHR$(34);                              :rem 69
55 PRINT","CHR$(34)B$(I+2)CHR$(34);","CHR$(34)B$(I
   +3):NEXT                                     :rem 113
56 PRINT"GOTO1":PRINT"{HOME}"                   :rem 196
57 POKE198,10                                   :rem 202
58 FORI=0TO5:POKE631+I,13:NEXT                   :rem 98
59 END                                           :rem 69
60 REM INSTRUCTIONS                             :rem 255
61 PRINT"{BLK}{3 DOWN}{6 SPACES}USE THIS PROGRAM F
   OR SPELLING":PRINT" PRACTICE.{2 SPACES}WHEN";
                                                 :rem 110
62 PRINT" REQUESTED, ENTER THE":PRINT" SPELLING WO
   RDS AND {RVS}PRESS RETURN{OFF}.{2 SPACES}WHEN"
                                                  :rem 50
63 PRINT" ALL (20) OF THE WORDS HAVE BEEN":PRINT"
   {SPACE}ENTERED, THEY WILL BE PLACED";        :rem 36
64 PRINT" INTO THE":PRINT" PROGRAM AS DATA STATEME
   NTS.{2 SPACES}RE-SAVE-"                      :rem 253
65 PRINT" ING THE PROGRAM AT THE END OF THE":PRINT
   " SESSION WILL SAVE THE";                    :rem 136
66 PRINT" ENTERED":PRINT" WORDS FOR USE AT THE NEX
   T PRACTICE."                                 :rem 224
67 PRINT"{5 DOWN}{12 RIGHT}{RVS}PRESS RETURN{OFF}"
                                                   :rem 0
68 GETR$:IFR$=""THEN68                           :rem 33
69 IFR$=CHR$(13)THENRETURN                      :rem 118
70 GOTO68                                        :rem 14
71 PRINT:PRINT"{CLR}{RVS}MISSPELLED WORDS:{OFF}":P
   RINT:REM PRINT OUT MISSPELLED WORDS,SCORE:rem 5
72 FORP=0TO19:IFW$(P)="W"THENPRINTTAB(4)A$(P)
                                                 :rem 245
73 NEXT                                         :rem 169
74 PRINT"{HOME}{19 DOWN}{RVS}SCORE ="100-K*5
                                                  :rem 69
75 PRINT:PRINT"{3 SPACES}AGAIN ? (Y OR N)       :rem 154
76 GETR$:IFR$=""THENGOTO76                       :rem 88
77 IFR$="Y"THENRUN1                             :rem 162
78 IFR$<>"N"THEN76                               :rem 28
79 POKE36869,240:POKEV,0:POKES,0                :rem 164
80 GOTO59                                        :rem 15
81 PRINT"{CLR}":POKE53280,2:S=54272:FORE=STOS+28:P
   OKEE,0:NEXT                                  :rem 104
83 POKE54296, 15 :POKE54277, 18 :POKE54278, 242
                                                 :rem 116
85 POKE 54276, 33 :POKE 54273, 4 :POKE54272, 48
                                                   :rem 9
87 FORT=1TO 300 :NEXT:POKE54276, 32:FORT=1TO 400 :
   NEXT                                          :rem 92
```

```
89 RETURN:REM{14 SPACES}FORE=STOS+28:POKEE,Ø:NEXT:
   RETURN                                        :rem 83
9Ø S=54272:FORE=STOS+28:POKEE,Ø:NEXT              :rem 1
1ØØ POKE54296, 15 :POKE54277, 42 :POKE54278, 25Ø
                                                 :rem 15Ø
11Ø POKE 54276, 33 :POKE 54273, 23 :POKE54272, 181
                                                 :rem 141
12Ø FORT=1TO 2ØØ :NEXT:POKE54276, 32:FORT=1TO 5ØØ
   {SPACE}:NEXT                                  :rem 128
13Ø FORE=STOS+28:POKEE,Ø:NEXT                    :rem 94
14Ø RETURN                                       :rem 117
```

Munchmath

Gerald R. Anderson

"Munchmath" is a math drill program that entertains as it teaches. Because of its multiple difficulty levels, it is suitable for a wide range of ages.

"Munchmath" presents an arcade-style character that relies on the player's correct answers to math problems to stay ahead of a ghost that is trying to gobble him up.

The program begins by asking for the player's name, the type of problems wanted (addition, subtraction, multiplication, or division), and the starting level of difficulty. Problems are then presented on the screen for the player to answer. Each correct answer scores ten points and moves "Munchie" one step closer to the power prize. The ghost, however, stays in hot pursuit only three steps behind. After 15 correct responses, Munchie eats the power prize and the tables are turned. Munchie chases the ghost across the screen, eventually catching him and scoring a bonus of 100 points. The difficulty level then advances one notch higher and new problems are presented.

The ghost moves into action when the player gives a wrong answer. First, the correct answer is displayed for the player to study. Then the ghost advances one step closer to Munchie. Three incorrect answers and the ghost catches poor Munchie and gobbles him up. This results in a loss of 50 points and a return to the next lowest level of difficulty.

If a *Q* is typed in response to a problem instead of a number, the game stops. A scoreboard is printed which shows the number of problems the player has been given, the number answered correctly, the number answered incorrectly, and the percentage of correct answers. The player may then choose to resume the game or to end play.

The program has been extensively tested by my six- and eight-year-old daughters, as well as the neighborhood children, and its appeal holds up very nicely.

Program Description

Here's a breakdown of the program:

Lines 100–170: Initialization and delay subroutines.

Lines 190–240: Answer-checking.

Lines 260–270: Print titles computer-style.

Lines 290–460: Generate problem and print it in proper format.

Lines 480–540: Ghost catches Munchie. Generate sound effects, subtract 50 points, and reduce difficulty level.

Lines 560–690: Munchie reaches the power prize and chases the ghost. Bonus of 100 points, advance to next level.

Lines 700–730: Move Munchie and ghost.

Lines 740–780: Print level and score. Clear old answer from screen.

Lines 800–880: Print scoreboard at end of game. Restart or end program.

Lines 900–910: Special characters created.

Lines 930–1070: Titles

Lines 1080–1280: Get player's name, choice, and level.

Lines 1300–1410: DATA statements for custom characters.

Munchmath

For mistake-proof entry, be sure to use "Automatic Proofreader," Appendix J.

```
100 POKE56,48:CLR:PRINT"{CLR}":SM=1073:CM=55345:L=
    1:BC=3                                    :rem 142
110 FORI=0TO27:POKE54272+I,0:NEXT:POKE54296,15:POK
    E54277,18:POKE54278,165                    :rem 56
120 S$="{HOME}{21 DOWN}":SF=54272:WV=54276 :rem 67
130 J$="99999999999999999999":P=3:GOTO900
                                              :rem 111
140 :                                         :rem 207
150 FORT=1TO300:NEXT:RETURN                     :rem 8
160 FORT=1TO40:NEXT:RETURN                    :rem 218
170 FORT=1TO90:NEXT:RETURN                    :rem 224
180 :                                         :rem 211
190 D=VAL(AN$):IFASC(AN$)=81ANDPR>1THEN800:rem 247
200 IFINT(D)<>INT(C)THEN230                    :rem 94
210 P=P+1:R=R+1:M=M+1:SC=SC+10:POKESF,223:POKESF+1
    ,29:POKEWV,17                              :rem 67
220 FORT=1TO5:NEXT:POKEWV,16:GOTO700           :rem 65
230 M=M+1:W=W+1:PRINTLEFT$(S$,10)SPC(20-LEN(C$))"
    {RVS}{RED}"C$"{5 SPACES}"                  :rem 46
240 POKESF+1,8:POKESF,100:POKEWV,33:GOSUB150:POKEW
    V,32:GOTO720                              :rem 136
250 :                                         :rem 209
```

```
260 POKESF+1,40:POKEWV,17                          :rem 246
270 GOSUB160:POKEWV,16:GOSUB160:RETURN             :rem 196
280 :                                              :rem 212
290 PR=PR+1:A=INT(RND(1)*5*L)+1                      :rem 3
300 B=INT(RND(1)*5*L)+1:IFB>ATHENA=A+B             :rem 202
310 E=INT(A*B):A$=STR$(A):B$=STR$(B)                :rem 23
320 IFQ=1THENC=A+B:X=43:GOTO360                    :rem 130
330 IFQ=2THENC=A-B:X=45:GOTO360                    :rem 136
340 IFQ=3THENC=A:C$=STR$(C):GOTO410                :rem 110
350 C=E:X=88                                       :rem 156
360 C$=STR$(C):PRINTLEFT$(S$,7)SPC(20-LEN(A$))"
    {RVS}"A                                         :rem 33
370 PRINTLEFT$(S$,8)SPC(18-LEN(B$))"{RVS} "CHR$(X)
    B"{OFF}{DOWN}{3 LEFT}999"                      :rem 176
380 PRINTLEFT$(S$,10)SPC(19-LEN(C$)):GOSUB1230:IFA
    N$=""THEN380                                   :rem 120
390 D=VAL(AN$):GOTO190                              :rem 17
400 :                                              :rem 206
410 PRINTLEFT$(S$,12)SPC(16)"{9 SPACES}"           :rem 170
420 PRINTLEFT$(S$,10)SPC(18)"{8 SPACES}"           :rem 171
430 PRINTLEFT$(S$,11)SPC(19)"7777":PRINTSPC(18)"8"
                                                   :rem 109
440 PRINTLEFT$(S$,12)SPC(18-LEN(B$))"{RVS}"B;E
                                                    :rem 70
450 PRINTLEFT$(S$,10)SPC(19-LEN(C$)):GOSUB1230:IFA
    N$=""THEN450                                   :rem 116
460 GOTO190                                        :rem 109
470 :                                              :rem 213
480 POKEWV,17:FORI=4TO33:POKESF+1,I:GOSUB160:NEXT:
    POKEWV,16                                       :rem 54
490 POKECM+P,2:GOSUB160:POKECM+P,5:GOSUB160
                                                   :rem 172
500 POKESF+1,14:POKEWV,33:POKESM+P,69:FORI=1TO230:
    NEXT                                           :rem 154
510 POKESM+P,64:FORI=1TO250:NEXT                     :rem 1
520 POKESM+P,32:POKEWV,32:SC=SC-50:IFSC<0THENSC=0
                                                   :rem 75
530 L=L-1:IFL=0THENL=1                              :rem 54
540 P=3:M=0:PRINT"{CLR}":GOTO1210                   :rem 34
550 :                                              :rem 212
560 POKESM+M-1,32:FORI=1TO6:POKECM+M,3:POKESM+M,60
    :POKECM+P,5                                     :rem 58
570 POKESF+1,14:POKEWV,129                          :rem 47
580 POKESM+P,62:GOSUB160:POKESM+P,58:POKECM+M,6:PO
    KESM+M,61:GOSUB160                             :rem 33
590 POKESM+M,32:POKESM+P,32:POKEWV,128:P=P-1:M=M-1
    :NEXT                                          :rem 197
600 FORI=12TO9STEP-1:POKECM+I,5:POKESM+I,62:POKECM
    +9,6:POKESM+9,60:GOSUB160                      :rem 202
```

```
610 POKESF+1,14:POKEWV,129:POKECM+9,3:POKESM+I,58
                                          :rem 252
620 GOSUB160:POKEWV,128:POKESM+I,32:NEXT  :rem 19
630 FORI=1TO5:PRINT"{HOME}{RVS}{DOWN}{RED}"TAB(15)
    "** 100 **":POKESF+1,15:POKEWV,33     :rem 15
640 GOSUB150:POKEWV,32                    :rem 87
650 PRINT"{HOME}{DOWN}{RVS}"TAB(15)"{9 SPACES}":GO
    SUB150:NEXT:L=L+1                     :rem 168
660 SC=SC+100:P=3:M=0:BC=BC+1:IFBC>31THENBC=7
                                          :rem 164
670 REM SETUP                             :rem 18
680 POKE53280,BC:POKE53281,1:PRINT"{CLR}{BLU}":PRI
    NTLEFT$(S$,3)SPC(9)J$                 :rem 33
690 POKE53272,28:PRINT"{HOME}"SPC(13)"{DOWN}{PUR}?
    ????????????{RVS}{RED}S"             :rem 132
700 POKESM+P-1,32:POKECM+P,5:POKESM+P,59:GOSUB150:
    POKESM+P,58                           :rem 48
710 IFSM+P=SM+18THEN560                   :rem 125
720 POKESM+M-1,32:POKECM+M,2:POKESM+M,61:GOSUB150:
    POKECM+M,6:POKESM+M,60                :rem 232
730 IFPEEK(SM+M)=PEEK(SM+P)THEN480        :rem 80
740 PRINTLEFT$(S$,16)SPC(16)"{RVS}{CYN}LEVEL:"L"
    {BLU}"                                :rem 198
750 PRINTLEFT$(S$,17)SPC(9)J$             :rem 178
760 PRINTLEFT$(S$,19)"{RVS}"SPC(12)N$"'S SCORE:"SC
                                          :rem 67
770 PRINTLEFT$(S$,7)SPC(17)"{4 SPACES}":PRINTSPC(1
    7)"{4 SPACES}":PRINTSPC(13)"{DOWN}{8 SPACES}"
                                          :rem 233
780 GOTO290                               :rem 115
790 :                                     :rem 218
800 POKE53272,21:POKE53280,6:POKE53281,7  :rem 245
810 PRINT"{CLR}{DOWN}{RVS}"SPC(13-LEN(N$)/2)N$"'S
    {SPACE}SCOREBOARD"                    :rem 255
820 PRINTSPC(14)"{2 DOWN}PROBLEMS:"PR-1   :rem 199
830 PRINTSPC(12)"{2 DOWN}{GRN}RIGHT ANSWERS:"R:PRI
    NTSPC(12)"{2 DOWN}{RED}WRONG ANSWERS:"W
                                          :rem 151
835 RP=PR-1:QQ=ABS(R/RP*100):Q1=INT(QQ+.5)  :rem 2
840 PRINTSPC(14)"{2 DOWN}{BLK}GRADE:"Q1"%" :rem 67
850 PRINTSPC(12)"{2 DOWN}PLAY AGAIN (Y/N)?":POKE19
    8,0                                   :rem 141
860 GETA$:IFA$<>"Y"ANDA$<>"N"THEN860      :rem 57
870 IFA$="Y"THENPR=0:R=0:W=0:SC=0:GOTO1100 :rem 20
880 END                                   :rem 119
890 :                                     :rem 219
900 FORF=55TO63:FORI=0TO7:READA:POKEF*8+I+12288,A:
    NEXT:NEXT                             :rem 213
910 FORI=0TO7:POKE32*8+I+12288,0:NEXT     :rem 186
```

```
920 :                                                    :rem 213
930 POKE53281,2:POKE53281,7                              :rem 251
940 PRINTLEFT$(S$,10)SPC(11)"{BLU}M {GRN}U{RED} N
    {SPACE}{BLU}C {BLK}H {GRN}M {RED}A {BLU}T
    {GRN}H"                                              :rem 207
950 POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND25
    1:Z=13312:Y=53248                                    :rem 96
960 FORI=0TO519:POKEI+Z,PEEK(I+Y):NEXT:FORI=664TO6
    71:POKEI+Z,PEEK(I+Y):NEXT                            :rem 68
970 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1
                                                         :rem 143
980 POKE53272,28:PRINTLEFT$(S$,10)SPC(11)"{RVS}
    {BLU}M {GRN}U{RED} N {BLU}C {BLK}H {GRN}M
    {RED}A {BLU}T {GRN}H"                                :rem 231
990 :                                                    :rem 220
1000 READF,G:IFF=-1THEN1040                              :rem 52
1010 POKESF+1,F:POKESF,G:POKEWV,33:GOSUB160:POKEWV
     ,32:GOSUB160                                        :rem 190
1020 GOTO1000                                            :rem 189
1030 :                                                   :rem 254
1040 GOSUB150:FORI=4TO24:PRINTLEFT$(S$,10)SPC(I)"
     {SPACE}{CYN}= {RED}<{2 SPACES}{GRN}:";:GOSUB1
     70                                                  :rem 7
1050 PRINTLEFT$(S$,10)SPC(I)" {BLU}< {PUR}=
     {2 SPACES}{GRN};"                                   :rem 72
1060 POKESF,195:POKESF+1,17:POKEWV,17:GOSUB170:POK
     EWV,16:NEXT                                         :rem 106
1070 PRINTLEFT$(S$,10)SPC(24)"{8 SPACES}"                :rem 218
1080 POKE53280,5:POKE53281,7:POKE53272,21                :rem 37
1090 PRINT"{CLR}"SPC(8)"{3 DOWN}{BLU}WHAT IS YOUR
     {SPACE}NAME";:GOSUB260:INPUTN$                      :rem 60
1100 PRINT"{CLR}{BLU}"SPC(13)"{5 DOWN}WHAT WOULD Y
     OU":GOSUB260                                        :rem 135
1110 PRINTSPC(11)"{DOWN}LIKE TO PRACTICE,{DOWN}":G
     OSUB260                                             :rem 224
1120 PRINTSPC(20-LEN(N$)/2)N$":":GOSUB260                :rem 92
1130 PRINTSPC(14)"{DOWN}{RED}1){GRN}ADDITION":GOSU
     B260                                                :rem 117
1140 PRINTSPC(14)"{DOWN}{RED}2){GRN}SUBTRACTION":G
     OSUB260                                             :rem 121
1150 PRINTSPC(14)"{DOWN}{RED}3){GRN}DIVISION":GOSU
     B260                                                :rem 146
1160 PRINTSPC(14)"{DOWN}{RED}4){GRN}MULTIPLICATION
     {BLU}":GOSUB260                                     :rem 124
1170 GETA$:Q=VAL(A$):IFQ<1ORQ>4THEN1170                  :rem 82
1180 PRINTLEFT$(S$,Q*2+10)SPC(14)"{RVS}"MID$(STR$(
     Q),2)                                               :rem 49
1190 PRINTLEFT$(S$,20)SPC(14)"LEVEL (1-9)?"
                                                         :rem 124
```

```
1200 GETA$:L=VAL(A$):IFL<1ORL>9THEN1200     :rem 60
1210 GOTO680                                :rem 155
1220 :                                      :rem 255
1230 PRINT"{RVS}? ";:AN$="":POKE198,0       :rem 248
1240 GETZA$:IFZA$=""THEN1240                :rem 101
1250 ZL=LEN(AN$):IFZA$=CHR$(20)ANDZLTHENPRINTZA$;:
     AN$=LEFT$(AN$,ZL-1)                    :rem 227
1260 IFZA$=CHR$(13)THENPRINT:RETURN         :rem 224
1270 IFZA$<>"Q"AND(ZA$<"0"ORZA$>"9")ORZL=5THEN1240
                                            :rem 132
1280 PRINTZA$;:AN$=AN$+ZA$:GOTO1240          :rem 83
1290 :                                        :rem 6
1300 DATA 0,0,0,0,0,0,255,255               :rem 106
1310 DATA 3,3,3,3,3,3,3,3                   :rem 171
1320 DATA 0,0,255,255,255,255,0,0            :rem 68
1330 DATA 24,60,110,126,126,126,60,24        :rem 10
1340 DATA 56,124,95,248,224,248,127,56       :rem 95
1350 DATA 60,126,255,219,255,255,169,169    :rem 198
1360 DATA 60,126,255,219,255,255,90,180     :rem 137
1370 DATA 120,116,30,14,30,124,120,0        :rem 201
1380 DATA 0,0,0,14,14,14,0,0                 :rem 57
1390 :                                        :rem 7
1400 DATA 16,195,22,96,28,49,33,125,33,125,33,125,
     33,125                                 :rem 195
1410 DATA 28,49,28,49,28,49,22,96,28,49,22,96,16,1
     95,-1,0                                :rem 10
```

3

Sound

Working with SID

Jerry M. Jaco

In this unique approach to the Commodore 64's SID chip, the author discusses the SID chip's anatomy and capabilities in the context of its essential similarity to the design of music synthesizers.

If you've decided you want to make music on your Commodore 64, but don't know where to begin, perhaps a look at how an analog synthesizer is used in an electronic music studio will clarify many aspects of the 64's amazing sound capabilities. Once we have covered the physical aspects of a synthesizer, we can begin to understand some of the techniques used to create sounds artificially.

Electronic music studios usually have at least one analog synthesizer. Most synthesizers have a modular design which allows the synthesizer to be built and expanded according to the dictates of budget, space, and ability. Each module on the synthesizer has a different function and the builder-user is free to duplicate or omit any of them.

Each module on the synthesizer is independent of all others. The only way to connect them is either by a panel of fancy selector switches or via the more common *patch cords.* Patch cords are simply pieces of electrical cable of varying lengths which have standard plugs attached on each end. Plugging one end of a patch cord into the output socket of one module and the other end of the patch cord into the input socket of another module creates an electrical pathway called a *patch.*

If a patch leads from a source module, such as an oscillator, to an output module, such as a mixer, the resulting sound will be audible to the outside world. (See Figure 1.) The term *signal* is used to describe the electrical current being passed from one module to another. A *source signal* is one that will eventually be heard as a real sound. A *control signal* is a varying voltage used to electronically control another module. It does not contain sound information per se.

Figure 1. Processing a Single Source Signal

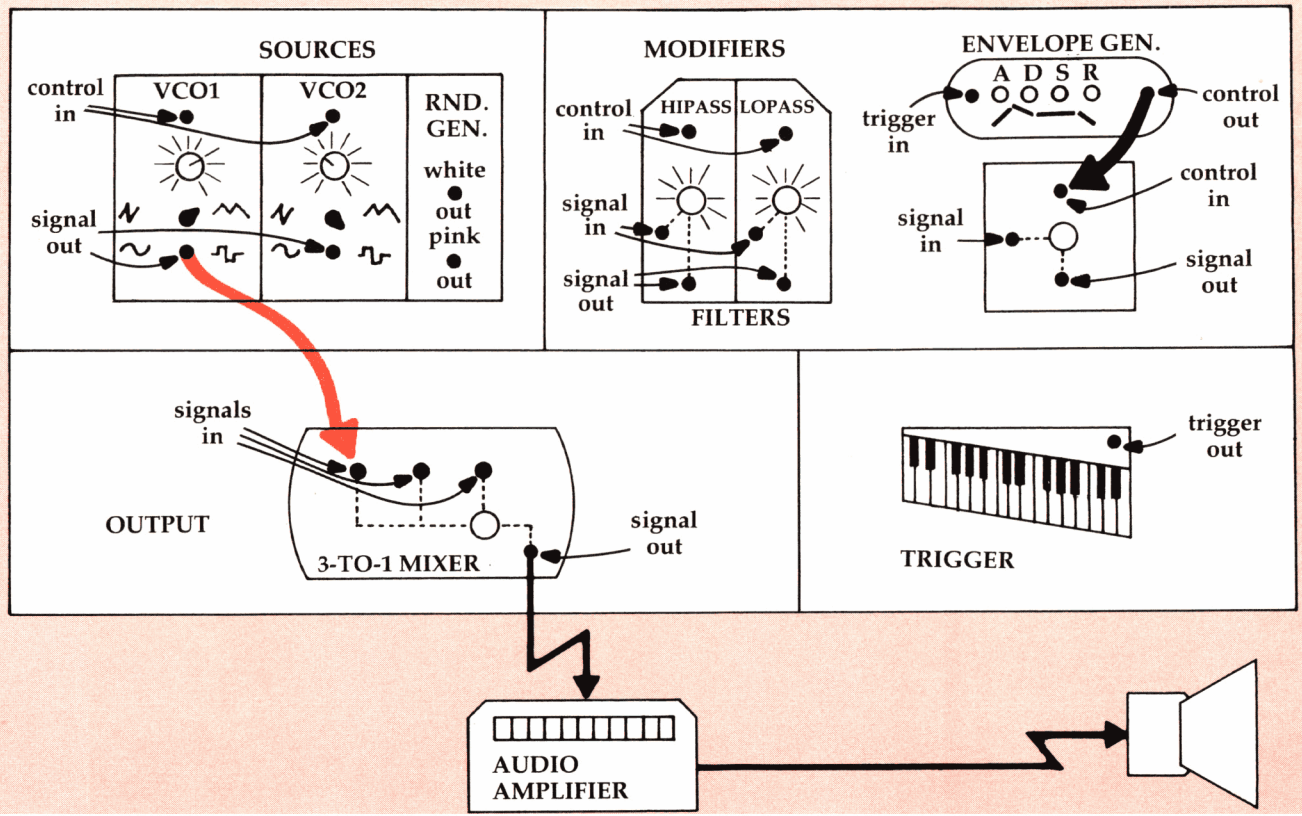
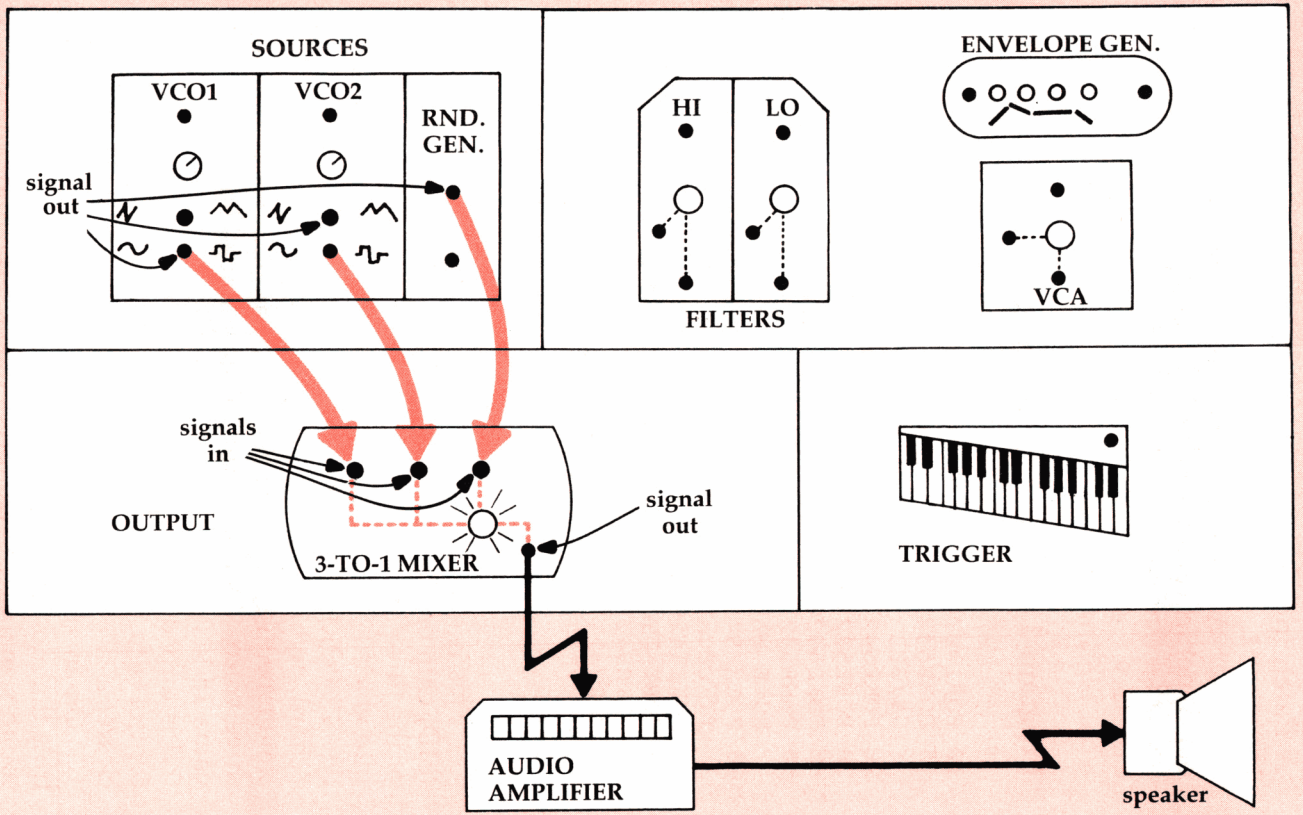


Figure 2. Processing Three Source Signals



A Patch for the SID Chip

In Figure 1, there is only one source signal being processed by the mixer. A mixer can handle up to three source signals on our hypothetical system, which it combines into one composite signal that gets sent on to the speakers, and so to your ears. (See Figure 2.) On the 64, Program 1 accomplishes exactly what the analog synthesizer does in Figure 2.

Program 1. Three Voices—or a Chord

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

```
10 FORI=0TO24:POKE54272+I,0:NEXT                :rem 189
15 POKE54278,249:POKE54285,249:POKE54292,249:REM S
   US/REL VALUES FOR ALL OSC'S                  :rem 250
20 POKE54272,37:POKE54273,17:REM OSC1            :rem 97
30 POKE54279,229:POKE54280,22:REM OSC2           :rem 151
40 POKE54286,214:POKE54287,28:REM OSC3           :rem 158
50 POKE54276,17:POKE54283,17:POKE54290,17:REM TRIA
   NGLE WAVE FOR ALL OSC'S                      :rem 250
60 POKE 54277,17:POKE 54284,17:POKE 54291,17:REM A
   TT/DECAY VALUES FOR ALL OSC'S                :rem 195
70 POKE54296,15:REM MASTER VOLUME ON            :rem 145
75 FORT=1TO500:NEXT:REM CHORD DURATION          :rem 186
80 POKE54276,16:POKE54283,16:POKE54290,16        :rem 52
90 FORT=1TO450:NEXT:REM REL. DURATION            :rem 92
95 POKE54296,0:REM TURN OFF VOLUME               :rem 29
96 END                                           :rem 70
```

This is a very basic patch for the Sound Interface Device (SID) chip on the 64. Lines 20, 30, and 40 set the frequencies of the three oscillators. Line 20 POKEs the values for middle C into voice 1. Line 30 POKEs the values for F into voice 2, and line 40 POKEs the values for A into voice 3. This gives us a "chord," which is simply three notes (voices) sounding simultaneously. Line 50 selects a triangle wave output for all three voices. Line 70 is the mixer volume control. When the value 15 is POKEd into this location, the master volume control is turned all the way up. When 0 is POKEd, the volume control is turned off, as in line 95. The other lines will become clearer as we go along.

On an analog synthesizer, *pots* (potentiometers) are controls that do things such as raise and lower the volume of a sound signal or change the frequency (pitch) of an oscillator. Pots are also the main components of game paddles and TV

volume controls. To make new sounds on an analog synthesizer, the user twists pots on each module and listens for the resulting effect. When the desired sound is found, it can be recorded on tape or the patch written down on a patch chart, marking the pathways made by the patch cords and the positions of the pots for future reference. Analog synthesizers are very useful in this way because drastic changes in a sound can be quickly made by simply twisting a knob or plugging a patch cord into something else.

Turning Knobs with POKEs

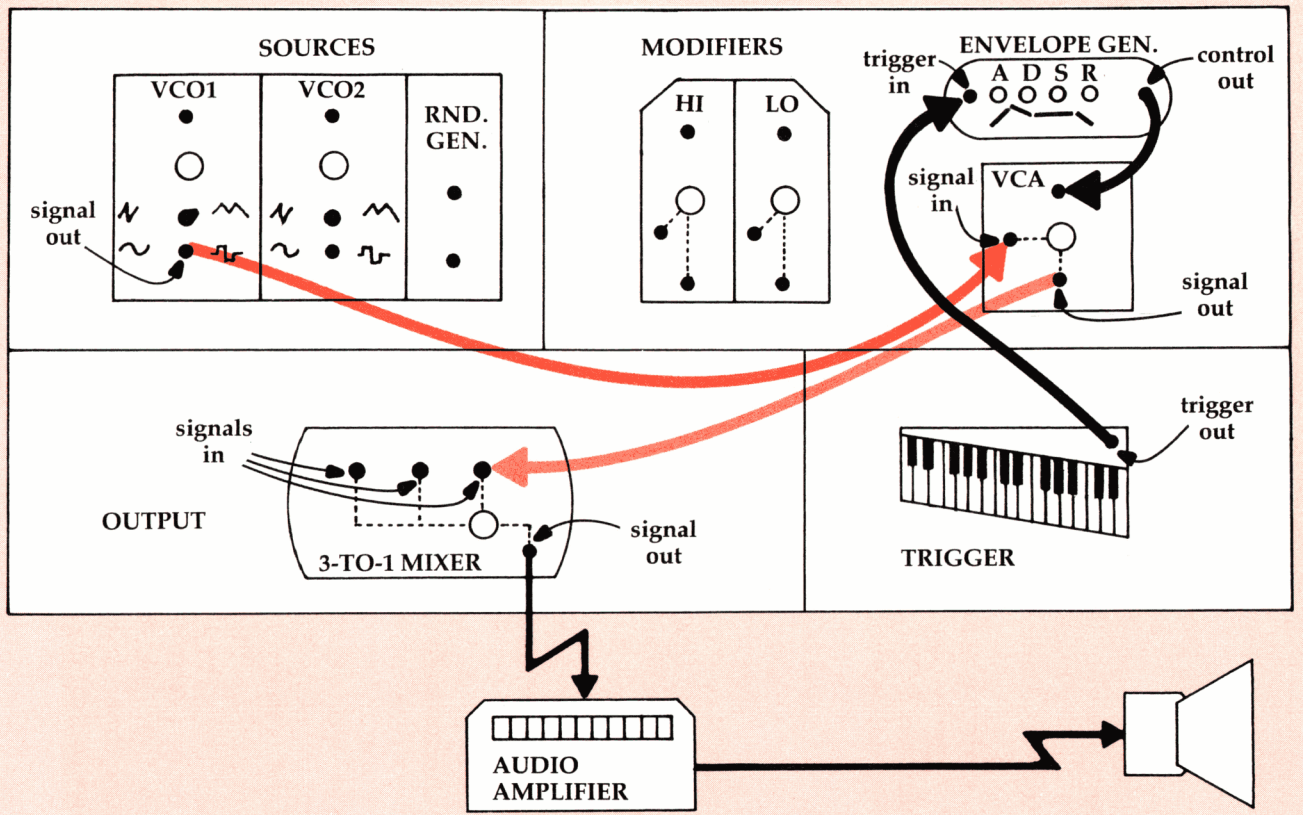
A digitally-controlled synthesizer, such as our SID chip, uses numbers POKEd into control registers to accomplish the same things that knob-twisting and patch-cord-plugging do on an analog synthesizer. For example, if you POKE a 16-bit value into the first two registers of the SID chip (54272 and 54273), you've set the frequency value for oscillator 1. POKE a four-bit number into the high nybble of the sixth register on the chip, and you've set the attack value of the envelope for oscillator 1. POKEing different values into other registers will activate them in the same way that turning the pots or setting switches will activate the analog synthesizer modules.

Envelope Generation

Look at Figure 1 again. It shows a direct path from a voltage-controlled oscillator (VCO1) to the mixer. If we were to break that path, sending the output of VCO1 to the input of the amplifier module (VCA), we would then need to send the output of VCA to the mixer so that the sound from VCO1 could still be heard. The patch shown in Figure 3 would be the result. Now we can make VCO1 signal even louder by adjusting the pot on VCA or on the mixer. The real reason for taking this route is that the envelope generator can be brought into play, since it directly controls the VCA.

There are four pots on the envelope generator module. The first controls the attack time; the second, the decay time. The third sets the sustain level, and the fourth controls the release time. On the SID, two registers in high-low nybble format control these functions. Perhaps the most important function is the sustain level. It's not a timing value, but rather the level at which the amplifier's volume control is set while the note is being sounded. If the sustain level is zero, no

Figure 3: Using the VCA



sound will be heard after the attack and decay phases have ended.

The envelope generator puts out an electrical signal which tells the amplifier when to turn up the volume and how long it should take, as well as how high to set the volume, and when to turn it all the way off again. This is why the amplifier module in the diagrams is called "VCA." This stands for Voltage Controlled Amplifier and means that the amplifier can be controlled by an incoming variable voltage, such as the one supplied by the envelope generator.

ADSR Values

On the SID chip, each voice has its own envelope generator. Within the group of seven registers (0-6) that control the three oscillators, register 5 contains the attack and decay values in high-low nybble format, and register 6 contains the sustain/release values. All values are four-bit numbers (nybbles). The attack value determines how long the amplifier should take to reach peak amplitude (maximum volume).

The decay value determines how long the amplifier should take to go from peak amplitude to the level specified by the sustain value. The release value is the time the amplifier will use to return to the lowest amplitude level ("off") from the sustain level.

Remember, though, that on the analog synthesizer as well as on the SID chip, the envelope will not go into effect until it is "triggered." The lowest order bit (bit 0), the gate bit, triggers each envelope on the SID chip. On the analog synthesizer, triggering of the envelope is accomplished through the use of an attached keyboard module. When a key is pushed down (and as long as it is held down), the attack, decay, and sustain values will go into effect in order. When the key is released, the release phase is triggered, and the VCA will close down the volume of the signal it is operating on over the length of time specified by the release value.

Program 2 demonstrates the effect of the various ADSR values:

Program 2. Effects of the ADSR Values

```
100 FORI=0TO24:POKE54272+I,0:NEXT          :rem 237
130 POKE54277,240:REM SLOW ATTACK/FASTEST DECAY RA
    TE                                     :rem 132
```

```
140 POKE54278,240:REM HIGHEST SUSTAIN LEVEL/FASTES
    T RELEASE RATE                              :rem 207
145 POKE54272,37:POKE54273,17:REM OSC1          :rem 153
150 POKE54276,129:REM NOISE WAVE OSC1           :rem 122
155 POKE54296,15                                :rem 102
160 FORT=1TO4500:NEXT:REM DURATION FOR ATTACK,DECA
    Y, AND SUSTAIN                                :rem 0
170 POKE54276,128:REM BEGIN RELEASE CYCLE       :rem 138
180 FORT=1TO4500:NEXT:REM REL. DURATION         :rem 188
190 END                                         :rem 113
```

In line 130, the attack value is all the way on, and the decay value is all the way off. In line 140, the sustain value is all the way on, and the release value is off. Each value is a four-bit number, 0 to 15. With the attack and sustain setting, the actual POKE value is shifted to the high nybble; thus, 240 is actually the attack value equal to 15 (for slowest attack) multiplied by 16. The sound generated is a random noise that gradually gets louder and then suddenly stops. It stops suddenly because we have set the release value to 0, allowing no time for a gradual decrease in volume.

Change the value 240 in line 140 to 255 and run the program again. The sound should slowly fade away. The high nybble of 54278 (sustain) is now 240 and the low nybble (release) 15, making a total of 255, the value we just POKEd into 54278. Try lowering the sustain value by two or three (2*16 or 3*16); that is, POKE 54278 with either 223 or 207 and see what happens. The sound should build up as before but should then fall off markedly. Change the decay value from 0 in line 130 to about 8 (POKE 54277,248) and hear how the drop-off is now smoothed out. Similarly, shorten the attack time to vary the start of the sound the same way the sustain value was altered. The results should be vastly different from those we started with, and we've been working with only two registers!

Look now at line 170. Notice that we subtracted one from the value we originally POKEd into 54276 in line 150. This zeros the gate bit in 54276, and it is the same as taking your finger off the keyboard on the analog synthesizer: the release cycle gets triggered. Of course, it works only if the VCA sustain level has been previously raised high enough to hear the tone. The delay loop in line 180 is also necessary to allow the release cycle to reach its lowest level.

For more explanation about the ADSR values, as well as a sound editor program that lets you alter the values and immediately hear the result, take a look at "Sound Editor 64," another article in this book.

Using Filters to Color Sound

Let's add a filter to the path in Figure 3. The path from the VCA to the mixer is broken so that filtering the modulated signal will be more easily heard. In our diagram, we have a choice of a high-pass or low-pass filter. On the SID chip, we can also utilize a band-pass filter.

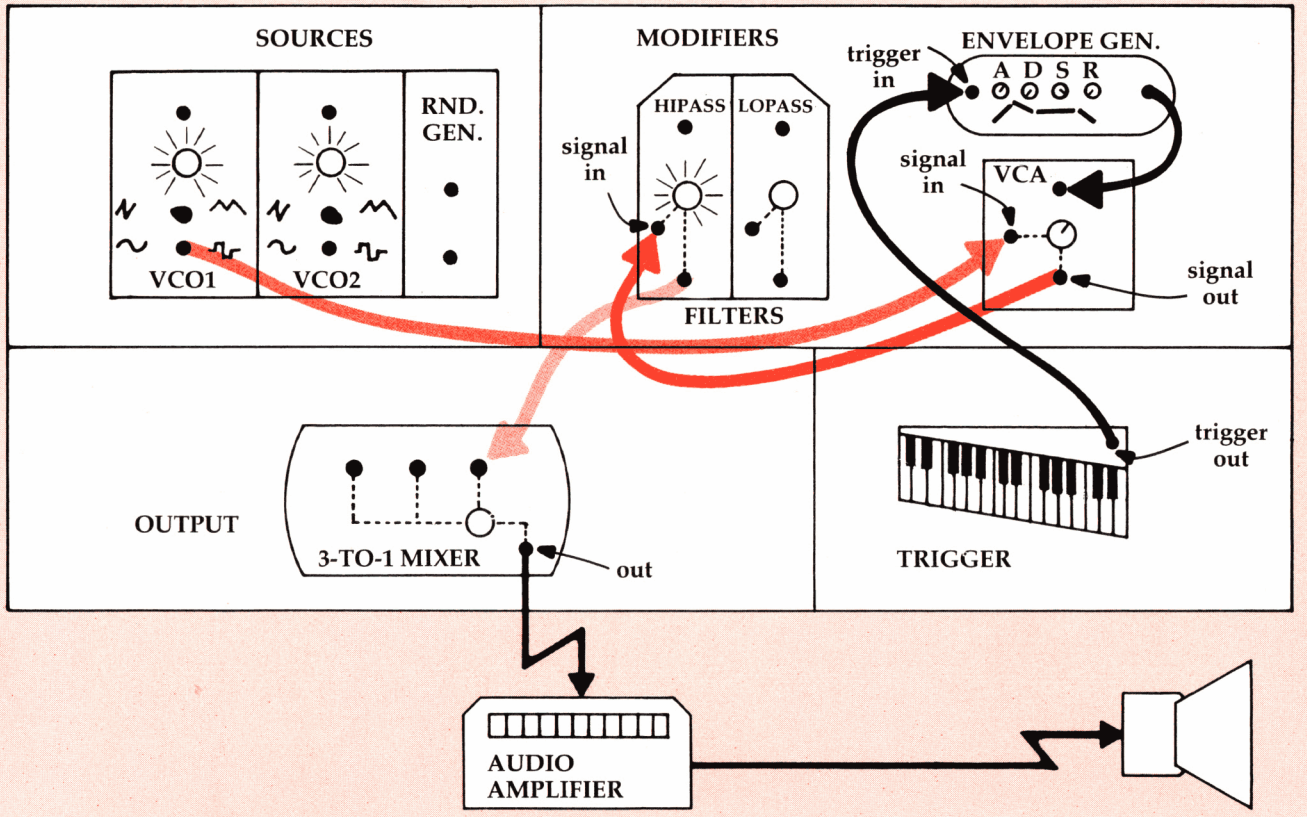
The pot on each filter is used to adjust the cutoff frequency, which is the frequency above which a high-pass filter allows frequencies in the sound spectrum to be heard and below which the filter suppresses them. The low-pass filter is the opposite of the high-pass filter in that it suppresses the frequencies above the cutoff value and allows those below it to sound. A band-pass filter allows frequencies to be heard within a narrow band surrounding the cutoff frequency (called a center frequency in this case), while suppressing all the rest. Use of filters constitutes a technique called *subtractive synthesis,* which selectively eliminates available frequencies of the sound spectrum, producing widely varying sound colors.

Figure 4 indicates that we've decided to filter VCO1 through a high-pass filter. VCO1 is set to produce a sawtooth wave. The path of the patch runs out of VCO1 into the VCA, and from the VCA into the high-pass filter. From there, the signal heads to the mixer and out to the speaker. Program 3 is a routine that does the same thing.

Program 3. Filtered Sound

```
200 FORI=ØTO24:POKE54272+I,Ø:NEXT          :rem 238
210 POKE54272,37:POKE54273,17:REM OSC1     :rem 146
230 POKE54277,120:REM MED. ATTACK/MED. DECAY
                                           :rem 255
240 POKE54278,245:REM HIGHEST SUSTAIN/MED. RELEASE
                                            :rem 27
245 POKE54293,40:POKE54294,5:REM CUTOFF FREQ. FOR
    {SPACE}HIGH-PASS FILTER                :rem 165
250 POKE54295,129:REM MED RES'NCE AND OSC1 TO BE F
    ILTERED                                :rem 212
253 POKE54276,33:REM SAWTOOTH WAVE OSC1     :rem 67
255 POKE 54296,79:REM FULL VOL AND CHOOSE HI-PASS
    {SPACE}FILTER                           :rem 2
```

Figure 4: Using a High-Pass Filter



```
260 FORJ=1TO250:POKE54294,J:NEXT:REM SWEEP CUTOFF
    {SPACE}FREQ. UPWARDS                    :rem 188
270 POKE54276,32:REM{2 SPACES}BEGIN RELEASE CYCLE
                                            :rem 85
280 FORT=1TO500:NEXT:REM REL. DURATION      :rem 137
290 POKE54296,0:REM TURN OFF VOLUME          :rem 74
295 END                                     :rem 119
```

To hear the effect of the filter, we will sweep the value of the cutoff frequency in line 260 from low to high. This will allow less and less of the available sound spectrum to be passed by the filter. Listen carefully to the richness of the tone as it is diminished. Switch the wave form to noise in line 253 by POKEing 129, instead of 33, into 54276 to hear a different version of the effect. Many effects are possible using filters.

Frequency Modulation

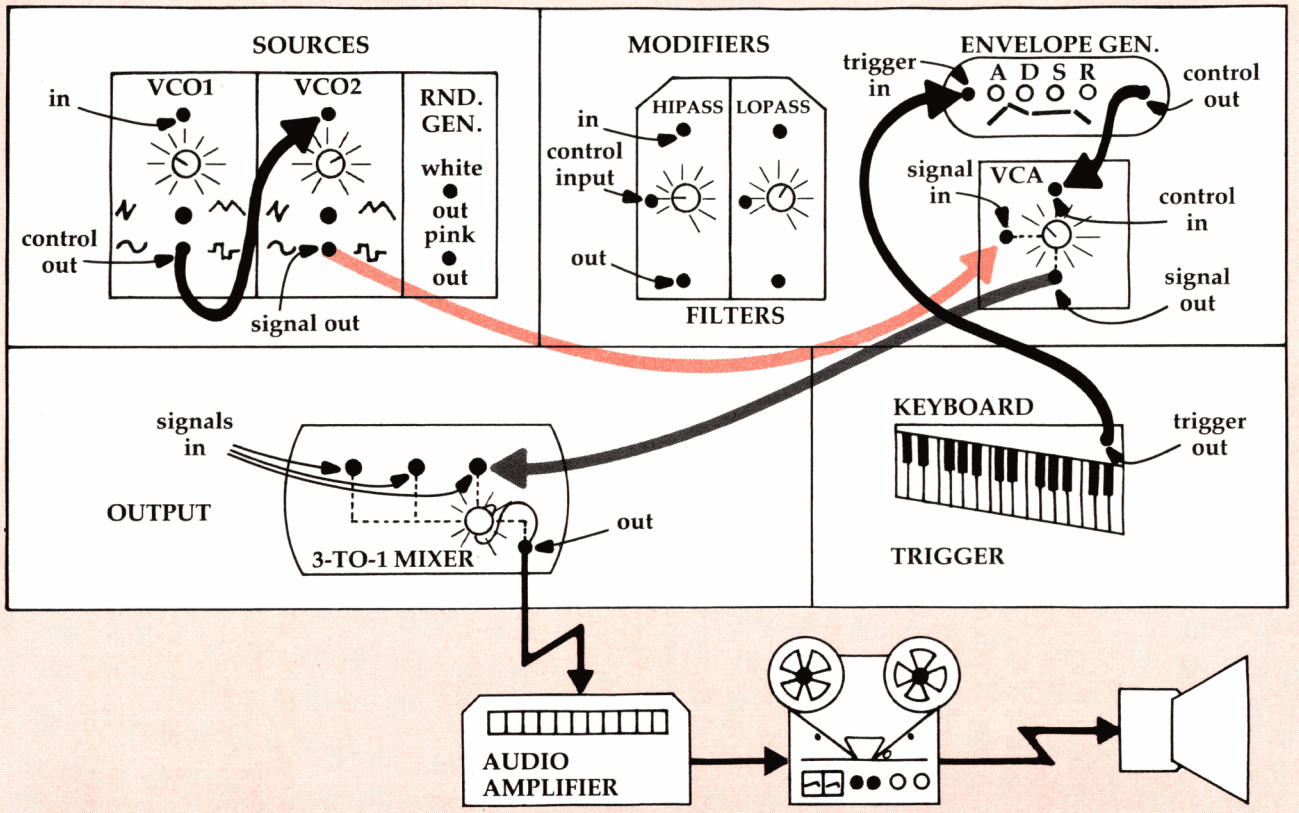
Figure 5 introduces another technique called frequency modulation. Notice now that the signal from VCO1 is entering the control input of VCO2, and that the signal from VCO2 is going through the VCA and on to the mixer. The frequency of VCO2 is now being controlled automatically by the output voltage of VCO1 instead of manually by the pot. This is another example of voltage control. The envelope generator controlled the VCA before and an oscillator now controls a VCO (Voltage Controlled Oscillator).

Frequency Modulation (FM), along with filtering and envelope control, is one of the most significant techniques of sound synthesis. Using one signal source to alter the sound quality of another provides incredibly powerful and varied tools for sound manipulation. Program 4 is one simple example of the FM technique.

Program 4. Siren

```
300 FORI=0TO24:POKE54272+I,0:NEXT            :rem 239
305 POKE 54278,240:REM FULL SUSTAIN/FASTEST RELEAS
    E RATE                                  :rem 129
310 POKE54276,33:REM SAWTOTH WAVE OSC1      :rem 238
320 POKE54286,3:REM CONTROL FREQ. OSC3      :rem 223
330 POKE54290,16:REM TRIANGLE WAVE OSC3      :rem 27
340 POKE54296,175:REM FULL VOL. & SELECT BAND-PASS
     & DISC. OSC3 FROM AUDIO                :rem 157
350 POKE54295,1:REM NO RES'NCE & CHOOSE OSC1 FOR F
    ILTER                                   :rem 121
360 POKE54293,255:POKE54294,78:REM CUTOFF FREQ.
                                            :rem 228
```

Figure 5: Frequency Modulation



```
375 FORT=1TO300                                    :rem 126
380 F=20000+PEEK(54299)*20:REM ADD OSC3 OUTPUT TO
    {SPACE}BASE FREQ.                              :rem 84
390 HF=INT(F/256):LF=F-256*HF:REM SPLIT NEW FREQ.
    {SPACE}INTO HIGH/LOW BYTES                     :rem 120
400 POKE54272,LF:POKE54273,HF:REM SET NEW OSC1 FRE
    Q                                              :rem 229
410 NEXT:POKE 54276,32:POKE54296,0                 :rem 165
420 END                                            :rem 109
```

The third oscillator on the SID chip is our control oscillator, as VCO1 is in Figure 5. We get access to a value corresponding to the wave shape of oscillator 3 in register 27 (54299). If oscillator 3 is set to a triangle wave, the value in register 27 will go up from 0 to 255 and then down from 255 to 0 in a symmetrical rhythm.

This is a nice shape for a siren sound, which is what Program 4 creates. Notice that the frequency of oscillator 3 in line 320 is very low. This value allows the tracing of the waveform to be heard as a siren. The range of frequencies under approximately 32 hertz is called the subaudio range and refers to the fact that the actual waveform at these frequencies is discernible as individual pulses instead of as a continuous tone. When oscillator 3's frequency is increased into the audio range (above about 29), the quality of the resulting tone becomes enjoyably less predictable.

Try POKEing 220 into 54286 at line 320 and running the routine. Note how the information in register 27 (54299) is utilized in line 380. It is increased by a factor of 20 and then added to the base frequency of 20000. Program 4 also uses a band pass filter, but for no particular reason other than simply to stick one in. Try a different value for the waveform in line 330. If you use 64 as your value, be sure to add a line to set oscillator 3's pulse width.

Synthesized 64

The techniques of sound manipulation described above, as used with an analog synthesizer, have perhaps given you a better picture of the working of the SID chip. As you learn more about the internal registers which control other functions, you'll discover others just as interesting as those discussed here.

Get a copy of the *Commodore 64 Programmer's Reference Guide* and read about ring modulation, filtering, and other advanced techniques. Sound effects are the most directly useful sound patches to work with at the start. Program 5 is an example of one I used for a Hangman program: it's the sound of nails being driven into wood. Imagine the other sound effects you can create for new game ideas.

The *User's Guide* and *Programmer's Reference Guide* have suggested patches for you to try out. Put some FOR-NEXT loops in, as we did in line 260 of Program 3, to have the computer "adjust the pots" for you, as it alters individual registers. Once you've found a patch you like, save the register values for future reference. As you become more acquainted with the way that sounds can be altered, you will find yourself noticing the subtler shades of sound color. You'll also begin to understand how the sounds on a TV commercial, videogame, or science fiction movie are created.

Program 5. Driving Nails into Wood

```
700 FORI=0TO24:POKE54272+I,0:NEXT          :rem 243
710 CT=0                                   :rem 156
720 POKE54278,5:REM SUSTAIN/RELEASE        :rem 168
730 POKE54277,5:REM ATTACK/DECAY           :rem 158
740 POKE54276,129:REM NOISE WAVEFORM       :rem 157
750 POKE54295,241:REM RES'NCE & VOICE       :rem 56
760 POKE54293,54:POKE54294,28:REM CUTOFF    :rem 84
770 READA:REM INPUT HI BYTE FREQ.VALUE      :rem 71
780 READB:REM INPUT LO BYTE FREQ.VALUE      :rem 83
790 IFB=-1THEN900:REM BRANCH ON END OF DATA
                                           :rem 195
800 POKE54273,A:POKE54272,B:REM SET FREQ.  :rem 122
810 FORT=1TO35:POKE54296,79:NEXT:REM TURN ON VOLUM
    E & FILTER                             :rem 157
820 POKE54276,128:REM RELEASE CYCLE         :rem 39
830 GOTO730:REM GET NEW NOTE               :rem 140
840 DATA17,37,19,63,21,154,22,227,25,177,28,214,32
    ,94,34,175,34,255                       :rem 35
850 DATA -1,-1                             :rem 159
900 CT=CT+1:IFCT+1<6THENRESTORE:FORT=1TO100*CT:NEX
    T:GOTO770                               :rem 67
910 POKE54296,0:REM TURN OFF VOLUME         :rem 73
```

Sound Editor 64

Daniel L. Riegal

The SID chip in the Commodore 64 is certainly versatile, but it can be confusing and difficult to use. Here's a program that will help—"Sound Editor 64." With it, you can experiment with sound on the 64 and even have it write BASIC sound routines for you.

Perhaps the most outstanding, and confusing, feature of the Commodore 64 is the Sound Interface Device (SID). Many sounds can be produced by the SID chip that are not possible on other home computers, with a quality that is truly amazing. However, it takes understanding and patience to coax just the right sounds from the SID.

While other home computers only require settings of frequency and duration to produce sound or music, the Commodore 64 has several parameters used to shape, modulate, and filter the sound. Unless you understand the basics involved in setting these parameters, you can expect little more than pops, clicks, or beeps. "Sound Editor 64" allows input of the various parameters in a straight-forward manner so that you will not only become familiar with them, but will also be able to try various combinations to see the impact that each has on the sound.

Attack, Decay, Sustain, and Release

The SID chip has three voices which can act independently or in combination. Sound Editor 64 uses voice 1 as its primary sound source. There is one register, at location 54296, which controls the volume of all three voices. It must be set with a value from 1 to 15 for sound to occur. This program uses the maximum volume setting, 15.

The volume of a sound passes through four phases, called

the *envelope*. These phases consist of Attack, Decay, Sustain, and Release (ADSR). Each voice has a gate, or switch, that is used to initiate the attack phase when set to 1, or initiate the release phase when reset to 0. The *attack* rate specifies the time allowed to reach maximum volume, as determined by the setting at location 54296. An attack value of 0 is very short, and 15 is very long. Explosions and percussion instruments have a low Attack value.

The *decay* rate is the time allowed for the volume to fall from the maximum to the *sustain* volume. As in attack, a value of 0 is short, and 15 is long. The sustain parameter determines the volume at which the sound is maintained until the voice gate is reset to 0, when the *release* phase begins. A sustain value of 0 is minimum volume, 8 is half of maximum volume, and 15 is the maximum volume attained during the attack phase. The release rate determines how fast the volume falls to 0 from the sustain volume after the voice gate is reset to 0. A value of 0 is fast, while a value of 15 is slow.

Duration

Duration, another parameter used with the SID chip registers, is the amount of time between turning the voice gate on and resetting it to 0. The values for duration used by Sound Editor 64 are intervals of 60 per second. Thus a value of 60 is 1 second and a value of 6 is .1 second (or 100 milliseconds). As you can see, the envelope is closely related to time. Each phase takes an amount of time as specified for each parameter. The duration time must be long enough to allow attack and decay to complete before the voice gate is reset. Otherwise sound distortions may occur. For this reason, very short sounds usually require ADSR values of attack 0, decay 0, sustain 15, release 0, as well as a small duration.

The SID chip can produce eight octaves (0–7) of tones. Sound Editor 64 dynamically generates and stores the tone settings of octave 7 using the highest note, B, as a base. The octave is divided into 12 tones, where each tone's frequency is 2↑(1/12) lower than the next higher tone (A#=B/2↑(1/12)). The frequency of a tone is also half that of the same note in the next higher octave (octave 6=octave 7/2). Therefore, the program can generate the scale for any octave *N* (where *N* is 0–7) by using the formula OCTAVE N=OCTAVE 7/2↑(7–N). This saves memory by eliminating the need for an array of 96

frequency settings to define eight octaves of 12 tones each; instead, it uses one octave and an array of 12 tones to calculate the settings.

Waveform

A sound's *waveform* determines its harmonic content, or "color." The SID chip provides *triangle* (17), *sawtooth* (33), *pulse* (65), and *noise* (129) waveforms. These can produce sounds of many different qualities, and the best way to learn about them is to just experiment with Sound Editor 64. When the Pulse waveform is selected, you also have to provide a value for the *Pulse Width* (0–4095). A value of 2048 produces a square wave, which creates a clear, hollow sound. Other values produce varying degrees of "body." Waveform values 19 and 21 combine the frequencies of voices 1 and 3 to produce more complex sounds. Value 19 synchronizes the two frequencies to produce complex harmonic structures, while value 21 modulates voice 1 with voice 3 to produce ringing sounds such as bells or gongs. Sound Editor 64 uses note C for voice 3's frequency, one octave lower than that specified for voice 1.

Sound Experimentation

Sound Editor 64 operates very simply. After you've typed it in and SAVEd it, RUN it. You'll see a title, and then the first prompt will appear. As each parameter prompt shows on the screen, enter the *a* value which falls in its range, then press RETURN. The parameters you'll need to fill, and the range of possible values listed in parentheses are:

Attack value (0–15)
Decay value (0–15)
Sustain value (0–15)
Release value (0–15)
Octave value (0–7)
Duration loop value (0 on up)
Waveform (17,19,21,33,65, or 129)
Pulse width value (0–4095) (Only used when waveform value 65 is entered.)

When you're entering values, make sure that you only use numerals. If you use any other characters, such as letters or other symbols on the keyboard, you'll have to start over again. Each parameter must have a value from the stated range entered when the prompt appears.

The duration loop value changes the speed at which the tones are played. If you want to hear each note more clearly, increase this value. The waveforms and their values are:

Triangle (17)
Synchronized voices 1 and 3 (19)
Modulated voices 1 and 3 (21)
Sawtooth (33)
Pulse (65)
Noise (129)

After you've entered the various parameter values, you can choose one of the four options using the appropriate function key. The functions, and the appropriate keys, are:

BASIC (f1). This option will list the BASIC program lines you would add to a program of your own to produce the sound for note C of the octave you selected.

Change (f3). You can modify the existing parameters with this option. After pressing f3, use the RETURN key to move to the line you want to change. Enter the modification, making sure you erase any unwanted numerals that may extend beyond the value you now desire, and hit RETURN.

Scale (f5). This option plays the 12 tones of the octave you've specified. Use this to hear what the sound is like.

Quit (f7). This terminates the program.

Sound Starts

Sound Editor 64 is best used to experiment with the various parameters of the SID chip's registers. To begin with, try out some of the following values. Varying the duration and octave values will change the sound you hear, making it more or less like the instrument listed.

Table 1. Instrument Values

Sound	Attack	Decay	Sustain	Release	Waveform	Pulse Width*
Trumpet	6	0	8	0	33	N/A
Violin	10	8	10	9	33	N/A
Xylophone	0	9	0	0	17	N/A
Piano	0	9	0	9	65	1000
Flute	9	10	0	0	17	N/A
Harpsi-chord	0	9	0	0	33	N/A
Organ	0	0	15	0	17	N/A
Clarinet	8	4	8	0	17	N/A
Chimes	0	11	0	9	19	N/A

*NA=Not Applicable

Sound Editor 64

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

```
100 REM SOUND EDITOR                                    :rem 197
110 PRINT"{CLR}","SOUND EDITOR{3 DOWN}"                 :rem 233
115 DIMF(11):F(11)=64814:FORF=10TO0STEP-1:F(F)=INT
    (1/2+F(F+1)/2↑(1/12)):NEXT                          :rem 37
120 SD=54272:V=SD+24:FORI=SDTOV:POKEI,0:NEXT:POKEV
    ,15                                                 :rem 111
130 DIMN$(11):N$(0)="C ":N$(1)="C#":N$(2)="D ":N$(
    3)="D#":N$(4)="E ":N$(5)="F "                       :rem 149
140 N$(6)="F#":N$(7)="G ":N$(8)="G#":N$(9)="A ":N$
    (10)="A#":N$(11)="B ":GOTO200                       :rem 246
150 PRINT"{HOME}{2 DOWN}ENTER OPTION [F1] BASIC
    {2 SPACES}[F3] CHANGE"                              :rem 91
152 PRINTTAB(13)"[F5] SCALE{2 SPACES}[F7] QUIT"
                                                        :rem 8
153 GETOP$:IFOP$=""THEN153                              :rem 17
155 IFOP$="{F7}"THENPRINT"{CLR}";:POKEV,0:END
                                                        :rem 240
160 IFOP$="{F3}"THEN200                                 :rem 177
165 IFOP$="{F1}"THEN500                                 :rem 184
168 IFOP$="{F5}"THEN400                                 :rem 188
170 GOTO150                                             :rem 103
```

```
200 INPUT"{DOWN}ENTER{2 SPACES}ATTACK VALUE (0-15)
    ";A                                          :rem 186
205 IFA<0ORA>15THENPRINT"{3 UP}":GOTO200   :rem 17
210 INPUT"ENTER{3 SPACES}DECAY VALUE (0-15)";D
                                                 :rem 91
215 IFD<0ORD>15THENPRINT"{2 UP}":GOTO210  :rem 136
220 POKESD+5,A*16+D                              :rem 39
230 INPUT"ENTER SUSTAIN VALUE (0-15)";S     :rem 45
235 IFS<0ORS>15THENPRINT"{2 UP}":GOTO230  :rem 170
240 INPUT"ENTER RELEASE VALUE (0-15)";R      :rem 7
245 IFR<0ORR>15THENPRINT"{2 UP}":GOTO240  :rem 170
250 POKESD+6,S*16+R                              :rem 75
260 INPUT"ENTER{2 SPACES}OCTAVE VALUE{2 SPACES}(0-
    7)";OC                                       :rem 219
261 IFOC<0OROC>7THENPRINT"{2 UP}":GOTO260 :rem 251
280 INPUT"ENTER DURATION LOOP{2 SPACES}VALUE";DU
                                                 :rem 221
285 IFDU<1THENPRINT"{2 UP}":GOTO280         :rem 99
290 INPUT"ENTER WAVEFORM 17 19 21 33 65 129";W
                                                 :rem 136
294 RS=0:H3=0:L3=0:IFW=19ORW=21THENRS=1    :rem 154
295 IFRS=1THENSC=INT(F(0)/2↑(8-OC)):H3=INT(SC/256)
    :L3=SC-H3*256                                :rem 217
296 POKESD+15,H3:POKESD+14,L3                    :rem 218
300 IFW=65THEN310                                :rem 228
303 PRINT"{38 SPACES}":GOTO150                   :rem 112
310 INPUT"ENTER PULSE WIDTH VALUE (0-4095)";PW
                                                 :rem 206
315 IFPW<0ORPW>4095THENPRINT"{2 UP}":GOTO310
                                                 :rem 188
320 PH=INT(PW/256):PL=PW-PH*256              :rem 95
330 POKESD+2,PL:POKESD+3,PH:GOTO150          :rem 172
400 FORF=0TO11:SC=INT(F(F)/2↑(7-OC)):X=INT(SC/256)
    :POKESD+1,X:POKESD,SC-256*X                  :rem 186
410 TD=TI+DU:POKE53280,F:PRINT"{HOME}{23 DOWN}
    {3 RIGHT}";N$(F):POKESD+4,W                  :rem 202
420 IFTI<TDTHEN420                               :rem 91
430 POKESD+4,W-1:NEXT:POKESD+4,0:POKE53280,14:PRIN
    T"{UP}{5 SPACES}":GOTO150                    :rem 114
500 PRINT"{HOME}{14 DOWN}10 SD=54272:V=SD+24"
                                                 :rem 149
502 PRINT"15 FORI=SDTOV:POKEI,0:NEXT:POKEV,15"
                                                 :rem 163
504 SC=INT(F(0)/2↑(7-OC))                        :rem 117
505 H=INT(SC/256):L=SC-256*H                 :rem 82
510 PRINT"20 POKESD,";MID$(STR$(L),2);":POKESD+1,"
    ;MID$(STR$(H),2);"{4 SPACES}"                :rem 129
520 PRINT"30 POKESD+5,";MID$(STR$(16*A+D),2);
                                                 :rem 239
```

```
525 PRINT":POKESD+6,";MID$(STR$(16*S+R),2);"
    {6 SPACES}"                              :rem 48
530 IFW=65THENGOSUB630                       :rem 110
535 IFRS=1THENGOSUB650                       :rem 137
540 PRINT"40 TD=TI+";MID$(STR$(DU),2);":POKESD+4,"
    ;MID$(STR$(W),2);"{9 SPACES}"            :rem 144
545 PRINT"50 IFTI<TDTHEN50{17 SPACES}"       :rem 104
550 PRINT"60 POKESD+4,0{20 SPACES}"           :rem 82
560 PRINT"{26 SPACES}"                       :rem 108
600 GOTO150                                  :rem 101
630 PRINT"35 POKESD+2,";MID$(STR$(PL),2);     :rem 78
640 PRINT":POKESD+3,";MID$(STR$(PH),2);"
    {10 SPACES}":RETURN                      :rem 124
650 PRINT"35 POKESD+15,";MID$(STR$(H3),2);   :rem 99
660 PRINT":POKESD+14,";MID$(STR$(L3),2);"
    {7 SPACES}":RETURN                       :rem 151
```

SYS Sound

Michael Steed

POKEing the SID chip's registers produces sounds on the 64. But that can become complicated and discouraging, especially to the beginning programmer. "SYS Sound" is a machine language program that will help you create sound in your own programs, without using those cumbersome POKEs.

The Commodore 64 has an amazing sound chip, as you've probably already discovered. However, to really make the SID chip sing, you've got to go through the laborious process of POKEing in values to various registers. If you've tried to use sound in your own programs, you know how difficult this can be. That is, until now. "SYS Sound" will make creating sound much easier, and you won't have to use a single POKE.

Careful Entering

Type in Program 1, SYS Sound, taking special care as you enter the DATA statements. It's a good idea to save a copy before you run the program, for one error can cause it to crash. SYS Sound includes a total checksum, which will tell you if you've entered all the DATA correctly, as well as individual line checksums if you use "Automatic Proofreader," found in Appendix J. You can even SAVE this program on a machine language monitor such as "Supermon." Other monitors, such as "Micromon," will not work, however, because both the program and monitor will try to use location 49152. The program displays the directions to save it with a monitor.

After you've got a working copy of SYS Sound, type RUN. You'll have to wait for a moment while the DATA is loaded into the computer's memory. Now you're ready to use SYS Sound in your own programming.

SYSing Sounds

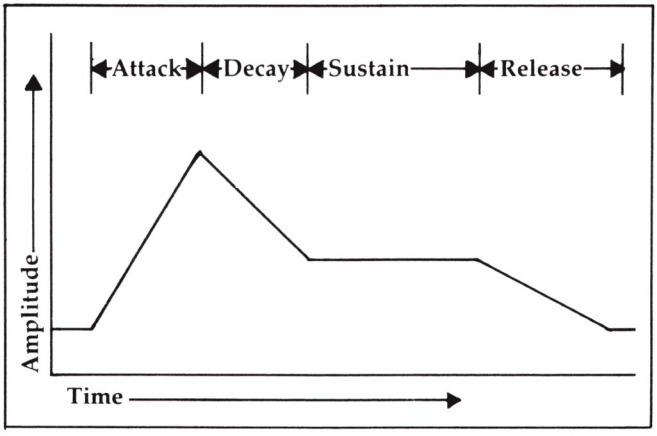
To use SYS Sound, all you need to do is type SYS 49152, followed by any of several possible parameters. The parameters must be separated by commas. The number 49152 could (and probably should) be defined as a variable, such as S or SOUND. You can then call SYS Sound directly from your own program, as long as it's still in memory. Once you've turned the computer off, however, SYS Sound disappears. You'd have to load it again to use it.

The parameters used in SYS Sound, and their meanings are:

- V*x*, where *x* is the voice number used for the note (1, 2, or 3). More than one voice may be used at the same time (see Program 2).
- A*x*, where *x* is the attack rate of the note. This is the time it takes the sound to reach its highest volume. The larger the number, the longer it takes. (See the figure for a graphic description of attack, decay, sustain, and release.)
- D*x*, where *x* is the decay rate of the note. This is the time it takes the sound to soften to the sustain volume.
- S*x*, where *x* is the sustain level of the note. The sound remains at this volume until the release starts.
- R*x*, where *x* is the release rate of the note. The release rate is the time it takes the sound to drop from the sustain volume to silence.
- W*x*[*y*], where *x* is the letter representing the waveform used for the sound. This can be *N* (noise), *S* (sawtooth), *T* (triangle), or *P* (pulse). If the chosen waveform is pulse, then a pulse rate (0 to 4095) must be entered after the waveform letter, such as WP2048 for a square wave.
- F*x*, where *x* is the frequency of the note (0 to 65535). Higher frequencies produce higher notes.
- L*x*, where *x* is the volume (loudness) of the note (0 to 15). Note that this is the overall volume, so all the voices will be affected by it.
- C clears the sound chip. This is equivalent to the following in BASIC:

```
10 S=54272:FOR I=0 TO 24:POKE S+I,0:NEXT
```

ADSR Envelope



Once certain parameters have been set, they need not be entered the next time the routine is used. For example, if all your sound effects were going to be done with voice 1, at volume 15, with the sawtooth waveform, attack 0, decay 9, and sustain and release 0, you could set all these at the beginning of your program by:

```
10 S=49152:SYS S,C,V1,L15,WS,D9
```

All parameters default to 0 initially, so A, S, and R needn't be entered. Then all that would need to be done to play a note would be:

```
20 SYS S,F5000
```

Any valid numeric expression may be used after the parameter letter. Also, if a parameter is entered more than once, only the last case will be considered. For example, SYS S,WS,WT, A0,A6 is effectively the same as SYS S,WT,A6.

To clear up any possible confusion, Program 2, "Circus Sounds," provides a simple example of a sound created with SYS Sound and its various parameters.

Program 1. SYS Sound

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

```
100 DATA 32,121,0,208,3,76,241,192,201      :rem 52
110 DATA 44,240,3,76,67,193,32,115,0       :rem 224
120 DATA 162,8,221,76,193,240,6,202,16      :rem 68
```

```
130 DATA 248,76,67,193,138,10,170,189      :rem 46
140 DATA 85,193,133,251,189,86,193,133     :rem 96
150 DATA 252,32,50,192,76,0,192,108,251    :rem 121
160 DATA 0,32,55,193,201,1,144,4,201       :rem 209
170 DATA 4,144,3,76,72,193,202,142,114     :rem 70
180 DATA 193,96,32,55,193,10,10,10,10      :rem 15
190 DATA 141,123,193,173,120,193,41,15     :rem 71
200 DATA 13,123,193,141,120,193,96,32      :rem 17
210 DATA 55,193,141,123,193,173,120,193    :rem 124
220 DATA 41,240,13,123,193,141,120,193     :rem 58
230 DATA 96,32,55,193,10,10,10,10,141       :rem 4
240 DATA 123,193,173,121,193,41,15,13      :rem 18
250 DATA 123,193,141,121,193,96,32,55      :rem 29
260 DATA 193,141,123,193,173,121,193       :rem 236
270 DATA 41,240,13,123,193,141,121,193     :rem 64
280 DATA 96,32,115,0,162,3,221,103,193     :rem 65
290 DATA 240,6,202,16,248,76,67,193,224    :rem 137
300 DATA 1,240,6,32,115,0,76,196,192       :rem 223
310 DATA 32,44,193,192,16,144,3,76,72      :rem 29
320 DATA 193,142,117,193,140,118,193       :rem 237
330 DATA 162,1,189,107,193,141,119,193     :rem 83
340 DATA 96,32,44,193,142,115,193,140      :rem 30
350 DATA 116,193,96,32,55,193,141,122      :rem 33
360 DATA 193,96,169,0,162,24,157,0,212     :rem 80
370 DATA 202,16,250,169,0,141,115,193      :rem 20
380 DATA 141,116,193,76,115,0,173,115      :rem 26
390 DATA 193,208,5,173,116,193,240,37      :rem 38
400 DATA 174,114,193,189,111,193,133       :rem 238
410 DATA 251,169,212,133,252,160,6,185     :rem 75
420 DATA 115,193,145,251,136,16,248,160    :rem 128
430 DATA 4,173,119,193,9,1,145,251,173     :rem 79
440 DATA 122,193,141,24,212,96,165,122     :rem 73
450 DATA 208,2,198,123,198,122,76,121      :rem 35
460 DATA 0,32,166,173,32,247,183,166       :rem 240
470 DATA 20,164,21,96,32,44,193,152,208    :rem 128
480 DATA 11,224,16,176,7,138,96,162,11     :rem 82
490 DATA 76,58,164,162,14,208,249,86        :rem 5
500 DATA 65,68,83,82,87,70,76,67,53,192    :rem 157
510 DATA 72,192,94,192,112,192,134,192     :rem 86
520 DATA 152,192,203,192,213,192,220       :rem 228
530 DATA 192,78,80,83,84,128,64,32,16      :rem 45
540 DATA 0,7,14,0,0,0,0,0,0,0,0,0,0        :rem 111
550 PRINT"{CLR}{DOWN}PLEASE WAIT..."       :rem 136
560 FORI=49152TO49531:READJ:POKEI,J:K=K+J:PRINT"
    {HOME}";SPC(54);J:NEXT                 :rem 247
570 IFK<>44621THENPRINT"ERROR IN DATA STATEMENTS":
    STOP                                   :rem 180
580 PRINT"{CLR}{3 DOWN}SYS SOUND"SPC(31)"[9 T]":
    Q$=CHR$(34)                            :rem 251
```

```
590 PRINT"TO SAVE IN MONITOR:":PRINT"{DOWN}.S "Q$"
    SYS SOUND"Q$",Ø1,CØØØ,C17C                :rem 85
600 PRINTSPC(15)"↑↑":PRINTSPC(15)"{DOWN}Ø1 FOR TAP
    E,":PRINTSPC(15)"Ø8 FOR DISK               :rem 23
```

Program 2. Circus Sounds

```
100 REM *** SYS SOUND EXAMPLE ***            :rem 5
110 REM PARENTHESES IN 180, 190, 200        :rem 108
120 REM ARE JUST FOR CLARITY                :rem 148
130 REM                                     :rem 120
140 REM * EXPERIMENT!!! *                    :rem 49
150 S=49152:SYS S,C,L15:T=TIME              :rem 254
160 READ D:IF D=Ø THEN SYS S,C:END          :rem 114
170 READ F1,F2,F3                           :rem 116
180 SYS S,V1,F(F1),WS,AØ,D9,SØ,RØ            :rem 81
190 SYS S,V2,F(F2),WS,A2,D4,S2,R2            :rem 85
200 SYS S,V3,F(F3),WT,A1,D2,S1Ø,R1Ø         :rem 171
210 T=T+1Ø*D:REM DURATION                   :rem 246
220 IF T>TIME GOTO 22Ø                      :rem 177
230 GOTO 16Ø                                :rem 101
300 DATA 2,6430,3215,1607                   :rem 201
310 DATA 2,7217,3215,1432                   :rem 202
320 DATA 2,8101,4050,2408                   :rem 198
330 DATA 2,8583,5728,3215                   :rem 223
340 DATA 1,9094,4547,2408                   :rem 222
350 DATA 1,9634,4817,2408                   :rem 223
360 DATA 2,12860,8101,2864                    :rem 8
370 DATA 2,3215,Ø,Ø                         :rem 149
380 DATA 2,10814,8101,2864                    :rem 7
390 DATA 2,9634,6430,2145                   :rem 219
400 DATA 2,2145,1607,1351                   :rem 200
410 DATA 2,8583,5407,1607                   :rem 219
420 DATA 2,1607,1351,1607                   :rem 204
430 DATA 2,2145,1072,536                    :rem 155
440 DATA 2,8583,2703,1607                   :rem 218
450 DATA 2,8101,2703,803                    :rem 154
460 DATA 2,7217,5407,3215                   :rem 214
470 DATA 2,8101,5728,2408                   :rem 217
480 DATA 2,2025,1607,803                    :rem 158
490 DATA 2,11457,8101,1607                    :rem 7
500 DATA 2,2025,1607,803                    :rem 151
510 DATA 2,3608,2864,1432                   :rem 213
520 DATA 2,7217,5728,2864                   :rem 226
530 DATA 2,8101,5728,4817                   :rem 220
540 DATA 2,6430,3215,1607                   :rem 207
550 DATA 2,7217,5407,2145                   :rem 215
560 DATA 2,3215,2703,Ø                       :rem 50
570 DATA 2,10814,3215,1607                    :rem 3
```

```
580 DATA 2,2145,1607,1351          :rem 209
590 DATA 2,1072,536,0               :rem 6
600 DATA 1,6430,1607,0             :rem 48
610 DATA 1,6430,1607,0             :rem 49
620 DATA 2,6430,1607,0             :rem 51
630 DATA 2,6430,1607,803           :rem 159
640 DATA 2,6430,3215,1607          :rem 208
650 DATA 2,7217,3215,1432          :rem 209
660 DATA 2,8101,4050,2408          :rem 205
670 DATA 2,8583,5728,3215          :rem 230
680 DATA 1,9094,4547,2408          :rem 229
690 DATA 1,9634,4817,2408          :rem 230
700 DATA 2,12860,8101,2864          :rem 6
710 DATA 2,6430,0,0                :rem 149
720 DATA 2,10814,8101,2864          :rem 5
730 DATA 2,9634,6430,2145          :rem 217
740 DATA 2,1072,1607,1351          :rem 205
750 DATA 2,8583,5407,1607          :rem 226
760 DATA 2,1607,1351,1607          :rem 211
770 DATA 2,2145,1072,536           :rem 162
780 DATA 2,8583,2703,1607          :rem 225
790 DATA 2,8101,5728,2408          :rem 222
800 DATA 2,7647,6430,2703          :rem 217
810 DATA 2,7217,3608,2864          :rem 223
820 DATA 2,11457,7217,2408          :rem 11
830 DATA 2,10814,7217,2703          :rem 6
840 DATA 2,4817,7217,2864          :rem 229
850 DATA 2,12860,6430,803          :rem 214
860 DATA 2,10814,6430,4291          :rem 9
870 DATA 2,9634,6430,803           :rem 173
880 DATA 2,8583,4291,3215          :rem 227
890 DATA 2,9634,8101,5728          :rem 231
900 DATA 2,1607,3215,0              :rem 50
910 DATA 2,9634,5728,1804          :rem 227
920 DATA 2,2025,4050,0              :rem 45
930 DATA 6,8583,5407,1072          :rem 226
940 DATA 0                         :rem 231
```

The Note Name Game

Jeff Behrens

"The Note Name Game" is an educational program which makes learning the notes of the musical scale easy and fun.

Musical notation is like anything else—it's easy once you learn it, but learning it is not always easy.

Sight-reading of notes is vital for anyone who wants to play a musical instrument, because instant note recognition is a must. That's the idea behind "The Note Name Game." My daughters, who are taking piano lessons, love playing it. Although it does not teach everything about musical notation, it does help students to practice quick recognition of notes in the treble and bass clefs.

Treble or Bass

The program begins by asking whether you want to practice notes on the treble clef (enter a T), the bass clef (B), or a mixture of both (M). The program then selects a note at random and places it on the appropriate clef.

Next, the program asks for the letter name of the note displayed. If your response is correct, you are told so, and the next note is displayed. If your response is wrong, the correct answer is highlighted on the screen and the next note is shown. The program constantly updates your score and displays it on the screen.

Notes are shown in sets of ten. If you wish to quit before finishing a set, type Q instead of the answer. Whether you finish or not, the score is printed and you are asked if you want to play again.

Customizing the Program

Depending on personal preference, there are some changes you might want to make. I find the TV picture is sharpest

when the screen and border are black and the cursor blue during the game. You may, of course, specify any screen/border combination by substituting the appropriate number for the 0 in the POKE statement on line 185 for the background and the value in the POKE V+32 statement in line 5 for the border color. You can even change the background color for the title screen by altering the POKE V+33 statement in lines 5 and 325. (See Appendix E for possible combinations).

The variables R and W, respectively, are the number of right and wrong answers. The string variable N$(2,24) is a string array containing the note names and the POKE values for the sound registers.

The Note Name Game

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

```
5 PRINT"{CLR}":V=53248:SD=54272:POKE646,14:POKEV+3
  2,Ø:POKEV+33,7:DIM N$(2,24):SC=Ø          :rem 78
6 NO%=25:POKEV+21,Ø                         :rem 69
8 FORI=SDTOSD+28:POKEI,Ø:NEXTI             :rem 219
1Ø FOR I=ØTO24:READN$(Ø,I):NEXTI            :rem 135
15 FOR I=ØTO24:READN$(1,I):NEXTI            :rem 141
2Ø FOR I=ØTO24:READN$(2,I):NEXTI            :rem 138
25 :::REM READ SPRITE DATA                    :rem 6
3Ø FOR I=OTO62: READQ: POKE832+I,Q: NEXTI  :rem 138
35 FOR I=OTO62: READQ: POKE896+I,Q: NEXTI  :rem 153
4Ø FOR I=OTO62: READQ: POKE96Ø+I,Q: NEXTI  :rem 141
45 :::REM TELL COMPUTER WHERE SPRITE IS     :rem 137
5Ø POKE2Ø42,13:POKE2Ø43,14:POKE2Ø44,15      :rem 116
55 :::REM POSITION SPRITE ON SCREEN         :rem 165
6Ø POKEV+4,16Ø:POKEV+5,7Ø                   :rem 191
65 POKEV+6,158:POKEV+7,11Ø                  :rem 25Ø
7Ø POKEV+8,158:POKEV+9,171                    :rem 1
75 :::REM COLOR SPRITES                     :rem 167
78 POKEV+41,1:POKEV+42,1:POKEV+43,1          :rem 6Ø
8Ø :::REM EXPAND SPRITES                    :rem 228
85 POKEV+29,28:POKEV+23,28                    :rem 3
9Ø :::REM SET SOUND PARAMETERS              :rem 1ØØ
95 POKESD+24,15:POKESD+5,4:POKESD+6,17Ø:POKESD+2,Ø
   :POKESD+3,9:POKESD+12,2                  :rem 164
96 POKESD+13,243:POKESD+19,Ø:POKESD+2Ø,245:rem 2Ø6
1ØØ PRINT"{CLR}{2 DOWN}";TAB(11);"{RVS}THE NOTE NA
    ME GAME{OFF}"                           :rem 81
1Ø5 PRINT"{5 DOWN}{6 RIGHT}I WILL PLAY A NOTE FOR
    {SPACE}YOU AND"                         :rem 79
11Ø PRINT"{DOWN}{3 RIGHT}THEN SHOW YOU A NOTE ON A
     STAFF."                                :rem 47
```

```
115 PRINT"{DOWN}{3 RIGHT}I WANT YOU TO TELL ME THE
     NAME OF":PRINT"{DOWN}{3 RIGHT}THE NOTE."
                                              :rem 5
120 PRINT"{3 DOWN}{4 RIGHT}INPUT{2 SPACES}{RED}
    {RVS}B{OFF} FOR BASS, {RVS}T{OFF} FOR TREBLE,"
                                            :rem 162
125 PRINTTAB(13);"{DOWN}OR{2 SPACES}{RVS}M{OFF} FO
    R MIXED."                                :rem 95
128 POKE198,0                               :rem 200
130 GETE$:IFE$=""THEN130                     :rem 83
135 IFE$<>"T"ANDE$<>"B"ANDE$<>"M"THEN130    :rem 233
185 POKE V+33,0                              :rem 16
190 FOR L=1TO10                              :rem 63
200 POKEV+21,0:PRINT"{CLR}{7}{2 DOWN}{RIGHT}WHAT
    {2 SPACES}NOTE":PRINT"{DOWN}{2 RIGHT}IS THIS?
    {HOME}"                                  :rem 94
205 M=25:S=0:IFE$="B"THENM=13               :rem 148
210 IFE$="T"THENM=13:S=12                   :rem 170
215 RN%=INT(RND(0)*M+S)                      :rem 48
217 IFRN%=NO%THEN215                        :rem 180
218 NO%=RN%                                  :rem 95
220 GOSUB4500                               :rem 221
225 POKEV+21,28:PRINT"{HOME}{DOWN}":GOSUB750
                                            :rem 199
230 FORZ=1TO2:PRINT"{16 RIGHT}{24 SPACES}";:NEXTZ
                                              :rem 2
235 GOSUB750:PRINT"{HOME}"                  :rem 212
245 IFRN%=24THENPRINT"{HOME}{29 SPACES}***
    {HOME}"                                 :rem 248
250 IFRN%=12THENPRINT"{HOME}{12 DOWN}{29 SPACES}
    ***{HOME}"                             :rem 189
255 IFRN%=0THENPRINT"{HOME}{23 DOWN}{30 SPACES}
    ***{HOME}                              :rem 40
260 POKE2014+54272-RN%*40,1:POKE2014-RN%*40,81
                                            :rem 223
265 PRINT"{HOME}{20 DOWN}('Q' TO QUIT){HOME}"
                                            :rem 190
268 PRINT"{HOME}{18 DOWN}{RVS}SCORE{OFF} :";SC;"
    {LEFT}%{2 SPACES}{HOME}"                 :rem 53
270 PRINT"{7 DOWN}{2 RIGHT}> ";             :rem 148
273 POKE198,0                               :rem 201
275 GETGU$:IFGU$=""THEN275                   :rem 21
280 IF(ASC(GU$)<65 OR ASC(GU$)>71)AND ASC(GU$)<>81
    THENPRINT"{8 UP}":GOTO270               :rem 106
285 PRINTGU$                                :rem 236
290 IFGU$="Q"THEN 310                       :rem 127
295 IFGU$=N$(0,RN%)THENGOSUB400              :rem 83
300 IFGU$<>N$(0,RN%)THENGOSUB500            :rem 132
```

```
305 IFR+W<>ØTHENSC=INT((R/(R+W))*1ØØ+.5):NEXT
                                          :rem 156
310 POKEV+21,Ø:PRINT"{CLR}"               :rem 161
315 PRINT"{7 DOWN}{9 RIGHT}YOUR SCORE WAS";SC;"
    {LEFT}%"                              :rem 174
318 POKE198,Ø                             :rem 201
320 PRINT"{5 DOWN}{4 RIGHT}WOULD YOU LIKE TO PLAY
    {SPACE}AGAIN";:INPUTY$                :rem 151
325 IFLEFT$(Y$,1)="Y"THENR=Ø:W=Ø:SC=Ø:POKE V+33,7:
    GOTO1ØØ                               :rem 93
330 SYS2Ø48:REM END OF PROGRAM             :rem 5
400 :::REM CORRECT                        :rem 56
410 POKESD+11,129                         :rem 176
420 FORI=536TO9Ø94STEP256:PRINT"{3 DOWN}{RIGHT}
    {WHT}{RVS} CORRECT {OFF}{WHT}":HI=INT(I/256):L
    O=I-HI*256                            :rem 71
430 PRINT"{UP}{2 SPACES}CORRECT {4 UP}{7}":POKES
    D+8,HI:POKESD+7,LO:NEXTI              :rem 244
440 FORT=1TO1Ø:NEXT:POKESD+11,128:FORT=1TO9ØØ:NEXT
    :R=R+1:RETURN                         :rem 59
500 :::REM INCORRECT                      :rem 208
505 POKESD+18,33:POKESD+16,Ø:POKESD+15,6  :rem 103
510 PRINT"{DOWN}SORRY, THAT'S":PRINT"{DOWN}INCORRE
    CT."                                  :rem 225
515 PRINT"{DOWN}IT WAS: ";N$(Ø,RN%)        :rem 94
520 FORT=1TO1ØØØ:NEXT:POKESD+18,32:FORT=1TO9ØØ:NEX
    T:W=W+1:RETURN                        :rem 117
750 FORX=1TO5                              :rem 33
755 PRINT TAB(16);:FORI=1TO24:PRINT CHR$(99);:NEXT
                                          :rem 24
760 PRINT"{16 RIGHT}{24 SPACES}";:NEXTX:RETURN
                                          :rem 1ØØ
780 PRINT"THE NOTE WAS:";N$(Ø,RN%)        :rem 2Ø3
785 W=W+1:RETURN                           :rem 5
1ØØØ :::REM TELL COMPUTER WHERE SPRITE IS :rem 225
1ØØ5 POKE2Ø42,13:POKE2Ø43,14:POKE2Ø44,15  :rem 213
1Ø1Ø :::REM POSITION SPRITE ON SCREEN     :rem 253
1Ø15 POKEV+4,16Ø:POKEV+5,7Ø                :rem 32
1Ø2Ø POKEV+6,158:POKEV+7,11Ø               :rem 82
1Ø25 POKEV+8,158:POKEV+9,171               :rem 98
1Ø3Ø :::REM COLOR SPRITES                 :rem 255
1Ø35 POKEV+41,1:POKEV+42,1:POKEV+43,1     :rem 15Ø
1Ø4Ø :::REM EXPAND SPRITES IN BOTH DIRECTIONS
                                          :rem 249
1Ø45 POKEV+29,28:POKEV+23,28               :rem 96
1Ø5Ø :::REM TURN ON SPRITES               :rem 1Ø4
1Ø55 POKEV+21,28                          :rem 116
1999 END                                  :rem 179
2ØØØ PRINT:GOSUB 497Ø                     :rem 221
```

Sound 3

```
2005 FORZ=1TO2:PRINT"{16 RIGHT}{24 SPACES}";:NEXTZ
                                                :rem 52
2010 GOSUB4970:PRINT"{HOME}"                    :rem 53
4500 POKE SD+1,VAL(N$(2,RN%)):POKESD,VAL(N$(1,RN%)
     ):POKESD+4, 65                            :rem 108
4510 FORT=1TO 600 :NEXT:POKESD+4, 64            :rem 26
4520 RETURN                                     :rem 171
4970 FORX=1TO5                                   :rem 89
4980 PRINT TAB(16);: FORZ=1TO24: PRINT CHR$(99);:N
     EXTZ                                       :rem 183
4990 PRINT"{16 RIGHT}{24 SPACES}";:NEXTX:RETURN
                                                :rem 157
5000 DATAE,F,G,A,B,C,D,E,F,G,A,B,C,D,E           :rem 68
5010 DATAF,G,A,B,C,D,E,F,G,A                     :rem 22
5020 DATA71,152,71,12                            :rem 48
5030 DATA233,97,104,143                         :rem 155
5040 DATA48,143,24,210                          :rem 100
5050 DATA195,209,31,96                          :rem 117
5060 DATA30,49,165,135                          :rem 110
5070 DATA162,62,193,60,99                        :rem 12
5080 DATA5,5,6,7,7                              :rem 165
5090 DATA8,9,10,11,12                            :rem 47
5100 DATA14,15,16,18,21                         :rem 142
5110 DATA22,25,28,31,33                         :rem 144
5120 DATA37,42,44,50,56                         :rem 154
5140 DATA0,224,0,0,208,0,0,216,0,0,204,0,0   :rem 5
5145 DATA206,0,0,199,0,0,199,0,0,199,0,0,206,0
                                                :rem 238
5150 DATA0,204,0,0,216,0,0,240,0,0,224,0,1   :rem 3
5155 DATA192,0,3,192,0,6,192,0,12,192,0,24,192
                                                :rem 248
5160 DATA0,48,192,0,96,192,0,224,192,0     :rem 121
5170 DATA48,111,128,97,248,192,195,96,96,198,96,48
     ,195,104,48,193                           :rem 246
5175 DATA232,48,96,248,96                       :rem 27
5180 DATA112,96,224,56,96,192,28,99,192,7   :rem 52
5185 DATA111,0,1,248,0,0,96,0,0,0,0,0,0,0,0,0,0
                                                :rem 238
5190 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0      :rem 33
5200 DATA 0,248,0,3,6,0,6,3,128,6,0,198,3,192,198,
     3,192,192,0,0,198,0,1,134,0,1             :rem 8
5210 DATA128,0,3,0,0,3,0,0,6,0,0,12,0,0,24,0,0,112
     ,0,1,192,0,3,0,0,0,0,0,0,0,0              :rem 142
5220 DATA0,0,0,0,0,0                           :rem 223
```

4

Colors, Characters, and Motion

Introduction to Custom Characters for the 64

Tom R. Halfhill

What are "custom characters"? Why might you want them? Are they hard to program? How do they work? This introduction to the concept of custom characters answers all these questions and more. Another article in this book, "How to Make Custom Characters on the 64," shows you exactly how to program custom characters.

Perhaps you've admired the screen graphics of a favorite arcade-style game, or the Old English letters of a Gothic text adventure. These kinds of shapes and special characters are not built into the computer itself. Maybe you've wondered how these effects are achieved and if they are difficult to program.

The secret is a technique called *custom characters,* also known as *redefined characters* or *programmable characters.* The terms are almost self-explanatory—with programming, you can design your own shapes and special characters to display on the TV screen. They can be almost any shapes you want: spaceships, aliens, animals, human figures, Old English letters, anything. In effect, you are *customizing* or *redefining* the characters already built into the computer.

For instance, if you redefine the letter A to look like an alien creature, every time you PRINT A on the screen you'll get the alien instead of the letter. Animation is as easy as erasing the character—by PRINTing over it with a blank space—and then PRINTing it in the next position. When this process is repeated rapidly, the alien seems to move across the screen.

Custom characters are especially useful to game programmers, but also are fun to experiment with for anyone interested in programming.

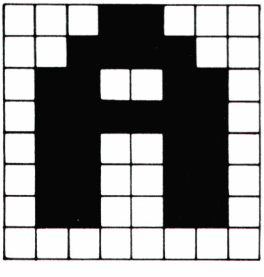
Character Sets

First, let's clarify exactly what a *character set* is. Briefly, it is the complete set or collection of characters that a particular computer can display on its video screen. *Characters* include letters of the alphabet (both upper- and lowercase), numbers, punctuation marks, symbols, and—on the Commodore 64—the 64 special graphics characters that are pictured on the front of the keys. In all, your 64 has a standard character set of 256 characters. This is the total set of characters which the computer is capable of displaying.

The character set is built into the computer, permanently stored in Read Only Memory (ROM). ROMs are memory chips that retain important information even when power is turned off between sessions. The character set is stored in ROM as a list of numbers. The numbers describe to the computer how each character is formed from a pattern of tiny dots.

You may be able to see these dots if you look very closely at your computer screen. (The dots might be too small to discern on some ordinary TV sets, but they are much more visible on a monitor.) All the characters in the character set are made up of these dots. The dots for each character are part of an 8-by-8 grid, for a total of 64 dots per grid . This method of forming characters is familiar to anyone who has seen the large time/temperature clocks on banks, or the scoreboards in sports stadiums. A computer displays characters the same way, except instead of light bulbs, the dots are very small pinpoints of glowing phosphor on the TV picture tube. (Figure 1 shows the dot pattern for the letter A on a Commodore 64.)

Figure 1. Dot Pattern for Character A



The character set is always kept in ROM, ready for the computer to use. Let's say you display a character on the screen—for instance, the uppercase letter A. The computer refers to the character set in ROM to see how it should display the A on the screen, much as you would refer to a dictionary to see how to spell a word. Once it looks up the dot pattern for an A, the computer displays the character. The whole process takes only a few microseconds, and happens every time a character is displayed, either by typing on the keyboard or using a PRINT statement in BASIC.

When the computer's ROM chips are preprogrammed for you at the factory, these dot patterns for each character are permanently burned into the chips so the computer will always display the same character set. Short of replacing the ROM chips themselves, there is nothing you can do to change this preprogramming. Normally, this would limit you to the built-in character set. Indeed, on some computers there is no alternative.

Fooling the Computer

However, on the 64—and on many other home computers—there is a way to modify the character set to suit your own needs. The technique requires fooling the computer.

Here's how it's done. The first obstacle to overcome is the preprogrammed ROM chips. It's not possible to erase or change information in ROM. But remember, there are two types of memory chips in computers: ROM and RAM.

RAM (Random Access Memory) is temporary memory that *can* be erased and changed. Programs loaded from disk or tape, or which you write yourself, are stored in RAM while they run. They can be changed at any time from the keyboard, or even erased altogether by typing NEW or switching off the computer. RAM is the computer's workspace.

So, the first step toward custom characters is to copy the list of numbers representing the character set from ROM into RAM.

This is a relatively simple programming task. You find out exactly where in ROM the character set is stored by looking at a *memory map*, a list of memory addresses inside the computer. (Memory maps are often found in reference or owner's manuals or magazine articles.) Once you know the beginning memory address of the ROM character set, you can write a

short routine which reads the list of numbers in ROM and then copies it into RAM. In BASIC, this is done with PEEKs and POKEs within a FOR-NEXT loop. One or two program lines are all it takes.

Now there's a copied image of the ROM character set in RAM. Again using POKEs, you can freely change the list of numbers to customize the characters any way you want (we'll cover this in detail in a moment).

OK so far, but there's one catch. The computer doesn't know you've relocated the character set. It still expects to find the character set where it always has, in ROM. It will continue to refer to ROM and will ignore your customized set in RAM.

That's why you have to "fool" the computer. The 64 contains a memory location, called a *pointer*, which points to the character set in ROM. Luckily, the pointer itself is in RAM. With a single POKE statement, you can change the number in this location to point to your custom character set in RAM, thereby fooling the computer into referring there for its information instead of ROM. The computer goes through its usual process of looking up the dot pattern for each character and displaying it on the screen, except it looks up *your* modified pattern instead of the pattern preprogrammed at the factory.

Clever, eh?

Character Patterns

Basically, if you've made it this far, you've got the picture. But there are still a few details to clean up.

For example, exactly how are characters customized?

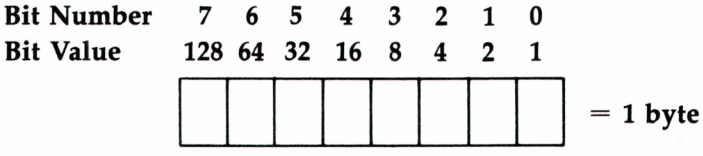
Recall that the character set is defined by a list of numbers which describes the dot patterns for each character, and that each character is formed by dots within an 8-by-8 grid. By changing these numbers, you change the shape of the dot pattern, and therefore the shape of the character.

It helps at this point to know something about the binary number system. Each byte of memory in your 64 is made up of eight bits. These bits can be set to 1 or 0, hence the term binary. A bit that is set to 1 is often referred to as being *on*, while a bit set to 0 is said to be *off*. The pattern of on and off bits in a byte creates a particular value, ranging from 0 to 255.

Within a byte, each bit has an individual value assigned to it. The bit on the far right represents a value of 1 when it is

on. The bit next to it, to the left, represents a value of 2 when it is on. This pattern continues, each bit to the left representing a value *double* that of the previous bit. Look at Figure 2 for a moment to see this pattern.

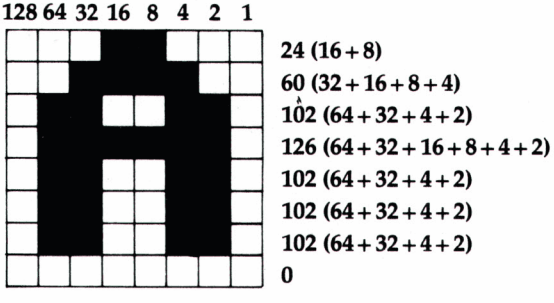
Figure 2. Binary Number Values



For instance, if the left-most and right-most bits were both on, while all the rest were off, the byte's total value would be 129. You arrive at that value by simply adding together the *on* bits' values. 128+1=129. To show the pattern of a character, whether it's a standard character or a custom character, binary values have to be added together. It's not difficult. Figure 3 shows the standard character A as an example.

The eight numbers running vertically along the right side of Figure 3 are the numbers which define the dot pattern for an A on a Commodore 64. These are the same eight numbers which the computer refers to when it looks up A in the character set. They are also the numbers you must change to customize the character. These numbers are decimal versions of the binary dot patterns.

Figure 3. Dot Pattern for A



Along the top of Figure 3, running horizontally *from right to left,* are the bit values.

Now, this is important: to understand how the numbers in the *vertical column* were determined, simply add up the numbers in the *horizontal row* which correspond to colored dots in the 8-by-8 grid. For example, the top row of the grid has two colored dots which form the peak of the A. (These are the same dots which will be lit up when the letter is displayed on the TV screen.) These two dots fall beneath the 8 and 16 of the top row of numbers. Because 8+16=24, the number in the right-hand column for that row is 24.

Likewise, the next number in the right-hand column is 60, because the colored dots in the second row of the grid fall beneath the 4, 8, 16, and 32, which add up to 60. And so on down to the very last row, which has no colored dots. This is represented by a 0 in the right-hand column. When the A is displayed on the screen, no dots will be lit up on this row of the grid. (All patterns for letters and numbers allow a blank line for the last row, and for the extreme right and left-hand columns, in order to keep the characters from running into each other on the screen.)

Customizing Characters

Once you understand how character patterns work, it's easy to customize them at will.

First, take some graph paper and mark off an 8-by-8 grid, or draw your own grid on a blank sheet. Along the top, write down the horizontal row of numbers as seen in Figure 3: 1, 2, 4, 8, 16, 32, 64, and 128. *Be sure to list them from right to left.*

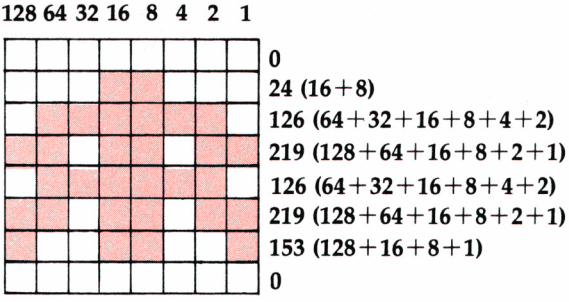
Second, design your custom character by coloring in dots on the grid. Figure 4 shows a sample design for a Space Invaders-type creature.

Third, add up the colored dots in each row, starting from the top. Write down each sum in a vertical column along the right, as seen in the figures.

You have now designed your own custom character. You can design as many of these as you'll need—up to the limit of 256 characters in the character set.

The only remaining step is to take the new series of eight numbers for each custom character and substitute them for the numbers in the standard character set. Remember, that's why you relocated the character set from ROM to RAM. Now that

Figure 4. Dot Pattern for a Customized Character (Space Invaders-Type Alien)



the list of numbers spelling out the patterns for the standard character set is in RAM, it can be changed to use your own numbers with POKE statements.

Specific Details

Up to now, this article has been fairly general in its explanations. The basic technique for customizing characters is the same for almost any computer on which the character set can be relocated and redefined. But the specific details vary for each computer: the character set's memory address in ROM, how to safely copy it to RAM, the memory address of the character set pointer, the order of characters within the character set, and so on.

For these details, as well as example programs and utilities, turn to the next article "How to Make Custom Characters on the 64."

How to Make Custom Characters on the 64

Gary Davis

Before reading this, be sure to see "Introduction to Custom Characters for the 64" in this book, especially if you're unfamiliar with the concepts of redefined characters. The following article includes "Chred 64," a character-editing utility that makes the task of customizing characters easy and fun.

The Commodore 64 allows you to change any character in the character set to suit your own needs. In order to understand how this is done, it is first necessary to understand how the 64 (and most other computers) store the character set.

If you look closely at the letters the computer puts on the screen, you'll notice that each character is made up of little dots in an 8 x 8 grid (see the figure).

Since there are 64 possible dots, or *pixels*, that can be either on or off, we need 64 "switches" for each character. This is done by using eight memory locations for each character. Since one memory location, or *byte*, is divided into eight *bits*, using eight bytes gives us the 64 switches we need for each character.

The bytes for each character are stored consecutively, with the first byte for each character representing the top row of dots in the character, the second byte the second row of dots, and so on. For a pixel to be on, the bit at its location must be *set*; for a pixel to be off, the bit must be *clear*. This is not as complicated as it sounds. The figure shows how the bit patterns of sets and clears are converted into the numbers that represent the character. When you make a series of bytes for

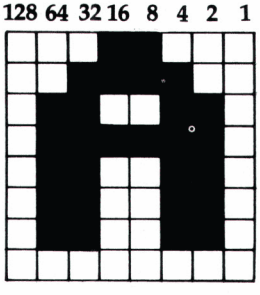
every character and store them in a computer, you have what is known as a *character generator.*

Relocating the Character Set

The character generator in most computers, including the Commodore 64, is stored in Read Only Memory (ROM). This way the computer is ready to display characters on the screen as soon as it is turned on.

Unfortunately, when the character generator is in ROM, you can't change the characters to suit your needs. When you can't change the existing character set, the simplest way to customize a new character set is to move it to Random Access Memory (RAM), and then tell the computer to use your character set rather than the one it has in ROM.

Pixel Pattern for Letter A



Telling the Commodore 64 where the new character set is located is relatively simple to do. Within the video controller chip (sometimes known as the VIC-II chip) is a special memory location that allows you to set a new character pointer (the location of the first byte of your character set).

Now let's try an experiment. Type POKE 53272,19 and press RETURN. Your screen will be filled with strange characters, but don't worry. You have told the 64 to use a RAM character generator, but you haven't supplied one yet. To return your screen to normal type POKE 53272,21 and press RETURN. You won't be able to read what you are typing until you press RETURN, but the computer understands. If this doesn't work, you can always restore the screen by pressing the RESTORE and RUN/STOP keys at the same time.

When you are designing a new character set, it is nice to have the normal one loaded into RAM to start with. Then you can make changes to it. Program 1 copies the 64's character set from ROM to RAM.

Before you type in this program, *you must enter:*

```
POKE 8192,0: POKE 44,32: NEW
```

This saves a place in RAM memory for your new character set and protects it from being overwritten by a BASIC program.

Now, type in the program and RUN it. After about 45 seconds the computer will come back and say READY. Now type POKE 53272,19 and press RETURN. Nothing appears to happen, but the characters you are now seeing on your screen are coming from your RAM character generator, not from ROM as usual.

To test this, type POKE 2056,255. The top of all the letter A's on the screen should now be a solid line. Try POKEing different numbers into memory locations between 2048 and 6143 and watch the results on the characters.

Using a Character Editor

By sketching an 8 x 8 grid as seen in the figure, it's possible to map out the entire character set on graph paper and convert your new characters to numbers to POKE into memory.

This method, however can be both time-consuming and frustrating. A far better way is to create your new characters on the screen and let the computer do all the calculations. With this thought in mind, I wrote a character editor called "Chred 64." With this utility (Program 2), you can redefine any of the text or graphics symbols and save them on tape or disk. This can then be loaded and used with any program.

In order to reserve memory for the alternate character set, it is necessary to set the start of BASIC pointer to 8192. This will leave you with 32K of RAM free for your BASIC program. To do this, *you must* type in the following:

```
POKE 8192,0
POKE 44,32
NEW
```

Now the memory from 2048 to 8191 is free to hold your new character set. You may type in or load Chred 64. After typing Chred 64 for the first time, be sure to save it on tape or

disk before you run it. If you have made a typing error, it is possible that the computer will "crash" and you'll have to type it all over again if you haven't saved a copy.

When you run Chred 64, the program first copies the resident character set from ROM to RAM and resets the character base to point to the RAM character set. The program then expands the current character being edited to eight times its normal size.

To edit the current character being displayed, you may use the cursor control keys, the asterisk, and the space bar.

To turn on a pixel, position the cursor and press the asterisk. To turn off a pixel, press the space bar. To clear the entire character, press CLR.

To edit a different character, press f1. You will be asked to supply a row and column. This refers to the block of characters displayed on the lower right corner of the screen. Just type a row number followed by the column number or letter. The character you selected will now be displayed, ready for you to edit.

More Editing Features

An interesting feature of the 64 is that, unlike the Commodore PET, the reverse-field (inverse video) characters are stored as part of the character set. This allows 256 redefined characters. To edit a character not being displayed, press f3. This will select and display the next block of 64 characters. Rest assured that you may mix characters from any of the blocks; only 64 characters are shown at a time for the purpose of editing.

Sometimes you may wish to edit more than one character at a time to make a larger shape. This can be easily accomplished by pressing f5. Instead of a single character, you will be able to edit a block of four characters. To go back to single character mode, just press f5 again.

After you have redefined several characters, the text on the screen may become unreadable as your new characters replace the existing ones. To restore the character set to normal, *without* destroying your new character set, press f7. To return to your new character set, press f7 again.

When you are done working with a character set, you can restore the font to the normal character set by pressing R. You will be asked "Are you sure?" Now is your last chance to save your character set. If you are really done, press Y; otherwise, press N.

Saving and Loading

After you have gone to the effort of creating a new character set, you will probably want to save it on disk or tape for use in other programs. To save your character set, press S. Follow the directions given on the screen. After the character set is saved, you will be returned to the editor. (When typing Chred 64, omit line 225 for use with tape.)

Sometimes you may wish to alter a character set that you have already created and saved. To load another character set, press L and follow the directions given on the screen. Be careful—the new character set is loaded on top of the current character set, so be sure to save it if you want to use it later.

OK, you've developed your new character set. To use it with another program, you will have to type POKE 8192, 0:POKE 44, 32: NEW, just as you do when you load Chred 64. To load in the character set, place the cassette containing your new character set in the recorder, or the disk in the drive. For tape, type LOAD " filename", 1,1 where "filename" is the name you gave when you saved the character set. For disk, type LOAD "filename", 8,1. To use the new character set, POKE 53272, 19. To return to the normal character set, POKE 53272,21.

I hope you have as much fun using this program as I had writing it.

Program 1. Character Set Transfer to RAM

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

```
10 POKE 56334,0:REM TURN OFF INTERRUPTS     :rem 83
20 POKE 1,51:REM TURN OFF VIDEO CHIP TO EXPOSE CHA
   RACTER GENERATOR                         :rem 220
30 FOR ADDRESS = 2048 TO 6143               :rem 204
40 POKE ADDRESS, PEEK (ADDRESS + 51200):{3 SPACES}
   REM COPY CHARACTERS TO RAM                :rem 32
50 NEXT ADDRESS                             :rem 170
60 POKE 1,55:REM TURN ON VIDEO CHIP         :rem 251
70 POKE 56334,129:REM TURN ON INTERRUPTS    :rem 135
80 END                                       :rem 63
```

Program 2. Chred 64

```
100 REM "CHRED 64"                          :rem 137
120 POKE53280,11:POKE53281,0:PRINT"{5}"  :rem 189
130 V=53248:SC=1024:CB=2048:CC=SC+40*21+9 :rem 222
140 SZ=7:FP=0:FO=0:TP=0:TY=0:SL=0:R$="0":C$="0"
                                            :rem 199
```

```
150 CO$="* {RIGHT}{LEFT}{DOWN}{UP}{HOME}{CLR}{F1}
    {F3}{F5}{F7}SLRQ"                          :rem 85
160 DN$="{HOME}{17 DOWN}"                       :rem 2
170 BL$="{18 SPACES}"                          :rem 203
180 NU$="0123456789ABCDEF"                     :rem 131
190 DEFFNA(F)=SC+62+40*CY+CX                    :rem 86
200 DEFFNB(F)=CB+((FO+TP)*8)+TY                :rem 251
210 PRINT"{CLR}{5 DOWN}{9 SPACES}{YEL}CHARACTER SE
    T LOADING"                                 :rem 97
220 GOSUB1500                                  :rem 218
225 POKE 49276,8:REM ENTER THIS LINE FOR DISK ONLY
     -- OMIT FOR CASSETTE                      :rem 94
230 SYS49152:POKEV+24,19                       :rem 127
240 POKE 53281,1:PRINT"{CLR}":POKE 53281,0:GOSUB10
    60                                         :rem 12
250 CY=0:CX=0                                  :rem 226
260 POKEFNA(0),PEEK(FNA(0))OR128                :rem 81
270 POKE198,0                                  :rem 198
280 GETCH$:IFCH$=""THEN280                     :rem 235
290 FORCH=1TOLEN(CO$):IFMID$(CO$,CH,1)=CH$THEN310
                                               :rem 149
300 NEXTCH:GOTO270                             :rem 105
310 POKEFNA(0),PEEK(FNA(0))ANDNOT128           :rem 112
320 ONCHGOTO330,370,410,430,450,470,490,500,590,77
    0,800,1040,910,900,840,1020                :rem 119
330 POKEFNA(0),42                              :rem 173
340 GOSUB550                                   :rem 177
350 POKEFNB(0),PEEK(FNB(0))OR2↑(ABS(TX-7)):rem 208
360 GOTO260                                    :rem 106
370 POKEFNA(0),32                              :rem 176
380 GOSUB550                                   :rem 181
390 POKEFNB(0),PEEK(FNB(0))ANDNOT2↑(ABS(TX-7))
                                               :rem 247
400 GOTO260                                    :rem 101
410 CX=CX+1:IFCX>SZTHENCX=0                    :rem 234
420 GOTO260                                    :rem 103
430 CX=CX-1:IFCX<0THENCX=SZ                    :rem 236
440 GOTO260                                    :rem 105
450 CY=CY+1:IFCY>SZTHENCY=0                    :rem 242
460 GOTO260                                    :rem 107
470 CY=CY-1:IFCY<0THENCY=SZ                    :rem 244
480 GOTO260                                    :rem 109
490 GOTO250                                    :rem 109
500 FORCY=0TOSZ:FORCX=0TOSZ:POKEFNA(0),32 :rem 158
510 GOSUB550                                   :rem 176
520 POKEFNB(0),0                               :rem 121
530 NEXTCX,CY                                   :rem 58
540 GOTO250                                    :rem 105
550 TP=FP:TX=CX:TY=CY:IFTX>7ANDTY<8THENTP=TP+1:TX=
    TX-8                                       :rem 177
```

```
560 IFTX<8ANDTY>7THENTP=TP+2:TY=TY-8       :rem 134
570 IFTY>7ANDTX>7THENTP=TP+3:TY=TY-8:TX=TX-8
                                           :rem 189
580 RETURN                                 :rem 125
590 PRINTDN$"{RVS}{YEL}ROW, COLUMN?{OFF}{5}";
                                           :rem 107
600 POKE198,0                              :rem 195
610 GETR$:IFR$=""THEN610                   :rem 115
620 IFVAL(R$)<>0ANDVAL(R$)<40RR$="0"THENR=VAL(R$):
    GOTO640                                :rem 165
630 GOTO600                                :rem 104
640 PRINTR$",";                             :rem 72
650 POKE198,0                              :rem 200
660 GETC$:IFC$=""THEN660                    :rem 95
670 IFC$=CHR$(20)THENPRINT"{2 LEFT}{2 SPACES}";:GO
    TO590                                   :rem 10
680 IFASC(C$)>64THENC=ASC(C$)-55:IFC>15THEN600
                                           :rem 174
690 IFVAL(C$)<>0ORC$="0"THENC=VAL(C$)      :rem 118
700 IFC>15THEN650                          :rem 215
710 PRINTC$                                :rem 140
720 FP=R*16+C                              :rem 189
730 IFSZ=15ANDFP>60THENFP=60:C$="C":C=12   :rem 112
740 GOSUB1290                              :rem 231
750 PRINTDN$;BL$                           :rem 204
760 GOTO250                                :rem 109
770 IFFO<191THENFO=FO+64:GOTO790           :rem 215
780 FO=0                                   :rem 161
790 FP=0:R$="0":C$="0":GOSUB1240:GOTO250   :rem 225
800 IFSZ=15THENSZ=7:GOTO830                :rem 213
810 IFFP>60THENFP=60:C$="C":C=12             :rem 76
820 SZ=15                                  :rem 234
830 POKE 53281,1:PRINT"{CLR}":POKE 53281,0:CX=0:CY
    =0:GOSUB1060:GOTO250                   :rem 160
840 PRINTDN$;"{RVS}{YEL}ARE YOU SURE? ";   :rem 156
850 POKE198,0                              :rem 202
860 GETCH$:IFCH$="N"THENPRINTDN$;"{OFF}{5}";BL$:
    GOTO250                                :rem 134
870 IFCH$<>"Y"THEN860                      :rem 193
880 PRINT"YES{OFF}{5}"                    :rem 140
890 SYS49152:GOSUB1310:PRINTDN$;BL$:GOTO250
                                           :rem 152
900 SL=1                                   :rem 166
910 PRINTDN$;:INPUT"{RVS}{YEL}FILE NAME";NA$:rem 5
920 POKE253,LEN(NA$)                       :rem 115
930 IFLEN(NA$)=0THEN970                     :rem 74
940 FORL=1TOLEN(NA$)                       :rem 196
950 POKE49359+L,ASC(MID$(NA$,L,1))         :rem 125
960 NEXT                                   :rem 222
```

```
970 SYS49269                                        :rem 173
980 PRINTDN$;BL$;DN$;"{6 UP}"                       :rem 167
990 IFSL=0THENSYS49292:GOTO1010                     :rem 170
1000 SYS49310                                       :rem 193
1010 POKE 53281,1:PRINT"{CLR}{5}":POKE 53281,0:G
     OSUB1060:SL=0:GOTO250                           :rem 32
1020 POKEV+24,21                                    :rem 104
1030 PRINT"{CLR}{3 DOWN}":END                       :rem 108
1040 IFPEEK(V+24)=19 THEN POKEV+24,21:GOTO260
                                                    :rem 55
1050 POKEV+24,19:GOTO260                            :rem 125
1060 PRINT"{HOME}{RVS}{YEL}CHARACTER EDITOR{OFF}"
                                                    :rem 65
1070 PRINT"{DOWN}{RVS}{YEL}F1{OFF}{5} EDIT NEW C
     HAR."                                          :rem 87
1080 PRINT"{RVS}{YEL}F3{OFF}{5} NEXT CHAR. BLOCK
     "                                             :rem 227
1090 PRINT"{RVS}{YEL}F5{OFF}{5} BLOCK SIZE"
                                                   :rem 150
1100 PRINT"{RVS}{YEL}F7{OFF}{5} FLIP CHARACTER S
     ET"                                           :rem 142
1110 PRINT"{RVS}{YEL} R{OFF}{5} RESTORE FONT":rem27
1120 PRINT"{RVS}{YEL} S{OFF}{5} SAVE CHAR. SET"
                                                    :rem 41
1130 PRINT"{RVS}{YEL} L{OFF}{5} LOAD CHAR. SET"
                                                    :rem 20
1140 PRINT"{RVS}{YEL} Q{OFF}{5} QUIT"               :rem 5
1150 PRINT"{HOME}{19 DOWN}{RVS}"TAB(21);" ";NU$;"
     {SPACE}{OFF}"                                  :rem 29
1160 FORL=1TO4:PRINTTAB(21)"{RVS}"MID$(NU$,L,1);SP
     C(16);" ":NEXT                                :rem 164
1170 PRINTTAB(21)"{RVS}{18 SPACES}{OFF}{2 UP}"
                                                   :rem 235
1180 PRINT"{HOME}"TAB(21);                         :rem 116
1190 PRINT"{RVS} ";MID$(NU$,1,SZ+1);:PRINT" {OFF}"
                                                   :rem 105
1200 FORL=1TOSZ+1                                   :rem 16
1210 PRINTTAB(21)"{RVS}"MID$(NU$,L,1);SPC(SZ+1) ;"
      {OFF}"                                       :rem 169
1220 NEXTL                                          :rem 80
1230 PRINTTAB(21)"{RVS}";:FORL=0TOSZ+2:PRINT" ";:N
     EXT:PRINT"{OFF}"                               :rem 82
1240 CH=FO                                          :rem 36
1250 FORY=1TO4                                      :rem 77
1260 FORX=1TO16                                    :rem 128
1270 POKESC+781+X+Y*40,CH:CH=CH+1                  :rem 143
1280 NEXTX,Y                                       :rem 231
1290 PRINT"{HOME}{19 DOWN}{5 SPACES}{RVS}EDITING "
     ;R$","C$"{OFF}":POKECC,FP+FO                  :rem 216
```

```
1300 IFSZ=15THENPOKECC+1,FP+FO+1:POKECC+40,FP+FO+2
     :POKECC+41,FP+FO+3                          :rem 125
1310 X=0:Y=0:CX=0:CY=0                            :rem 15
1320 GOSUB1390                                    :rem 19
1330 IFSZ<>15THEN1380                            :rem 222
1340 X=8:Y=0:FP=FP+1:GOSUB1390                    :rem 27
1350 X=0:Y=8:FP=FP+1:GOSUB1390                    :rem 28
1360 X=8:Y=8:FP=FP+1:GOSUB1390                    :rem 37
1370 FP=FP-3                                     :rem 148
1380 RETURN                                      :rem 172
1390 TP=FP:TX=CX:TY=CY:IFTX>7ANDTY<8THENTP=TP+1:TX
     =TX-8                                       :rem 228
1400 IFTX<8ANDTY>7THENTP=TP+2:TY=TY-8            :rem 176
1410 IFTY>7ANDTX>7THENTP=TP+3:TY=TY-8:TX=TX-8
                                                 :rem 231
1420 TE=8*(FO+TP)+CB:REM CHAR. POINTER           :rem 239
1430 POKE251,TE-INT(TE/256)*256                  :rem 233
1440 POKE252,INT(TE/256)                          :rem 94
1450 TE=FNA(0)+X+40*Y:REM SCREEN LOC.            :rem 117
1460 POKE253,TE-INT(TE/256)*256                  :rem 238
1470 POKE254,INT(TE/256)                          :rem 99
1480 SYS49209                                    :rem 212
1490 RETURN                                      :rem 174
1500 FORL=49152TO49319                           :rem 232
1510 READD:POKEL,D:NEXT                          :rem 197
1520 RETURN                                      :rem 168
1530 REM FONT COPIER ROUTINE                     :rem 204
1540 DATA120,169,51,133,1,169,1,141,13,220,169,0,1
     33,251,133,253,169,208,133                  :rem 189
1550 DATA252,169,8,133,254,160,0,177,251,145,253,2
     30,251,230,253,208,246,230                  :rem 205
1560 DATA252,230,254,165,252,201,225,208,236,169,1
     29,141,13,220,169,55,133,1                  :rem 205
1570 DATA88,96                                   :rem 242
1580 REM CHAR EXPAND AND DISPLAY                 :rem 121
1590 DATA160,0,162,0,169,128,133,250,177,251,37,25
     0,208,4,169,32,208,2,169,42                   :rem 3
1600 DATA145,253,24,102,250,240,8,230,253,208,2,23
     0,254,208,229,230,251,208,2                 :rem 230
1610 DATA230,252,165,253,24,105,33,133,253,165,254
     ,105,0,133,254,232,224,8,208                 :rem 33
1620 DATA201,96                                   :rem 17
1630 REM SAVE AND LOAD ROUTINES                   :rem 73
1640 DATA169,128,133,157,169,1,162,1,160,1,32,186,
     255,165,253,162,208,160,192                  :rem 11
1650 DATA32,189,255,96,169,0,133,251,169,8,133,252
     ,169,251,162,16,160,25                       :rem 33
1660 DATA32,216,255,96                           :rem 116
1670 DATA169,0,162,0,160,8,32,213,255,96         :rem 226
```

SuperBASIC Sprite Editor

Martin C. Kees

Adding sprites to your programs, especially to games, can make them graphically impressive. But designing the sprites and creating the necessary DATA statements is time-consuming if you have to do it on graph paper. "SuperBASIC Sprite Editor" makes designing sprites easy and fun. Using SuperBASIC, a powerful program that adds 41 new commands to your 64's BASIC, this sprite editor is versatile, yet simple to use. SuperBASIC is necessary to run this program.

Sprites, those graphics blocks that you can sculpt into any shape you want, are a powerful feature on the Commodore 64. They're very useful when you're designing games, for they move quickly and smoothly. It's even quite easy to create animation using sprites. However, drawing sprite patterns on graph paper and then calculating the DATA statements to place in your program can be tiresome, especially when you have several sprite patterns to create.

That's where a sprite editor comes in handy. A good editor should make it easy and fun to design sprites. It should allow you to change colors at will, create multicolored or single colored sprites, show the sprites' final shape, and create the DATA values you'll need later. If it's even more powerful, it should let you move the sprites on the screen, animate them, and store and load them to and from tape or disk.

"SuperBASIC Sprite Editor" gives you all these functions, and more. It's easy to use, fast in its execution, and includes a variety of commands.

Sprite Creation

Maybe you've already designed your own sprites. In that case,

you can type in SuperBASIC Sprite Editor and use it immediately. If you're just starting to learn about sprites, however, it's a good idea to first read another article in this book, "Sprites Made Easy." Included in that article is a section called "Sprite Creation," which will explain the rudiments of sprite design. After reading through that, you should have a good idea of what a sprite is, and how its DATA numbers are calculated. You'll be relieved to know that you won't have to calculate those values yourself if you use SuperBASIC Sprite Editor. The program can do that for you. All you'll have to do is type those values into your own program.

SuperBASIC

SuperBASIC Sprite Editor is written in SuperBASIC, a powerful addition to the BASIC in your 64 which adds 41 new commands and enhances 8 existing commands. You type it in and save it as you would any other BASIC program. However to use this editor, you first need to have a copy of SuperBASIC loaded into your computer. SuperBASIC makes writing programs like Sprite Editor easier, and makes such programs much more powerful. If you haven't already, read the article on SuperBASIC and type in the program before you begin entering SuperBASIC Sprite Editor. *Remember that you can't use this program unless you've got SuperBASIC LOADed and RUN on your 64.*

As you type in SuperBASIC Sprite Editor, you'll come across strange-looking commands, such as [DLCS or [FCOL. Don't worry, the program listing is correct; this is how SuperBASIC notes its new commands. Every time you see the [symbol in the program, just press the SHIFT and colon keys together. This will give you the bracket symbol on the screen. Type in the rest of the command (DLCS, for instance) as you would any other command on the 64. Typing in SuperBASIC Sprite Editor will take some time, but it will be worth the effort. Once you've entered it, SAVE it to be safe. You're now ready to design up to 127 sprite patterns.

Functions and Command Keys

Although the program is for the most part self-explanatory, especially if you've used or seen other sprite editors at work, a few details may be helpful to you. Once the program is run, it will take a few moments to set up. A menu display then

appears, showing you all the functions and command keys that SuperBASIC Sprite Editor uses. Briefly, they are:

f1 Selects the background color that shows on the screen. Pressing the f1 key repeatedly will cycle through all 16 colors available on the 64.
f2 Selects the border color of the screen. Works just as the background color selection does.
f3 Selects the color for multicolor 0 when you're designing a multicolor sprite. As with the previous commands, pressing this key will cycle through all the available colors.
f4 Selects text color.
f5 Selects sprite color, either in multicolor, or normal mode.
f7 Selects multicolor 1 when the editor is in multicolor mode.
1, 2, 3 These keys set the pixel the cursor is presently on when you're using the single-color mode. It's like setting that bit on. When you're using the multicolor mode, the keys work a bit differently. The 1 key sets pixels on for multicolor 0, the 2 key sets pixels on in the sprite color, and the 3 key turns on pixels for multicolor 1.
SPACE The space bar turns off any pixel(s) at the present cursor position.
Cursor Keys The normal cursor keys move the blinking cursor around the sprite pattern so that you can set and clear individual pixels. Remember that you have to use SHIFT/CRSR DOWN to move up, and SHIFT/CRSR RIGHT to move left.
CLR/HOME You can clear an entire sprite display pattern by pressing the SHIFT key along with this key. It's a handy command if you decide to start over as you're designing a sprite pattern.
R This key shifts the sprite pattern one pixel horizontally. You can only move in one direction (towards the left), but it will wrap around if you press the key several times.
V This key will shift the sprite pattern one pixel vertically. It moves upward, but will wrap around.
L You can flip the sprite pattern laterally using this key. If the sprite points towards the right, for instance, using this key will make it point to the left.
F Similar to the previous key, this flips the pattern vertically. What once pointed up will now point down.
C This key toggles the multicolor mode. Press it once, and you're in single-color mode; press it again, and you can design multicolored sprites.

S You have a choice of storing an edited sprite in any of 127 blocks. Once you've created a sprite to your satisfaction, you can store it by pressing this key. The program will ask for the block to assign the sprite to, and you should enter a number from 1 to 127. Note that this does not permanently store the sprite pattern. If you turn your computer off, then on again, the pattern will disappear. You need to use the O key command to store a pattern to disk or tape. However, if you're editing more than one pattern in a session, the S key command is quite useful.
U You can recall any sprite pattern with this key. Again, the program will ask for the block number; respond with a number from 1 to 127. That sprite will then display on the screen.
P Using the Preview command, you can look at all the sprite pattern blocks, one at a time, at your own leisure. Pressing the key displays the next sprite pattern.
O Stores the sprite pattern information permanently. You'll be asked from which block you want to save and to which block, the filename you'd like to call that pattern, and the device number (1 for tape, 8 for disk). The sprite pattern will then SAVE out to tape or disk, with your selected filename.
I Loads previously created sprite pattern files from tape or disk.
M This is perhaps the handiest command key, for as you learn to use the sprite editor, you'll find yourself constantly wanting to look at the list of command key options.
A Sprite animation is also handled by this editor. When you use this key, you'll be asked to provide several parameters. *Start block* asks for the sprite pattern block number you'd like to begin the animation with. *End block* asks the last block to animate. If you've designed three sprites to show a human figure in motion, for example, you could designate Block 1 as the starting block, and Block 3 as the ending block. As the sprite is animated, then, it will cycle through all three patterns.

Horizontal and vertical shift refer to the speed you want the sprite to move in those directions. If you want the sprite to move only horizontally, for instance, enter a value in the third parameter, and then hit RETURN for the fourth. Placing values in both shift parameters will move the sprite diagonally on the screen. Time delay sets the speed at which the sprite is animated. Higher values increase the animation speed. You can expand the sprite in the X-direction, the Y-direction, or

both. The last two parameters ask for the starting X and Y coordinates of the sprite. Refer to the *Commodore 64 Programmer's Reference Guide* for the coordinates which will show on the screen.

D Using this command key, you can see the DATA statements you would include in your own program. The program will ask for the block to be displayed, and the beginning line number of the DATA statement. The computer will calculate the values you would need to create that sprite pattern. You will, however, have to type these values into your own programs yourself. SuperBASIC Sprite Editor does not allow you to merge sprite DATA files with your own programs.

Drawing Sprites

The best way to discover how to use SuperBASIC Sprite Editor is to simply experiment. Use it to create as many sprites as you need, and then use the D command key to display the DATA statement values. This eliminates much of the work you would have to do with paper and pencil; all that remains for you to do is to enter those lines within your own game or program.

When you first use this program, you'll probably find that there are sprite patterns already in each block. Use the U command key to call a block, type 1 and RETURN. You're in Block 1 now. If it's filled, use SHIFT CLR/HOME to erase the sprite pattern. You've now got an empty pattern to work with.

If you switch from the sprite pattern display to the menu (by pressing the M command key), and then back again to the display (by pressing any key from the menu screen), you'll notice that your single-colored sprite has changed colors. To get back to the original color, just hit the C toggle key twice.

You'll find SuperBASIC Sprite Editor a valuable addition to your programming library. It's a utility you'll often use as you discover the power of sprites on the 64. Moreover, it makes creating sprites fun, instead of the chore it once was.

SuperBASIC Sprite Editor

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

```
1 REM SUPERBASIC SPRITE EDITOR                :rem 164
2 REM EDIT SPRITES INTO BLOCKS 0-127{10 SPACES}IN
  {SPACE}BANK 1                                :rem 57
3 REM FILES CREATED CAN BE LOADED TO{10 SPACES}BAN
  K 1 BY LOAD"NAME",8,1                        :rem 85
```

```
5 GOTO TA+10                                          :rem 143
10 POKE55,0:POKE56,64:CLR:CB=7*4096                    :rem 11
15 DIMBP(8),FS$(1)                                    :rem 198
20 [BANK1:[DLCS0,CB:[CB2K6:[VS1K11:PRINT"{YEL}
   {CLR}PATIENCE?{HOME}";                             :rem 161
25 [BKG40,1,12,15:[ECGR1                               :rem 95
30 GOSUB 5000:SYSCA:128                                :rem 58
40 FORJ=0TO7:POKECB+J,96:POKECB+J+224,0:POKECB+J+2
   32,0:POKECB+264+J,255:NEXT                          :rem 72
50 POKECB+225,255:POKECB+226,255:POKECB+232,96:POK
   ECB+233,224:POKECB+234,224                          :rem 86
60 FORJ=1TO21:PRINT"{24 SPACES}@":NEXT                :rem 207
70 PRINT"£££££££££££££££££££££££]"                    :rem 53
80 GOSUB2000                                          :rem 170
190 MC=1:[FCOL7:POKE650,128                            :rem 83
195 B=0:SK=12:M0=1:M1=15:TC=7:EX=14                     :rem 4
200 [VS1K11:SYSUP:MC,128:SYSCA:0:GOSUB700              :rem 91
210 R=0:C=0:SC=27648:SW=-1:GOTO220                     :rem 218
215 POKESP,PEEK(SP)AND254:IFMCTHENPOKESP+1,PEEK(SP
   +1)AND254                                           :rem 226
216 SW=-1                                             :rem 222
220 SP=SC+40*R+C:SW=-SW:POKESP,PEEK(SP)+SW:IFMCTHE
   NPOKESP+1,PEEK(SP+1)+SW                             :rem 205
230 GET A$:IFA$=""THEN220                              :rem 76
235 IFA$<>"P"THENPRINT"{HOME}{25 RIGHT}{11 SPACES}
   ";:PL=0                                              :rem 2
240 IFA$="{RIGHT}"THENC=C+1+MC:IFC>23THENC=0:GOTO2
   15                                                  :rem 202
250 IFA$="{LEFT}"THENC=C-1-MC:IFC<0THENC=23-MC:GOT
   O215                                                 :rem 10
260 IFA$="{DOWN}"THENR=R+1:IFR>20THENR=0:GOTO215
                                                       :rem 62
270 IFA$="{UP}"THENR=R-1:IFR<0THENR=20:GOTO215
                                                      :rem 191
280 IFA$="{HOME}"THENR=0:C=0:GOTO215                    :rem 5
285 IF(A$="1"ORA$="2")ANDMC=0THENA$="3"                :rem 103
290 IFA$="1"ORA$="2"ORA$="3"THENPOKESP,VAL(A$)*64+
   32                                                  :rem 158
295 IFA$="1"ORA$="2"ORA$="3"ANDMCTHENPOKESP+1,VAL(
   A$)*64+32                                            :rem 98
296 IF A$=" "ANDMCTHENPOKESP,32:POKESP+1,32:A$="
   {RIGHT}":GOTO240                                     :rem 74
297 IF A$=" "THENPOKESP,32:A$="{RIGHT}":GOTO240
                                                      :rem 239
300 IFA$="R"ANDMCTHENPOKESP,PEEK(SP)AND254:POKESP+
   1,PEEK(SP+1)AND254                                  :rem 232
302 IFA$="R"ANDMCTHENSYSRO:SYSRO:GOTO500               :rem 112
305 IFA$="R"THENPOKESP,PEEK(SP)AND254:SYSRO
                                                      :rem 237
```

```
310 IFA$="V"ANDMCTHENPOKESP,PEEK(SP)AND254:POKESP+
    1,PEEK(SP+1)AND254                          :rem 237
313 IFA$="V"ANDMCTHENSYSVR:GOTO500              :rem 163
315 IFA$="V"THENPOKESP,PEEK(SP)AND254:SYSVR
                                                :rem 249
320 IFA$="{CLR}"THENSYSUP:MC,128                  :rem 1
330 IFA$="L"ANDMCTHENPOKESP,PEEK(SP)AND254:POKESP+
    1,PEEK(SP+1)AND254                          :rem 229
333 IFA$="L"ANDMCTHENSYSLA:GOTO500              :rem 128
335 IFA$="L"THENPOKESP,PEEK(SP)AND254:SYSLA
                                                :rem 214
340 IFA$="F"ANDMCTHENPOKESP,PEEK(SP)AND254:POKESP+
    1,PEEK(SP+1)AND254                          :rem 224
342 IFA$="F"ANDMCTHENSYSFL:GOTO500              :rem 127
345 IFA$="F"THENPOKESP,PEEK(SP)AND254:SYSFL
                                                :rem 214
350 IFA$="{F2}"THENTC=TC+1:TC=TCAND15:[FCOLTC
                                                :rem 186
360 IFA$="{F1}"THENB=B+1:B=BAND15:[BKGDB          :rem 2
370 IFA$="{F3}"THENM0=M0+1:M0=M0AND15:GOTO600
                                                 :rem 10
380 IFA$="{F5}"THENSK=SK+1:SK=SKAND15:GOTO600
                                                :rem 144
390 IFA$="{F7}"THENM1=M1+1:M1=M1AND15:GOTO600
                                                 :rem 18
400 IFA$="{F4}"THENEX=EX+1:EX=EXAND15:[EXTCEX
                                                :rem 229
410 IFA$="U"THEN610                              :rem 37
420 IFA$="S"THEN640                              :rem 39
430 IFA$="P"THENPL=PL+1:PL=PLAND127:GOTO670
                                                :rem 136
440 IFA$="C"THENMC=ABS(NOT(MC=1)):SYSUP:MC,0:C=INT
    (C/2)*2:GOTO600                             :rem 182
450 IFA$="O"THENGOSUB800                        :rem 164
460 IFA$="I"THENGOSUB830                        :rem 162
470 IFA$="M"THEN2500                             :rem 83
480 IFA$="A"THEN3000                             :rem 68
490 IFA$="D"THEN1000                             :rem 70
500 SYSCA:0:GOTO215                             :rem 141
600 IFMC=0THEN[BKG4B,SK,SK,SK:GOSUB700:GOTO500
                                                :rem 173
605 [BKG4B,M0,SK,M1:GOSUB700:GOTO500            :rem 182
610 PRINT"{HOME}{22 DOWN}";:INPUT"BLOCK";BL
                                                :rem 110
620 PRINT"{HOME}{22 DOWN}{15 SPACES}";           :rem 45
630 SYSUP:MC,BL:GOTO500                         :rem 201
640 PRINT"{HOME}{22 DOWN}";:INPUT"BLOCK";BL
                                                :rem 113
650 PRINT"{HOME}{22 DOWN}{15 SPACES}";           :rem 48
```

```
660 SYSCA:BL:GOTO215                                    :rem 242
670 PRINT"{HOME}{26 RIGHT}BLOCK";PL;"{LEFT}
    {2 SPACES}";:SYSUP:MC,PL:GOTO500                    :rem 132
700 [DSPR0,0,1,1,270,60,MC,SK,M0,M1                     :rem 161
705 [DSPR1,0,0,1,284,115,MC,SK,M0,M1                    :rem 220
710 [DSPR2,0,1,0,270,175,MC,SK,M0,M1                    :rem 218
715 [DSPR3,0,0,0,284,210,MC,SK,M0,M1                    :rem 218
720 RETURN                                              :rem 121
800 PRINT"{HOME}{22 DOWN}";:INPUT"SAVE FROM BLOCK"
    ;BL                                                 :rem 210
805 PRINT"{HOME}{22 DOWN}{35 SPACES}";                   :rem 50
810 PRINT"{HOME}{22 DOWN}";:INPUT"SAVE TO BLOCK";B
    E                                                    :rem 59
820 PRINT"{HOME}{22 DOWN}{35 SPACES}";                   :rem 47
825 IFBE<0ORBL<0THENRETURN                              :rem 203
830 PRINT"{HOME}{22 DOWN}";:INPUT"FILE NAME ";FS$(
    1)                                                  :rem 249
835 PRINT"{HOME}{22 DOWN}{35 SPACES}";                   :rem 53
840 PRINT"{HOME}{22 DOWN}";:INPUT"DEVICE NUMBER";B
    P(0)                                                  :rem 6
841 PRINT"{HOME}{22 DOWN}{35 SPACES}";                  :rem 50
845 IFA$="I"THEN900                                     :rem 39
850 FORJ=43TO46:BP(J-42)=PEEK(J):NEXT                   :rem 239
860 SB=4*4096:BL=64*BL+SB:BE=64*(BE+1)+SB:BP(5)=BL
    AND255:BP(6)=INT(BL/256)                            :rem 192
865 BP(7)=BEAND255:BP(8)=INT(BE/256)                     :rem 17
870 POKE43,BP(5):POKE44,BP(6):POKE45,BP(7):POKE46,
    BP(8)                                               :rem 193
880 SAVEFS$(1),BP(0),1:POKE43,BP(1):POKE44,BP(2):P
    OKE45,BP(3):POKE46,BP(4)                            :rem 246
890 RETURN                                              :rem 129
900 TA=490:LOADFS$(1),BP(0),1                            :rem 61
910 END                                                 :rem 113
999 END                                                 :rem 130
1000 [VS1K9                                             :rem 122
1005 INPUT"{CLR}DATA FOR BLOCK";BL                       :rem 98
1100 INPUT"START LINE NUMBER";X                         :rem 168
1105 Y=4*4096+64*BL:BL=0:NL=14:PRINT"{CLR}":rem 36
1110 FORJ=0TO3:[KSPRJ:NEXT                              :rem 207
1115 PRINTX"DATA";:FORJ=1TONL:V$=STR$(PEEK(Y))
                                                        :rem 138
1120 PRINTRIGHT$(V$,LEN(V$)-1)",";:Y=Y+1:NEXT:X=X+
     10                                                  :rem 78
1130 PRINT"{LEFT} ":BL=BL+1                              :rem 34
1140 IFBL<5THEN1115                                      :rem 75
1150 IFBL=5THENNL=7:GOTO1115                            :rem 206
1155 GETA$:IFA$=""THEN1155                              :rem 187
1157 FORJ=0TO3:[ESPRJ:NEXT                              :rem 212
1160 [VS1K11:GOTO500                                    :rem 178
```

```
2000 [VS1K10:[FCOL12:[BKG40,1,12,11:PRINT"{CLR}
     {RVS}SPRITE{SHIFT-SPACE}EDITOR{SHIFT-SPACE}ME
     NU{OFF} "                                  :rem 91
2005 PRINT" F{E} BACKGROUND COLOR "            :rem 110
2010 PRINT" F{R} BORDER COLOR "                 :rem 73
2020 PRINT" F{W} SPRITE MULTI 0"               :rem 160
2030 PRINT" F{H} TEXT COLOR "                  :rem 212
2040 PRINT" F{J} SPRITE COLOR"                 :rem 104
2050 PRINT" F{Y} SPRITE MULTI 1"               :rem 168
2060 PRINT" {E} {R} {W}{2 SPACES}SETS PIXEL
     {SPACE}ON"                                 :rem 13
2070 PRINT" SPACE CLEARS PIXEL"                :rem 194
2075 PRINT" CURSOR{SHIFT-SPACE}KEYS MOVE EDIT CURS
     OR"                                        :rem 148
2076 PRINT" CLR/HOME CLEARS DISPLAY"            :rem 41
2080 PRINT" R HORIZONTAL SHIFT"                :rem 245
2090 PRINT" V VERTICAL SHIFT"                   :rem 74
2100 PRINT" L LATERAL FLIP"                    :rem 144
2110 PRINT" F VERTICAL FLIP"                   :rem 224
2115 PRINT" C SINGLE/MULTICOLOR TOGGLE"         :rem 26
2120 PRINT" S STORE EDIT SPRITE"               :rem 243
2130 PRINT" U RECALL STORED SPRITE"            :rem 199
2140 PRINT" P PREVIEW STORED SPRITES"          :rem 133
2150 PRINT" O STORE SPRITES IN FILE"           :rem 214
2160 PRINT" I LOAD SPRITE FILE"                :rem 122
2170 PRINT" M DISPLAY MENU"                    :rem 179
2180 PRINT" A ANIMATE MODE"                    :rem 129
2185 PRINT" D DATA LIST"                       :rem 187
2190 GETA$:IFA$=""THEN2190                     :rem 187
2200 RETURN                                    :rem 164
2500 [VS1K10:[FCOL12:[BKG40,11,11,15            :rem 63
2510 GETA$:IFA$=""THEN2510                     :rem 179
2520 [VS1K11:[BKG4B,M0,SK,M1:[FCOLTC:GOTO500
                                               :rem 255
3000 FORJ=0TO3:[KSPRJ:NEXT                     :rem 207
3002 ZZ=53265:WV=128:[VS1K9:PRINT"{CLR}";:INPUT"ST
     ART BLOCK";BL                              :rem 22
3005 INPUT"END BLOCK";BE                       :rem 160
3010 INPUT"HORIZ SHIFT";HS                     :rem 120
3020 INPUT"VERTICAL SHIFT";VS                   :rem 85
3030 INPUT"TIME DELAY";TD                       :rem 11
3040 INPUT"X EXPAND 0/1";XE                     :rem 27
3050 INPUT"Y EXPAND 0/1";YE                     :rem 30
3060 INPUT"X POSITION";XP                       :rem 77
3070 INPUT"Y POSITION";YP                       :rem 80
3080 PRINT"{CLR}"                               :rem 47
3085 FORJ=0TO3:[KSPRJ:NEXT                     :rem 220
3090 FORJ=BLTOBE:WAITZZ,WV:[DSPR0,J,XE,YE,XP,YP,MC
     ,SK,M0,M1                                 :rem 119
```

```
3100 XP=XP+HS:YP=YP+VS                                  :rem 180
3110 IFXP>348THENXP=0                                    :rem 29
3120 IFXP<0THENXP=340                                    :rem 20
3130 IFYP<0THENYP=255                                    :rem 28
3140 IFYP>255THENYP=0                                    :rem 31
3145 FORGX=1TOTD:NEXT                                   :rem 117
3150 NEXT                                                 :rem 8
3160 GETA$:IFA$=""THEN3090                              :rem 185
3165 FORJ=0TO3:[ESPRJ:NEXT:[DSPR0,0,1,1,270,60,MC,
     SK,M0,M1                                            :rem 25
3170 [VS1K11:[FCOLTC:GOTO500                              :rem 5
5000 Y=32768:RESTORE 5020:CA=Y:RO=CA+97:UP=CA+160:
     VR=CA+310:LA=CA+384:FL=33221                        :rem 47
5005 READV:IFV<0THENRETURN                              :rem 214
5010 POKEY,V:Y=Y+1:GOTO5005                             :rem 146
5020 DATA32,0,192,24,169,0,162,6,6,20,42,202,208,2
     50,24                                              :rem 168
5030 DATA105,64,133,21,173,136,2,133,79,169,0,133,
     78,133,253                                         :rem 185
5040 DATA133,255,160,0,162,4,24,177,78,41,128,240,
     1,56,38                                            :rem 35
5050 DATA2,200,24,177,78,41,64,240,1,56,38,2,200,2
     02,208                                             :rem 226
5060 DATA231,132,254,164,255,165,2,145,20,230,255,
     164,254,192,24                                     :rem 126
5070 DATA208,213,24,165,78,105,40,133,78,144,2,230
     ,79,230,253                                        :rem 238
5080 DATA165,253,201,21,208,192,96,173,136,2,133,7
     9,133,21,169                                       :rem 40
5090 DATA1,133,20,169,0,133,78,133,255,162,21,160,
     0,177,78                                           :rem 84
5100 DATA133,2,177,20,145,78,200,192,23,208,247,16
     5,2,145,78                                         :rem 188
5110 DATA24,165,20,105,40,133,20,144,2,230,21,165,
     21,133,79                                          :rem 105
5120 DATA165,20,133,78,198,78,202,208,213,96,32,0,
     192,165,20                                         :rem 193
5130 DATA133,2,32,0,192,24,169,0,162,6,6,20,42,202
     ,208                                               :rem 118
5140 DATA250,24,105,64,133,21,173,136,2,133,79,169
     ,0,133,78                                          :rem 135
5150 DATA133,254,133,251,133,252,169,0,133,253,164
     ,251,177,20,230                                    :rem 168
5160 DATA251,133,80,165,2,208,25,169,0,6,80,42,170
     ,189,48                                            :rem 46
5170 DATA129,164,252,145,78,230,252,230,253,165,25
     3,201,8,208,233                                    :rem 184
5180 DATA240,30,169,0,6,80,42,6,80,42,170,189,50,1
     29,164                                             :rem 247
```

```
5190 DATA252,145,78,200,145,78,200,132,252,230,253
     ,165,253,201,4                              :rem 123
5200 DATA208,226,230,254,165,254,201,3,208,177,169
     ,0,133,252,133                              :rem 122
5210 DATA254,24,165,78,105,40,133,78,144,2,230,79,
     165,251,201                                 :rem 237
5220 DATA63,208,154,96,32,224,32,96,160,224,173,13
     6,2,133,21                                  :rem 184
5230 DATA133,79,169,0,133,20,169,40,133,78,160,0,1
     77,20,153                                   :rem 135
5240 DATA0,144,200,192,24,208,246,162,20,160,0,177
     ,78,145,20                                  :rem 172
5250 DATA200,192,24,208,247,165,79,133,21,165,78,1
     33,20,24,105                                 :rem 31
5260 DATA40,133,78,144,2,230,79,202,208,225,160,0,
     185,0,144                                   :rem 126
5270 DATA145,20,200,192,24,208,246,96,0,173,136,2,
     133,21,169                                  :rem 182
5280 DATA0,133,20,133,2,169,0,133,251,169,23,133,2
     52,164,251                                  :rem 170
5290 DATA177,20,133,253,164,252,177,20,72,165,253,
     145,20,104,164                              :rem 132
5300 DATA251,145,20,230,251,198,252,165,251,201,12
     ,208,226,24,165                             :rem 166
5310 DATA20,105,40,133,20,144,2,230,21,230,2,165,2
     ,201,21                                     :rem 238
5320 DATA208,199,96,169,0,168,133,251,169,0,133,2,
     133,20,173                                  :rem 195
5330 DATA136,2,133,21,24,105,3,133,79,169,32,133,7
     8,24,165                                     :rem 86
5340 DATA251,101,20,133,20,144,2,230,21,24,165,251
     ,101,78,133                                 :rem 200
5350 DATA78,144,2,230,79,177,20,72,177,78,145,20,1
     04,145,78                                   :rem 160
5360 DATA24,165,20,105,40,133,20,144,2,230,21,56,1
     65,78,233                                   :rem 120
5370 DATA40,133,78,176,2,198,79,230,2,165,2,201,10
     ,208,216                                     :rem 89
5380 DATA230,251,165,251,201,24,208,166,96,-1
                                                 :rem 212
```

Sprites Made Easy

Paul F. Schatz

If you've always wanted to create sprites on your Commodore 64, but have been put off by all the complicated POKEs, this article is your answer. It lets you modify BASIC to add three new sprite commands to make the job much easier. An accompanying side article also explains the rudiments of sprite design.

One of the most powerful features of the Commodore 64 is its sprite animation ability. Sprites, also called MOBs (for Movable Object Blocks), are in effect graphics blocks which you can sculpt into any shape and move about the screen. Since they move independently of the screen image and move more smoothly than custom characters, they are often used when creating games or demonstrating animation.

Sprites are accessed from BASIC by a series of POKEs. The Video Interface Controller (VIC-II chip) holds several registers which you manipulate to create and move sprites on your screen. Manipulating these VIC-II registers can get complicated, however, especially for the beginning programmer, because the routines require numerous POKEs for each sprite. Turning on and off various sprite functions can become confusing. Crossing the invisible *seam* on the 64's screen is especially cumbersome.

A solution is to add some new commands to BASIC to control the sprites. This article provides a method for adding three new commands to BASIC which will allow you to control sprites more easily.

If you're unfamiliar with the methods used to design and create sprites on the 64, refer to the accompanying section, "Sprite Creation," before you continue.

Modifying BASIC

The Commodore 64 is a flexible computer and it's possible to use the Random Access Memory (RAM) under the BASIC

Read Only Memory (ROM) for a modified BASIC. You make a duplicate of BASIC, place it in RAM, and then modify "RAM BASIC" to suit your needs. The technique was outlined by Jim Butterfield in his article "Commodore 64 Architecture," which appeared in the January 1983 issue of *COMPUTE!* magazine. It was also used in my article "Commodore 64 Hi-Res Graphics Made Simple," which appeared in the August 1983 issue of *COMPUTE!'s Gazette*. Refer to these two articles for other uses of this same process.

"Sprite BASIC," which I'll call my BASIC modification program, replaces three old keywords, LET, WAIT, and VERIFY, with three new keywords, OFF, MOVE, and SPRITE. Notice that the new keywords are the same length as the ones they replace. A new keyword has to be mapped exactly into the old keyword's spot in the keyword lookup table. Program 1 is the BASIC program which moves the BASIC ROM code to RAM, modifies it, and loads the new machine language routines into a safe area of memory. Machine language is an excellent method of programming sprite movements, since it is both very fast and very efficient. (Sprite BASIC extends from $C000 to $C0E2.)

Sprite BASIC is loaded into the Commodore 64 by typing in and running Program 1. When typing it in, be as accurate as possible, since an incorrect number may cause the computer to crash when you type RUN. To clear this, you'd have to switch it off and on again, erasing anything you'd already entered. To be safe, SAVE the program before running it for the first time, and use the "Automatic Proofreader" in Appendix J.

It will take the computer a minute or so to run the program. Be patient. When the READY prompt appears again, type in:

```
POKE 1,54
```

This switches on Sprite BASIC. If you want to return to Commodore (your original) BASIC, simply type in:

```
POKE 1,55
```

Since you can switch from the old BASIC to Sprite BASIC within programs with these POKEs, your program can contain both the old and new BASIC command words.

Sprite BASIC is also switched off by pressing the RUN/STOP and RESTORE keys simultaneously. Because the new BASIC tokenizes the new keywords, make sure you have

Sprite BASIC turned on as you enter your own program. The old keywords that were replaced cannot be used unless the old BASIC is switched back on.

The New Commands

After you've entered and switched on Sprite BASIC, you'll have three new commands available while you program sprites.

OFF <number>

This statement disables (turns off) the sprite designated by the number. Sprites are numbered from 0 to 7, so a number 8 or greater will give an ILLEGAL QUANTITY ERROR.

MOVE <number>, <number>, <number>

This new keyword enables (turns on) a sprite and places it at the desired location on the screen. The first number is the sprite's number (0–7). The next two numbers are the X and Y coordinates, respectively, of the sprite's upper left corner. Because the sprite display area is larger than the screen area, the X coordinate must be 24 or greater, while the Y coordinate must be 50 or greater for the sprite to be fully visible. Allowed values for the X coordinate range from 0 to 511, although those greater than 344 are totally off the screen. Y values can range from 0 to 255, but numbers greater than 250 are completely off the screen. Any number greater than the accepted range will cause an ILLEGAL QUANTITY ERROR message.

SPRITE <number>, <number>, <number>, <number>

This new statement defines a sprite. The first number is the number of the sprite being defined. The second number is the 64-byte data block where the values used to actually draw the sprite are stored. This number can have values from 0 to 255. For example, sprite data stored in memory locations 832 to 895 (cassette buffer) is block 13 (832/64=13). The third number in this command is the color of the sprite. The color codes are:

0 Black	4 Purple	8 Orange	12 Med Gray
1 White	5 Green	9 Brown	13 Light Green
2 Red	6 Blue	10 Light Red	14 Light Blue
3 Cyan	7 Yellow	11 Dark Gray	15 Light Gray

The fourth number determines the size of the sprite. If the number is 0, the sprite is normal size. A 1 entered here doubles

the sprite's width. If the number is 2, the sprite is doubled in height. Entering a 3 doubles *both* the width and the height.

Some Sample Programs

You're now ready to enter and run a couple of simple programs using Sprite BASIC. Both demonstrate how this new BASIC can be used for easy animation. The first program animates a sprite which looks like a butterfly by moving it as it changes its shape. Actually two sprites are used. The program displays first one, then the other, to simulate movement. To see this, LOAD and RUN Sprite BASIC, type NEW, switch on the new BASIC by typing POKE 1,54, and enter Program 2. Before you run it, SAVE it on tape or disk.

A peculiarity of the Commodore 64 concerning sprites is that there are actually two separate sections of the screen for the X, or horizontal, coordinates. An invisible seam runs all the way down the screen immediately after the 255th X coordinate. Normally, you would have to POKE a value into an additional register each time a sprite moved across this seam. Notice, however, that you don't have to do this when you use Sprite BASIC. After you enter Program 2 and type RUN, it moves the sprite smoothly across the seam from left to right. This is one of the advantages of using something like Sprite BASIC, for the computer does as much as possible for you.

To see a joystick-driven sprite, type in NEW and enter Program 3. Make sure that Sprite BASIC is loaded and enabled before you run Program 3. Plug a joystick into port 2 and you'll be able to maneuver the tie fighter-shaped sprite across the screen.

Just Starting

Using Sprite BASIC, you can create and move your own sprites with much more ease than if you had to POKE each register on your own. All you really have to do is design a sprite, calculate the DATA numbers, which allow the 64 to display it properly, and the new BASIC does all the rest.

This lets you concentrate on creating unique sprites, or in using them to your program's advantage. A game, for example, would be much easier to program, with sprites, using this new programming tool. Try some of your own sprites, perhaps simply replacing the DATA numbers in the sample programs with your own sprite information.

Sprite Creation

Gregg Keizer

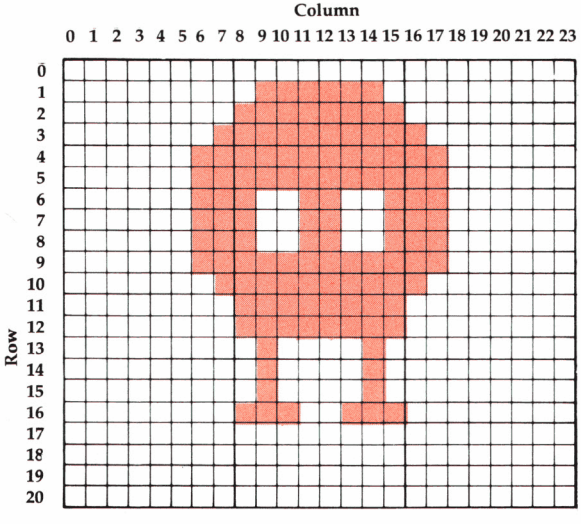
Drawing Sprites

Creating a sprite is much like creating a custom character—it must be drawn. The 64 does not do this for you; you have to place the data information within a program for the computer to look at, and then *draw* the sprite on the screen.

A sprite is much larger than a custom character, consisting of a graphics block 24 pixels wide by 21 pixels high. A custom character is only an 8-by-8 pixel block. The information to draw a sprite uses more memory than a custom character because of its size, so fewer sprites can be displayed at a time. Eight sprites are available to you on the Commodore 64.

Just as when you create custom characters, you can use graph paper to design your sprites. Take a piece of graph paper and outline an area 24 blocks wide by 21 high. Simply fill in the blocks in the pattern to create a sprite. Figure 1 shows a sample sprite drawn in this way.

Figure 1. Graphing a Sprite



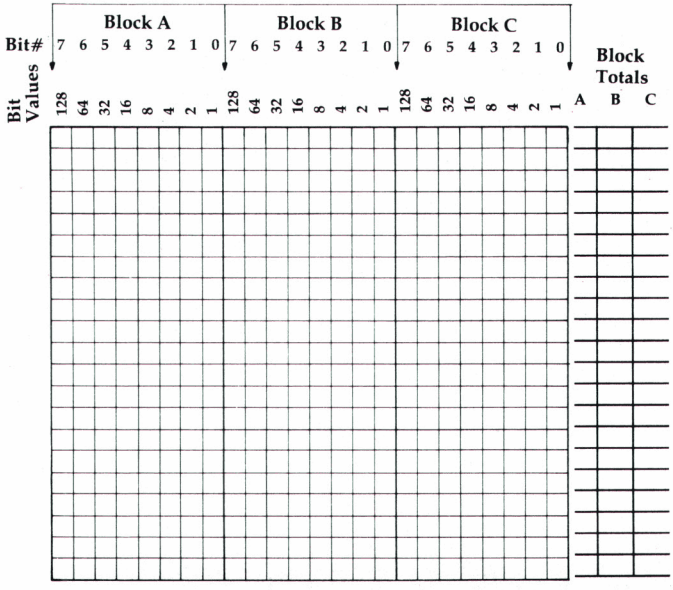
The blocks that are filled in will be *on*, or displayed in the color you later select for your sprite, while the empty blocks will be *off*, or shown in the screen's background color.

Drawing sprites is not enough for the computer, however. It cannot just look at something and display it on the screen. Instead, it needs numbers it can refer to which *tell* it what to create. You have to do this.

Bit Values

To come up with the numbers the 64 needs to draw your sprites, you'll have to do some addition. As when creating custom characters, to show some of a sprite's pixels *on* and others *off*, bits have to be set. It's not as hard as it sounds. Figure 2 shows you how it's done.

Figure 2. Sprite Worksheet



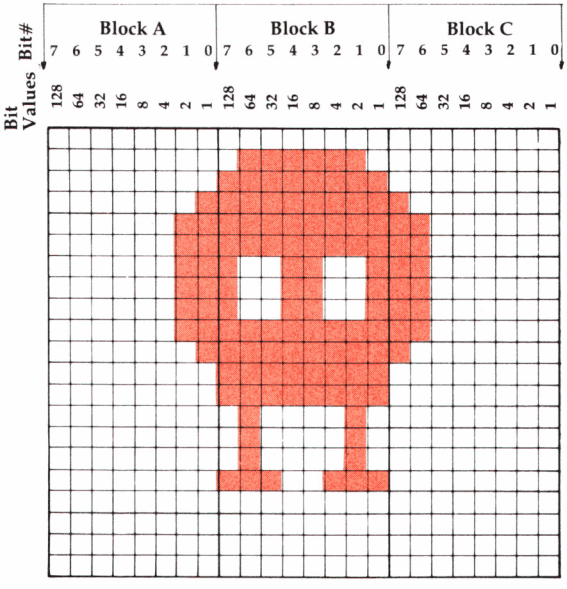
This is similar to the graph paper you used to design your sprite, only bit values have been assigned to each pixel. As in Figure 1, there are 24 columns and 21 rows. Each box represents one pixel in your sprite.

The similarity with custom characters ends here. Instead

of only one eight-bit block in each row, a sprite has three. These have been named Block A, Block B, and Block C in Figure 2. When the 64 looks at the numbers to create a sprite, it starts with the eight-bit block in the upper left corner, moves across the first row, and then jumps down to the left-most block on the next row. The last number it reads to create a sprite represents the bottom right corner of Block C.

Calculating the bit values to show a sprite is only a matter of adding together the values of the bits you want *on*. Figure 3 shows the same sample sprite, but with its bit values computed.

Figure 3. Sprite Computation



Block Totals A	B	C
0	0	0
0	126	0
0	255	0
1	255	128
3	255	192
3	255	192
3	153	192
3	153	192
3	153	192
3	255	192
1	255	128
0	255	0
0	255	0
0	66	0
0	66	0
0	66	0
0	231	0
0	0	0
0	0	0
0	0	0
0	0	0

The first row has none of its pixels on, so the bit value for all three bytes is 0. Row 2, however, has six bits in the Block B byte turned on. These bits, numbers 1 through 6, have a total bit value of 126 (2+4+8+16+32+64). The other two bytes, represented by Blocks A and C, are 0, since neither has any bits on.

Each byte is calculated in this same way. Remember that each row of a sprite consists of three bytes, and that each

must be figured separately. Figure 2 makes this simple, for each byte has its own total column at the far right.

When you've finished computing the bit values for a sprite, you should have 63 numbers. These are the numbers the Commodore 64 will look at to display your sprite. Normally, you would insert them in a program in several DATA statements and have the computer READ from this table. For instance, using the numbers for the sample sprite, the DATA statements would look like this:

```
DATA 0,0,0,0,126,0,0,255,0
DATA 1,255,128,3,255,192,3,255,192
DATA 3,153,192,3,153,192,3,153,192
DATA 3,255,192,1,255,128,0,255,0
DATA 0,255,0,0,66,0,0,66,0
DATA 0,66,0,0,231,0,0,0,0
DATA 0,0,0,0,0,0,0,0,0,-1
```

(The -1 is used to fill up the 64-byte block each sprite occupies in memory. Without that additional number, you may get an error message.)

Every sprite you design is created like this. But once you have it designed, you have to POKE other values into the 64 to make it appear.

Normally, you would have to POKE values into the computer to do such things as enable the sprite (turn it on), locate the sprite's DATA in an available memory address, set its color, and finally, place it on the screen. This is where sprite creation becomes tedious. By modifying BASIC, you can get the Commodore 64 to do much of this for you. "Sprites Made Easy" gives a detailed description on how to make sprite control easier.

Program 1. Sprite BASIC

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

```
10 A=0: REM INITIALIZE CHECKSUM                :rem 114
20 REM MOVE BASIC ROM TO RAM                    :rem 80
30 FORI=40960TO49151:POKEI,PEEK(I):NEXTI       :rem 217
40 REM CHANGE LET TO OFF                        :rem 81
50 FORI=41150TO41152:READN:POKEI,N:A=A+N:NEXTI
                                                :rem 113
60 READL,H:POKE40988,L:POKE40989,H:A=A+L+H:rem 254
70 DATA 79, 70, 198, 2, 192                     :rem 120
```

```
80 REM CHANGE WAIT TO MOVE                          :rem 1
90 FOR I=41189TO41192:READN:POKEI,N:A=A+N:NEXTI
                                                    :rem 133
100 READL,H:POKE41008,L:POKE41009,H:A=A+L+H :rem 9
110 DATA 77, 79, 86, 197, 19, 192                   :rem 123
120 REM CHANGE VERIFY TO SPRITE                     :rem 108
130 FORI=41201TO41206:READN:POKEI,N:A=A+N:NEXTI
                                                    :rem 157
140 READL,H:POKE41014,L:POKE41015,H:A=A+L+H :rem 7
150 DATA 83,80,82,73,84,197,96,192                  :rem 163
160 REM READ IN NEW ROUTINES                        :rem 145
170 FORI=49152TO49384:READN:POKEI,N:A=A+N:NEXTI
                                                    :rem 189
180 IFA<>30780THENPRINT"ERROR IN DATA STATEMENTS"
                                                     :rem 40
190 END                                             :rem 113
200 DATA 80, 70, 83, 32,158,183,224, 8,176, 31,189
    ,219,192, 45, 21,208,141                        :rem 237
210 DATA 21,208, 96, 32,158,183,224, 16,176, 14,13
    4, 2, 32,253,174, 32,235                        :rem 222
220 DATA183,165, 21,201, 2,144, 3, 76, 72,178,138,
     72,166, 2, 32, 10,192                          :rem 70
230 DATA189,219,192, 45, 16,208,141, 16,208, 70, 2
    1,144, 9,189,211,192, 13                        :rem 28
240 DATA 16,208,141, 16,208,138, 10,170,104,157, 1
    ,208,165, 20,157, 0,208                         :rem 255
250 DATA166, 2,189,211,192, 13, 21,208,141, 21,208
    , 96, 32,158,183,224, 8                         :rem 228
260 DATA176,193,134, 2,169,248,133,251,173, 24,208
    , 41,240, 9, 12,133,252                         :rem 70
270 DATA173, 0,221, 73,255, 74,102,252, 74,102,252
    , 32,253,174, 32,158,183                        :rem 65
280 DATA138,164, 2,145,251, 32,253,174, 32,227,192
    ,224, 16,176,146,138,153                        :rem 179
290 DATA 39,208,185,219,192, 72, 45, 29,208,141, 2
    9,208,104, 45, 23,208,141                       :rem 86
300 DATA 23,208, 32,253,174, 32,227,192,224, 4,176
    ,223,134, 2, 70, 2,144                          :rem 160
310 DATA 9,185,211,192, 13, 29,208,141, 29,208, 70
    , 2,144, 9,185,211,192                          :rem 178
320 DATA 13, 23,208,141, 23,208, 96, 1, 2, 4, 8, 1
    6, 32, 64,128,254,253                           :rem 171
330 DATA251,247,239,223,191,127                      :rem 43
340 DATA 32,158,183,164,2,96                        :rem 104
```

Program 2. Butterfly

```
10 READ SB: IF SB<0 THEN 180: REM READ SPRITE DATA
                                                :rem 207
20 LO= SB*64: FOR I= 0 TO 62                     :rem 69
30 READ SD: POKE LO+I,SD: NEXT I                 :rem 19
40 GOTO 10                                      :rem 254
50 DATA 13: REM SPRITE DATA BLOCK 13            :rem 193
60 DATA 14, 32, 0, 31, 112, 0, 63, 112, 0, 63, 186
   , 0                                          :rem 235
70 DATA 127, 217, 128, 127, 237, 128, 63, 247, 0,
   {SPACE}63, 254, 0                            :rem 111
80 DATA 31, 252, 0, 15, 248, 0, 15, 240, 0, 31, 22
   4, 0                                          :rem 31
90 DATA 31, 192, 0, 13, 128, 0, 0, 0, 0, 0, 0, 0
                                                :rem 230
100 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0                                           :rem 227
110 DATA 14: REM SPRITE DATA BLOCK 14           :rem 240
120 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 50, 0:rem 6
130 DATA 60, 121, 128, 127, 125, 128, 255, 191, 12
    8, 255, 239, 0                              :rem 48
140 DATA 255, 254, 0, 255, 252, 0, 255, 248, 0, 12
    7, 240, 0                                   :rem 41
150 DATA 63, 224, 0, 127, 192, 0, 62, 0, 0, 28, 0,
    0                                           :rem 189
160 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0                                           :rem 233
170 DATA -1: REM END OF DATA                    :rem 180
180 PRINT"{CLR}": POKE 53281,1: REM WHITE SCREEN
                                                 :rem 37
190 X=0: Y=0: REM STARTING POSITION              :rem 94
200 POKE 1,54: REM TURN ON SPRITE BASIC         :rem 196
210 VERIFY 0, 13, 11, 0: REM DEFINE SPRITE:rem 178
220 WAIT 0, X, Y: REM PUT SPRITE 0 ON SCREEN
                                                :rem 125
230 FOR T = 0 TO 100: NEXT T: REM DELAY LOOP:rem 5
240 VERIFY 0, 14, 11, 0: REM REDEFINE SPRITE
                                                 :rem 77
250 FOR T = 0 TO 100: NEXT T: REM DELAY LOOP:rem 7
260 X=X+3: Y=Y+ 3*(INT(RND(1)*3)-1)              :rem 66
270 IF X>345 THEN X=0                            :rem 78
280 IF (Y<30) OR (Y>250) THEN Y=150             :rem 237
290 GOTO 210                                    :rem 103
```

Program 3. Tie Fighter

```
10 READ SB: IF SB<0 THEN 120: REM READ SPRITE DATA
                                              :rem 201
20 LO= SB*64: FOR I= 0 TO 62                   :rem 69
30 READ SD: POKE LO+I, SD: NEXT I              :rem 19
40 GOTO 10                                    :rem 254
50 DATA 13: REM SPRITE DATA BLOCK 13          :rem 193
60 DATA 192, 0, 3, 192, 0, 3, 192, 40, 3, 192, 171
   , 3                                        :rem 253
70 DATA 194, 171, 195, 194, 155, 195, 194, 90, 195
   , 194, 90, 195                             :rem 95
80 DATA 194, 106, 195, 250, 170, 235, 254, 170, 23
   9, 254, 170, 255                           :rem 163
90 DATA 194, 171, 195, 194, 175, 195, 195, 191, 19
   5, 195, 255, 195                           :rem 202
100 DATA 195, 255, 195, 192, 255, 3, 192, 60, 3, 1
    92, 0, 3                                   :rem 8
110 DATA 192, 0, 3, -1: REM END OF SPRITE DATA
                                               :rem 8
120 X= 184: Y= 150: REM POSITION IN MIDDLE OF SCRE
    EN                                         :rem 89
130 PRINT"{CLR}":POKE 53281,3: POKE53280,3: REM CY
    AN SCREEN AND BORDER                       :rem 39
140 POKE 53276,1: REM MULTICOLOR SPRITE 0      :rem 87
150 POKE 53285,15: POKE 53286,11: REM AUX COLORS
                                               :rem 59
160 POKE1,54: REM TURN ON SPRITE BASIC        :rem 201
170 VERIFY 0, 13, 12, 1: REM DEFINE SPRITE 0
                                              :rem 233
180 WAIT 0, X, Y: REM POSITION SPRITE         :rem 113
190 GOSUB 300: IF J=15 THEN 190                :rem 44
200 GOTO 180:REM MOVE SPRITE                  :rem 144
300 REM READ JOYSTICK                           :rem 3
310 J= PEEK(56320) AND 15: REM PORT 2          :rem 95
320 IF (J AND 8)=0 THEN X=X+1: REM MOVE RIGHT
                                              :rem 130
330 IF (J AND 4)=0 THEN X=X-1: REM MOVE LEFT
                                               :rem 46
340 IF (J AND 2)=0 THEN Y=Y+1: REM MOVE UP:rem 167
350 IF (J AND 1)=0 THEN Y=Y-1: REM MOVE DOWN
                                               :rem 60
360 IF Y<50 THEN Y=50: REM STAY IN RANGE      :rem 175
370 IF Y>229 THEN Y=229                       :rem 191
380 IF X<24 THEN X=24                          :rem 78
390 IF X>295 THEN X=295                       :rem 197
400 RETURN                                    :rem 116
```

5

Inside Your 64

Using the Function Keys:
A BASIC Tutorial

Charles Brannon

Perhaps you've pressed those function keys to the right of the keyboard and were dismayed to find they did nothing. Don't worry, they work fine; they just need a program to "come alive." With this tutorial, you'll find it's easy to write your own programs using function keys.

One day, somebody had a good idea. There were dozens of programs: word processors, spreadsheets, data bases, and they all required you to press certain keys to perform the various functions. For example, a word processor would save your text to disk with CTRL-S (meaning to hold down a special ConTRoL key while you press S). The arrow keys that move the cursor were among the first "function keys"; they replaced various CTRL-keys that did the same thing.

Mystery Keys

So someone added a number of mysterious keys to a computer keyboard. Dedicated (used only for one task) word processors have special labelled keys to cut, paste, copy, edit, etc. Since computers are general-purpose, the keys had to be unlabelled so every application could do something different with the keys. The idea caught on. These days, function keys are the rage. You can hardly buy a computer without them.

Special, set-aside, unlabelled function keys are defined by whatever program is currently running. Frequently, programmers assign powerful functions to the keys. It's a gimmick of sorts; it would be just as easy to assign the function to the normally unused CTRL keys (and link them in an easy-to-

remember fashion, such as CTRL-Q for Quit, CTRL-E to Erase, etc.). There is undeniable convenience, however, in having your own special "programmable" keys.

The Sad Truth

Fundamentally, the function keys are no different from any other key on the keyboard, so it is as unrealistic to assume they'll always do something as it is to think that pressing the fire button on the joystick will always fire a shot. If you've used the joystick, you know that it tells you only which way the player is pushing (north, south, east, west, or diagonal) and whether the fire button is pressed or not. Period. You have to write (or buy) special programs that move a figure based on the position of the joystick.

The function keys on the Commodore 64 are the same. When you run commercial software, the keys do everything from changing border colors to shifting the screen, selecting difficulty, or restarting a game. But the real power of the function keys comes when you understand how to use them in your own programs.

GETting to the Point

The primary BASIC command used to read the keyboard is GET. When you type GET followed by a variable name (GET A$ or GET XZ), the computer looks at the keyboard and puts whatever key is being pressed into the variable. But it looks only once, and if you didn't press a key, the computer merrily goes on to something else. GET will not wait for a key to be pressed. This is a good feature; but if you do want to wait for a key, you would do something like:

```
10 GET A$
20 IF A$="" THEN 10
```

or

```
10 GET N
20 IF N=0 THEN 10
```

The phrase: IF A$="" means: if A-string equals the null string (nothing is between the quotes; it's just two quotes in a row), then go back to line 10. So as long as no key is pressed, line 20 will keep sending the computer back to line 10 to check again. The second example is waiting for you to press a number key from 1–9 (it uses 0 to mean no key pressed, so

pressing 0 won't make it stop waiting). This type of GET command used with a numeric variable (instead of a string) is dangerous, though. If the user presses any other key, the program will crash (stop running and return to BASIC) with a ?SYNTAX ERROR message. It's just as easy to convert a string into a number with the VAL command, so the second statement could be rephrased:

```
10 GET N$
20 IF N$="" THEN 10
30 N=VAL(N$)
```

It's easy to improve; if you wanted to accept only numbers above, you could change line 20 to:

```
20 IF N$<"0" OR N$>"9" THEN 10
```

which means: if N-string has an ASCII value (a code used in your computer to order characters—A, which has an ASCII value of 65 is "less than" Z, which has an ASCII code of 90) less than that of "0" or greater than that of the character "9" then loop back to line 10.

Incidentally, the ASCII code for the null string (quote-quote) is zero, which is less than 48, the code for "0", so the loop will also wait for a key. If you're curious about ASCII, check out the BASIC commands ASC and CHR$ in your manual. You'll also find the ASCII codes and their character equivalents in Appendix F at the back of this book.

Strictly Logical?

So if you just want to accept a yes or no answer (Y for Yes, N for No), then this will work just fine:

```
10 GET A$:IF A$<>"Y" AND A$<>"N" THEN 10
```

Computer logic with IF-THEN, AND, OR, and NOT can get a bit tricky, so let me explain this line. The computer will GET a key and put it into A$. Remember that the user may not have pressed the key yet, so A$ could be any key, or it could be the null string (""). In the latter case, the null string is not equal to "Y" *and* it is not equal to "N," so it will loop back to 10. If you pressed "X," it will also loop. But if you pressed "Y," A$ would be equal to "Y" (meaning A$<>"Y" is false) but it would not equal "N" (A$<>"N" is true). Since both conditions are not true, AND fails, and the program continues. A common mistake would be:

```
10 GET A$:IF A$<>"Y" OR A$<>"N" THEN 10
```

This would loop back to line 10 no matter what key was pressed. If either A$ did not equal "Y" *or* A$ did not equal "N," then the computer would loop. The only way for the test to fail would be for A$ to be "not equal" to "Y" and "not equal" to "N"; in other words, it would have to be both equal to "Y" *and* equal to "N." I told you it was tricky! By the way, another common mistake is something like:

```
10 GET A$:IF A$<>"Y" AND <>"N" THEN 10
```

This will give you a ?SYNTAX ERROR, but it seems to read all right in English. It's just that the computer requires you to repeat the variable for each symbol such as <>, <,>, or =.

If you've tried some of the examples, you'll find that GET only changes the value of the variable. It does not print the key on the screen. This is also handy; you don't want a bunch of keys printed out just to move your spaceship using the keyboard. To make a simple "video typewriter," try this (remember the semicolon on line 20):

```
10 GET X$:IF X$="" THEN 10
20 PRINT X$;:GOTO 10
```

On to Great Frontiers

We're nearly ready to use the function keys. Try this: press the quote (SHIFT-2) and then press the function keys (SHIFT to get the even-numbered keys). What magic is this? Each key now seems to print some cryptic symbol! The computer can read the function keys just like any other key, but PRINTing them won't display anything unless you are in quote mode (where you can program cursor controls into PRINT statements). But you can take advantage of the symbols to easily interpret the function keys. You use GET to read them, of course. Try this program:

```
10 GET F$:IF F$="" THEN 10
20 IF F$="{F1}" THEN PRINT"FUNCTION ONE"
30 IF F$="{F2}" THEN PRINT"FUNCTION TWO"
40 IF F$="{F3}" THEN PRINT"FUNCTION THREE"
50 IF F$="{F4}" THEN PRINT"FUNCTION FOUR"
60 IF F$="{F5}" THEN PRINT"FUNCTION FIVE"
70 IF F$="{F6}" THEN PRINT"FUNCTION SIX"
80 IF F$="{F7}" THEN PRINT"FUNCTION SEVEN"
90 IF F$="{F8}" THEN PRINT"FUNCTION EIGHT"
```

The {F1}, {F2}, and so on, mean for you to press the appropriate function key inside the quotes. You'll get the aforementioned symbols. See Appendix B for the symbols printed on the screen when you press each function key.

What will you do with the function keys? It's really up to you. For example, to restart a game, you might do something like this:

```
530 PRINT"PRESS F1 TO PLAY AGAIN"
540 GET A$:IF A$<>"{F1}" THEN 540
```

You could also organize a bunch of subroutines, one for each key, that does something associated with the key (maybe eight sound effects):

```
10 GET RQ$:IF RQ$="" THEN 10
20 IF RQ$="{F1}" THEN GOSUB 500
90 IF RQ$="{F8}" THEN GOSUB 1000
```

Each function key also has a corresponding ASCII number. Try this program. It prints out the ASCII (ordered) value for any key pressed:

```
10 GET A$:IF A$="" THEN 10
20 PRINT CHR$(34);A$;CHR$(34),ASC(A$)
30 GOTO 10
```

The CHR$(34) puts the computer in quote mode so that if you press CLR/HOME or something, you'll see its symbol instead of the screen clearing.

Here is a summary of the ASCII values for the function keys:

f1: 133 f2: 137
f3: 134 f4: 138
f5: 135 f6: 139
f7: 136 f8: 140

They're in order from f1–f7, and f2–f8, separately. So you could use a statement like this to check for f6:

```
342 IF F$=CHR$(139) THEN PRINT "FUNCTION SIX"
659 IF ASC(F$)=139 THEN GOSUB 4153
```

See how CHR$ and ASC work?

You Take It from Here

Now that you've got the word on function keys, you can start making your programs "user friendly" too. And you can share

a double feeling of power: not only does pressing one key raise your garage door, put out the cat, and make coffee in the morning, but you also know that you're the one that made the computer do it.

How to Use Arrays

Don Stauffer

Using arrays is a handy BASIC programming technique. This tutorial explains what they are and how to use them when programming on your 64.

Arrays, sometimes called subscripted variables, are an important feature of Microsoft BASIC, but there is little documentation on what they are and how to use them. This is particularly true of the 64.

Some time ago, a friend of mine, a new computer owner, called with a programming problem. He was working on a program in which he needed to generate random numbers for a variable (R). However, he wanted ten different values for R and wanted to save them for later use in the program, in statements where he would use these R values in calculations. I told him that was a perfect spot to use an array. After he looked up arrays in all the reference books he had on the machine, he wasn't much better off than when he first called, so we spent a session going over arrays. It seemed to me that the best way to know how to use arrays was to start with the basics.

What is an Array?

An array is a type of variable which can have a number of values at any one time. For instance, let's look at a variable, T, which might stand for the maximum temperature for a particular day. T(1) might be the temperature of day 1, T(2) the temperature of day 2, and so on. The number in the parentheses is called the *subscript.* In fact, arrays are sometimes called *subscripted variables.* Although the best way to understand arrays is through examples, which we'll get to shortly, we should first learn a little about how the computer stores and uses arrays.

Since an array is a set of several values, it obviously takes

more memory than a normal variable. In fact, unless the computer knows how many values your variable will have, it does not really know how much memory to set aside for that variable. We tell the computer this information with a DIMension statement:

```
DIM X(15),Y(20)
```

In this example, we told the program we were going to use two arrays, X and Y, and that X would have a maximum of *16* values, and Y would have a maximum of *21*. Notice that the number of values set up is always one greater than the number specified in the DIM statement. Although it's confusing, this is because the computer starts counting with 0, not 1. To avoid confusion, some programmers simply ignore the 0 and treat X(15) as an array of 15 values. This wastes a tiny amount of memory, but it usually doesn't matter.

With the 64, the DIMension statement is optional unless you are going to use more than 11 values. I recommend, however, that you always DIMension arrays, even if they will have less than 11 values. It is good programming practice, and it will save considerable memory since the computer will not set aside unnecessary memory space. Also, the DIM statement initially sets all array values to zero. Good programming practice dictates that the array should be DIMensioned in one of the first statements of the program, and it obviously must occur before any reference to the array. The DIM statement must not be executed more than once, however, or an error results.

A particular value of an array is called an element. Each element is referred to by a subscript, which is why the array is sometimes called a subscripted variable. In the following statement:

```
LET X(5)=27.3
```

element 5 of the X array is set to 27.3. Whenever the computer comes across a set of parentheses with a number enclosed following a variable name, it knows you are indicating an array. From now on, we will call each separate value in an array an *element*. In our previous DIMension statement, we indicated that X would have 16 elements, and Y would have 21. In the assignment statement, we set element 5 of the X array to 27.3.

As an example of the use of arrays, let's take a look at Program 1, which is part of my friend's program.

Program 1. Arrays and Average Values

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

```
10 PRINT"{CLR}":DIM R(10)                          :rem 221
100 FOR N=1 TO 10                                   :rem 56
110 R(N)=INT(RND(1)*10+1)                           :rem 73
120 NEXT N                                          :rem 32
130 REM MAIN PART OF{6 SPACES}PROGRAM FOLLOWS
                                                    :rem 167
140 GOSUB 500                                       :rem 170
150 PRINT:PRINT"PRESS A KEY TO COMPUTE";:PRINT"
    {3 SPACES}ANOTHER AVERAGE"                      :rem 247
160 GETA$:IF A$=""THEN 160                          :rem 81
170 PRINT"{CLR}":GOTO100                             :rem 0
500 REM SUBROUTINE FOR{4 SPACES}COMPUTING AVERAGE
    {5 SPACES}R                                     :rem 115
510 SM=0                                            :rem 163
515 PRINT:PRINT"{RVS}ARRAY{OFF}{2 SPACES}{RVS}VALU
    ES{OFF}":PRINT                                  :rem 145
520 FOR N=1 TO 10                                   :rem 62
530 SM=SM+R(N)                                      :rem 49
535 PRINT"R(";N;")=";R(N)                           :rem 130
540 NEXT N                                          :rem 38
550 AV=SM/10                                        :rem 158
560 PRINT:PRINT"AVERAGE =";{5 SPACES}AV             :rem 61
570 RETURN                                          :rem 124
```

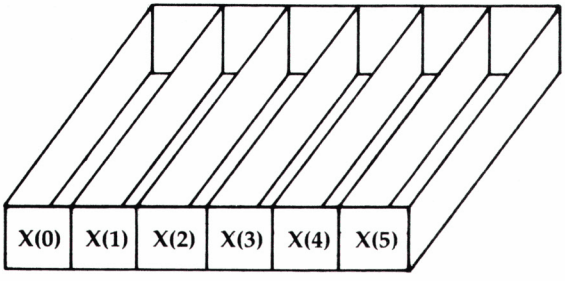
Line 10 contains the DIMension statement. Lines 100–120 assign ten random numbers to the ten locations or variables of the R array. The main part of the program is irrelevant to our discussion of arrays, but the subroutine starting at line 500 uses the array further and is a good example. The program is written to find the average value of the ten numbers. The sum is first set to zero in line 510. The FOR-NEXT loop (lines 520–540) recalls the values stored previously in line 110 and computes the sum, which is divided by ten to compute the average in line 550.

Two-Dimensional Arrays

Arrays can have more than one dimension. The arrays we've seen so far are one-dimensional. We can visualize the one-dimensional array as a line of boxes or pigeonholes, as in Figure 1, in which to place values, or a list of values like a list on a piece of paper.

Figure 1. One-Dimensional Array

A one-dimensional array can be thought of as a row of boxes or pigeonholes.



DIMX(S)

The one-dimensional array is probably the most common, but the two-dimensional array is used often, too. The two-dimensional array is often visualized as a table of rows and columns. For instance, an array DIMensioned by the statement:

DIM X(4,3)

would be visualized as a table of *five* columns by *four* rows, as shown in Figure 2. Again, notice that DIM X(4,3) actually sets up a 5x4 table because the elements are numbered starting with 0. As with one-dimensional arrays, you may choose to ignore the 0 column and row, spending a few bytes of memory to eliminate a possible source of confusion.

Figure 2. Two-Dimensional Array

A two-dimensional array is frequently visualized as a table of rows and columns.

	Column 0	Column 1	Column 2	Column 3	Column 4
Row 0	X(0,0)	X(1,0)	X(2,0)	X(3,0)	X(4,0)
Row 1	X(0,1)	X(1,1)	X(2,1)	X(3,1)	X(4,1)
Row 2	X(0,2)	X(1,2)	X(2,2)	X(3,2)	X(4,2)
Row 3	X(0,3)	X(1,3)	X(2,3)	X(3,3)	X(4,3)

Frequently, a particular problem can be solved by either a one- or a two-dimensional array, and the choice is strictly a matter of style, up to the programmer. Programs 2 and 3 illustrate a similar problem, the first with a one-dimensional array, and the second with a two-dimensional array.

In Program 2, the problem is to record the high temperature for each day, and then find the average high temperature for the week.

Program 2. One-Dimensional Array

```
20 DIM TM(7)                                        :rem 101
30 REM ENTER DATA                                   :rem 223
40 INPUT "{CLR}ENTER DAY NUMBER";N                  :rem 121
50 PRINT:PRINT "ENTER HIGH TEMPERATURE FOR DAY":IN
   PUT TM(N)                                        :rem 184
60 IF N<7 THEN 40                                    :rem 73
70 REM                                               :rem 75
80 REM A SUBROUTINE, NOT SHOWN HERE, WOULD STORE T
   HE ARRAY TO TAPE                                 :rem 222
100 REM                                             :rem 117
120 GOSUB 1000                                      :rem 212
130 END                                             :rem 107
1000 REM ROUTINE FOR FINDING AVERAGE HIGH TEMPERAT
     URE                                             :rem 26
1010 REM A ROUTINE FOR READING THE TAPE, NOT SHOWN
     , WOULD BE INCLUDED HERE                        :rem 79
1030 PRINT                                           :rem 81
1040 SM=0                                           :rem 210
1050 FOR N=1 TO 7                                    :rem 67
1060 SM=SM+TM(N)                                    :rem 175
1065 PRINT"DAY";N;"TEMP=";TM(N)                     :rem 113
1070 NEXT N                                          :rem 85
1080 AV=INT(SM/7)                                   :rem 223
1090 PRINT:PRINT"AVERAGE HIGH":PRINT"TEMPERATURE F
     OR WEEK=";AV;" DEGREES                          :rem 84
1100 RETURN                                         :rem 162
```

The one-dimensional array TM is DIMensioned to 7. An actual application program would have some sort of data file routines, but since tape or disk file handling is another subject altogether, let's leave the storage and retrieval out. Lines 40 and 50 assign the value of the high temperature to the appropriate box in the array. The average high temperature is then found in the subroutine starting at line 1000, in the same manner as in the preceding problem.

Program 3 handles a similar problem using a two-dimensional array.

Program 3. Two-Dimensional Array

```
20 DIM TM(52,7)                                    :rem 248
30 REM ENTER DATA                                  :rem 223
40 INPUT"{CLR}ENTER WEEK NUMBER ";WK               :rem 27
50 INPUT"ENTER DAY OF WEEK ";DY                    :rem 46
60 PRINT"ENTER HIGH TEMPERATURE":INPUT TM(WK,DY)
                                                   :rem 74
65 IF WK<52 THEN 40                                :rem 210
70 REM                                             :rem 75
80 REM SUBROUTINE 500, NOT SHOWN HERE, WOULD STORE
                                                   :rem 5
90 REM DATA ON TAPE                                :rem 46
100 REM GOSUB 500 TO TAPE ROUTINE HERE             :rem 161
110 GOSUB 1000                                     :rem 211
120 END                                            :rem 106
1000 REM READ TAPE AND COMPUTE AVERAGE             :rem 214
1010 REM A TAPE READ ROUTINE, NOT SHOWN, WOULD BE
     {SPACE}FOUND HERE                             :rem 221
1030 REM                                           :rem 168
1040 S1=0                                          :rem 182
1050 FOR W=1 TO 52                                 :rem 124
1060 S2=0                                          :rem 185
1070 FOR D=1 TO 7                                  :rem 59
1080 S1=S1+TM(W,D)                                 :rem 242
1090 S2=S2+TM(W,D)                                 :rem 245
1100 NEXT D                                        :rem 69
1110 WA=S2/7                                       :rem 131
1120 PRINT"WEEK ";W;"AVERAGE IS ";WA;"DEGREES"
                                                   :rem 186
1130 NEXT W                                        :rem 91
1140 YA=S1/365                                     :rem 238
1150 PRINT"YEARLY AVERAGE HIGH TEMP"               :rem 191
1160 PRINT"IS ";YA;" DEGREES"                      :rem 136
1170 RETURN                                        :rem 169
```

In this version, we store the temperatures week by week and day by day in a table of 52 rows of 7 columns (line 20). We have a column for every day of the week, and a row for every week of the year. The first part of the program stores our data in the array by week number and the number of the day in the week. The subroutine starting at line 1000 again figures the average, but with a new twist (as an advantage of using

the two-dimensional array). Now we can find the average temperature for each week as well as for the year.

Another Use of Arrays

Another handy use of arrays is to relate two sets of values to one another. This can easily be done if each set of values is an array, and these values can then be related by the subscript. A common use of arrays for this purpose is relating a set or sets of values to people's names. The names are held in a *string array,* such as N$(X), while the values are held in *numeric arrays* (having the same dimensions as N$, of course). Program 4 illustrates the use of arrays in a teacher's gradebook program.

Program 4. Arrays and Grades

```
20 DIM N$(15),T1(15),T2(15),HW(15),FS(15)  :rem 52
30 PRINT"{CLR}"                           :rem 199
40 REM DISPLAY MENU                       :rem 147
50 PRINT"{4 SPACES}{RVS}SELECT OPTION{OFF}"
                                          :rem 115
60 PRINT:PRINT"1-ENTER NAMES IN FILE"       :rem 5
70 PRINT:PRINT"2-ENTER SCORES, FIRST{3 SPACES}TEST
   "                                      :rem 159
80 PRINT:PRINT"3-ENTER SCORES, SECOND{2 SPACES}TES
   T"                                     :rem 213
90 PRINT:PRINT"4-ENTER SCORES,{9 SPACES}HOMEWORK"
                                           :rem 71
100 PRINT:PRINT"5-COMPUTE FINAL SCORE"    :rem 142
110 PRINT:INPUT"ENTER NUMBER";Q             :rem 0
120 ON Q GOSUB 1000,2000,3000,4000,5000   :rem 128
130 END                                   :rem 107
1000 REM INITIALIZE{7 SPACES}STUDENT NAME FILE
                                          :rem 255
1010 FOR N=1 TO 15                        :rem 110
1020 INPUT"ENTER LAST NAME";N$(N)         :rem 182
1030 NEXT                                   :rem 3
1040 OPEN 1,1,2,"NAMES"                   :rem 199
1050 FOR N= 1 TO 15                       :rem 114
1060 PRINT#1,N$(N)                        :rem 229
1070 NEXT N                                :rem 85
1080 CLOSE 1                              :rem 112
1090 RETURN                               :rem 170
2000 REM ENTER TEST SCORES                 :rem 51
2010 OPEN 1,1,0,"NAMES"                   :rem 195
2020 FOR N=1 TO 15                        :rem 112
2030 INPUT#1,N$(N)                        :rem 230
```

```
2040 NEXT N                                          :rem 83
2045 CLOSE 1                                        :rem 114
2050 REM ENTER DATA BY{5 SPACES}NAME                :rem 255
2060 FOR N=1 TO 15                                  :rem 116
2070 PRINT"ENTER SCORE FOR ";N$(N)                  :rem 199
2080 INPUT T1(N)                                    :rem 126
2090 NEXT N                                          :rem 88
2100 REM NOW SAVE T1{7 SPACES}ARRAY AS FILE TO
     {6 SPACES}TAPE                                  :rem 79
2110 OPEN 2,1,2,"TEST1"                             :rem 196
2120 FOR N=1 TO 15                                  :rem 113
2130 PRINT#2,T1(N)                                  :rem 248
2140 NEXT                                             :rem 6
2150 CLOSE 2                                        :rem 112
2160 RETURN                                         :rem 169
3000 REM NOW WOULD{9 SPACES}FOLLOW TWO MORE:rem 38
3010 REM SUBROUTINES{7 SPACES}LIKE THE ONE
     {10 SPACES}ABOVE, EXCEPT                        :rem 85
3020 REM REPLACE T1{8 SPACES}WITH T2 IN SUB-
     {7 SPACES}ROUTINE STARTING                      :rem 44
3030 REM AT LINE 3000,{5 SPACES}AND CALL THE FILE
     {5 SPACES}"TEST2".                              :rem 42
3040 REM THEN USE HW{7 SPACES}AND FILENAME
     {9 SPACES}"HMWRK" FOR THE                       :rem 15
3050 REM ROUTINE AT 4000                             :rem 43
4000 REM HOMEWORK FILE{5 SPACES}HERE                 :rem 88
5000 REM READ TAPE{9 SPACES}FILES AND COMPUTE
     {5 SPACES}SCORE                                 :rem 206
5010 OPEN 1,1,0,"NAMES"                             :rem 198
5020 FOR N=1 TO 15                                  :rem 115
5030 INPUT#1,N$(N)                                  :rem 233
5040 NEXT                                             :rem 8
5050 CLOSE 1                                        :rem 113
5060 OPEN 2,1,0,"TEST1"                             :rem 201
5070 FOR N=1 TO 15                                  :rem 120
5080 INPUT#1,T1(N)                                    :rem 1
5090 NEXT                                            :rem 13
5100 CLOSE 2                                        :rem 110
5105 INPUT"HIT RETURN TO CONTINUE";Q                :rem 248
5110 OPEN 3,1,0,"TEST2"                             :rem 199
5120 FOR N=1 TO 15                                  :rem 116
5130 INPUT#3,T2(N)                                    :rem 0
5140 NEXT                                             :rem 9
5150 CLOSE 3                                        :rem 116
5155 INPUT "HIT RETURN TO CONTINUE";Q               :rem 253
5160 OPEN 4,1,0,"HMWRK"                             :rem 228
5170 FOR N=1 TO 15                                  :rem 121
5180 INPUT#4,HW(N)                                   :rem 31
5190 NEXT N                                          :rem 92
```

```
5200 CLOSE 4                                        :rem 113
5210 REM NOW COMPUTE{7 SPACES}FINAL SCORE           :rem 163
5220 FOR N= 1 TO 15                                 :rem 117
5230 FS(N)=T1(N)+T2(N)+HW(N)                         :rem 28
5240 NEXT N                                          :rem 88
5250 REM NOW PRINT OUT{5 SPACES}SCORES              :rem 248
5260 OPEN 1,4,7                                     :rem 243
5270 PRINT#1,"NAME","SCORE"                          :rem 44
5280 FOR N = 1 TO 15                                :rem 123
5290 PRINT#1,N$(N),FS(N)                             :rem 82
5300 NEXT N                                          :rem 85
5310 RETURN                                         :rem 169
```

For demonstration purposes, this program is not a complete program as it stands, and contains no error trapping or user prompts. It could, however, be expanded into a useful gradebook program with some fill-in work. It is instructive of the use of arrays to relate variables. The main program, up to line 130, creates a menu selection which sends the program to the appropriate subroutine.

The first routine, starting at line 1000, is used at the beginning of the school term to enter the students' names in a string array, N$(N). The DIMension statement in line 20 of the main program, and all of the FOR-NEXT loops, would have to be adjusted to the actual number of students in the class. Subroutine 2000 would be used to enter the scores of the first test. By reading the N$ array in lines 2010 to 2045, the program prompts the teacher with the student's name for data entry (line 2070). A similar subroutine would be used for each test and maybe a homework score.

Subroutine 5000 puts it all together at the end of the term. After reading the grades from all the files, line 5230 figures the grade for every student. In effect, the variable N is a student number which relates each element of each of the four files. This illustrates how N can still be used as a separate variable, even when you've set up a numeric array N(X) or a string array N$(X).

These examples of the use of the array are general but easy to expand on. Arrays can be used in a variety of ways. I'm sure that after using them for a while, you can come up with many more applications on your own.

Adding New Keywords to BASIC

Sheldon Leemon

There are lots of programs available which will enhance your Commodore 64 by adding new keyword commands to BASIC. But learning how to program these additions yourself is rarely explained. This article, for programmers familiar with machine language, includes examples and the source code used to create five new keywords, and shows you how to program new BASIC commands yourself.

While Commodore 64 BASIC is a useful all-purpose language, it does have some limitations. There are no special graphics or sound commands to support the machine's bitmap or sprite graphics, or its superior musical abilities. It lacks error trapping, so that any error causes the program to stop. In fact, after a little thought, almost any programmer could come up with a "wish list" of new commands that he or she would like to see added to BASIC.

To overcome these limitations, some programmers devise machine language subroutines which allow them to simulate new BASIC commands. Often, however, they have problems integrating these new routines into the framework of the existing BASIC. The USR and SYS commands are most often used to add machine language routines to BASIC, but these commands do not easily pass values to the machine language program (if you were creating a DRAW command for hi-res graphics, for example, you would have to specify the screen position each time you used the command). And these commands are not always convenient, for their syntax is often

strange and their use requires you to know the address of each routine.

Wedges?

On the old Commodore PET machines, there was a way to add new commands to BASIC using what was called a "wedge." This is a routine that intercepts the part of the BASIC interpreter program that reads the program text. The wedge routine is designed to read the text before BASIC does, and compare that text to a list of new commands (like the short disk commands of the DOS support program). If one of these commands is spotted, the wedge executes the new routine. If not, control is handed over to the normal BASIC routines.

There are a couple of problems with this technique, however. The most important one is execution speed. Since the BASIC routine which the wedge diverts has to read every single character of the program being executed, the time that the wedge takes to check for each character for new commands can drastically slow down your program. The more commands added, the greater this slowdown. To counteract this effect, wedge commands are often set up to execute only in direct mode (that's why you can't call the DOS wedge from a program while it's running). Even so, a wedge as efficient as the DOS support program still slows down program execution a little. Another problem is that adding new commands with a wedge is hard to do in a way so they can be used simultaneously with DOS support and other wedge programs.

Fortunately, the 64 isn't limited to the wedge method of adding new commands. The 64 was designed to allow the addition of new commands which function exactly like regular BASIC commands, and which do not slow program execution. To explain how this is possible, however, first requires an explanation of how Commodore's Microsoft BASIC operates.

Microsoft BASIC

When you enter a line of BASIC program text, a tokenization routine scans the line to see if any of the words match its list of command keywords. When it finds such a word (like PRINT, for example) the routine replaces the ASCII characters of the keyword with a single character, called a *token*. Each token has a value of 128 or higher, and represents a single BASIC command. These tokens are interpreted by BASIC

when the RUN or LIST command is entered. When a program is RUN, the BASIC interpreter starts to read the program text. Each time it comes to a character with a value of 128 or greater (that isn't in quotes, or in a DATA or REM statement), it tries to execute the command which corresponds to that token. When you LIST the line, a detokenization routine expands the token from a single character back to its ASCII equivalent.

Therefore, in order to add new tokenized keywords to BASIC, and to be able to LIST and RUN them, you must have a way of intercepting the BASIC interpreter routines that tokenize, detokenize, and execute keywords. At first that might seem impossible, because the BASIC interpreter is in ROM, which cannot be changed. Nonetheless, it is possible to gain access to these routines via the BASIC Indirect Vector Table, which is in RAM. This table, which starts at location 768 ($300) and continues to 779 ($30B), contains the addresses of six crucial BASIC routines. They are IERROR (768–769), which prints BASIC error messages; IMAIN (770–771), the main program loop of BASIC which waits for you to enter a line after the READY prompt; ICRNCH (772–773), the routine that crunches the text of keywords into single-character tokens; IQPLOP (774–775), which expands those tokens back into ASCII characters; IGONE (776–777), which executes BASIC statement tokens; and IEVAL (778–779), which among other things evaluates tokenized BASIC functions (like INT and ASC). Whenever BASIC wants to execute one of these routines, it does not go directly to its ROM location, but rather jumps to the address indicated in the Indirect Vector Table. At power-on time, these vectors are set to the addresses of the normal ROM BASIC routines. However, it's possible to change these vectors so that when BASIC wants to perform one of these functions, it first goes to your routine. In this way, you can create new tokenized commands with their own error messages, and LIST or execute them. You can even change the function of normal BASIC commands.

Making New Keywords

The first step is to design a routine to tokenize your new keywords. Since BASIC 2.0 only uses keyword tokens from 128–204, you can use numbers 205–254 for fifty new commands (255 is used for PI). If you need more than that, you'll

have to go to a two-character token system, such as the one used by Simon's BASIC. The tokenization process is somewhat tricky, because you not only have to check the text input buffer starting at location 512 for your new keywords, but you must also be sure *not* to tokenize those words when they appear in a DATA statement, a REMark, or as a literal string in quotes. The method used in Program 1, "64 Keywords," closely parallels the normal BASIC tokenization routine. It first calls the regular tokenization routine, and then looks for new keywords. Since the normal keywords will be tokenized first, your new keywords cannot contain any of the old keywords. For example, the new tokenization routine will not recognize the keyword COLOR, because by the time it looks for it, the OR will have been changed to the single-character token for the BASIC keyword OR. Once the new tokenization routine is installed, lines containing these new tokens will not LIST correctly until a new detokenization routine is installed. The token conversion routine used in Program 1 is also based on the normal BASIC detokenization routine. It looks for token numbers 204 and up, and when it finds one, it expands that token to the ASCII equivalent. Otherwise, control is passed back to the old tokenization routine.

Once the new tokens are in place, the method for executing them is pretty straightforward. Statements such as PRINT are executed by the routine GONE, which is pointed to by the vector IGONE at location 776–777. That routine reads the next character, and determines whether it is a token. If it is, it looks up the execution address in a table, and passes control to it. Our new execution routine needs only to check if the character is a token numbered 204 or higher. If it is, its address is looked up in the table, and the routine is executed. If not, control passes back to the old routine. Functions, like INT and SGN, are evaluated by the routine EVAL, which is pointed to by the RAM vector IEVAL at address 778–779. New functions can be added by intercepting this routine, and checking for one of our new token characters. When such a token is found, the function is evaluated, and the result is placed in the Floating Point Accumulator. In all other cases, control is passed back to the old routine. Notice that in Program 1 the way in which numbers are passed to the new commands is modeled on the old BASIC commands. Therefore, if your new command needs two inputs, you can study a BASIC command

such as PEEK or POKE to see how it gets its inputs.

Though it's not necessary, you can add new error messages if you want. The easiest way to do this is to set locations 34–35 ($22–23) as a pointer to your new error message text, and enter the normal BASIC error handler routine at 42055 ($A447)

Program 1 gives a practical demonstration of how to implement these new commands. After you type in and RUN the program, the additional commands FILL, FCOL, LOOK, PAUSE and KILL will become available. Make sure you SAVE the program before you try to RUN it; a single error in the DATA statements could cause the computer to lock up. If that happens, you'll need to turn off your computer, and the program will be lost if it hasn't been SAVEd. A brief description of the syntax of these commands follows:

FILL *character,color*. This command fills the entire screen with 1000 repetitions of one character. The parameter *character* refers to the POKE value (0–255) of the character used to fill the screen, and *color* refers to the foreground color used for the fill character. If a character number from 256–65535 is used, a NOT A VALID CHARACTER error message will appear. If a color value from 16–256 is used, you will get a NOT A VALID COLOR error message.

FCOL *color*. This command is similar to FILL, but changes only the foreground color of text on the screen, and not the actual characters.

LOOK*(address)*. This is a new BASIC function. It returns the value of the two-byte word *address* and *address+1* (in BASIC, the equivalent formula is PEEK(address) + 256* PEEK(address)+1). As with PEEK, the format should be PRINT LOOK(address).

PAUSE *jiffies*. This command pauses execution of the program from 0 to 65535 jiffies (each jiffy is 1/60 second).

KILL. Finally, KILL disables all our new commands, and restores the old BASIC. The new commands can be restored with a SYS 12*4096 (49152) statement.

After you have run Program 1, type in Program 2. This program demonstrates the use of FILL, FCOL, and PAUSE. Remember, the computer will not understand and tokenize these new commands until after you have installed them with Program 1.

Using the Source Code

FILL and FCOL, though dramatic when used on the low-res text screen, are most helpful for changing the color map of the high-res screen. Like the other new commands presented here, they were selected more for their brevity than their inherent usefulness. The real purpose of Program 1 is to show how a machine language programmer can hook in new BASIC commands. The source code of this program, which follows, can be used as a framework for adding your own commands. To do this, you must:

1) Place the text of your new keywords in the table labeled KEYTXT. The last letter of each keyword should have its high bit set (in other words, use the ASCII value+128). Functions should all be put at the end of the table. Keep in mind that the text of these words should not include any of the old keywords. COLOR for example will not tokenize correctly, because it contains OR.

2) Place the address of the routines in the order in which their keywords appear in the KEYTXT table in the statement vector table STVEC and the function vector table FUNVEC. Note that the correct vector for a statement is its address *minus* one.

3) If you wish to add error messages, replace the text in the table starting with ERMSG0 with your own text. As with keywords, the last letter of each message should have the high bit set. You can also replace the labels CHRERR and COLERR with new labels, indicating the nature of your new error messages.

The BASIC Indirect Vector Table gives you the power to add new commands, or alter existing commands. This explanation can allow machine language programmers to upgrade the capabilities of Commodore 64 BASIC to match the rest of the machine.

Source Code for 64 Keywords

```
;ZERO PAGE EQUATES
;
ENDCHR = $Ø8 ;TEMP FLAG
COUNT  = $ØB ;TEMP FLAG
VALTYP = $ØD ;VARIABLE TYPE FLAG
GARBFL = $ØF ;TEMP FLAG
LINNUM = $14 ;UTILITY POINTER
INDEX  = $22 ;UTILITY POINTER
```

```
FORPNT = $49
JMPER  = $54 ;JMP TO FUNCTION
FACHO  = $62 ;FLOATING PT. ACC.
FBUFPT = $71 ;TEMP SAVE AREA
CHRGET = $73 ;BASIC READS TEXT
CHRGOT = $79 ;READ TEXT AGAIN
TXTPTR = $7A ;PNTR TO CURRENT TEXT
TIME   = $A0 ;SOFTWARE CLOCK (MSB)
LDTB1  = $D9 ;LINE LINK TABLE
;
;PAGE TWO EQUATES
;
BUF    = $200 ;TEXT INPUT BUFFER
HIBASE = $288 ;SCREEN MEMORY PAGE
VECSAV = $2A7 ;VECTOR SAVE AREA
;
;BASIC INDIRECT VECTORS
;
IERROR = $300 ;PRINT ERROR MESSAGE
IMAIN  = $302 ;MAIN 'READY.' LOOP
ICRNCH = $304 ;TOKENIZE KEYWORDS
IQPLOP = $306 ;PRINT KEYWORDS
IGONE  = $308 ;EXECUTE STATEMENTS
IEVAL  = $30A ;EVALUATE FUNCTIONS
;
;BASIC ROM ROUTINES
;
ERROR  = $A437 ;? ERROR MESSAGES
MAIN   = $A483 ;MAIN 'READY' LOOP
CRNCH  = $A57C ;TOKENIZE KEYWORD
PLOOP  = $A6F3 ;LIST NON-TOKEN
PRIT4  = $A6EF ;PRINT LAST CHAR
QPLOP  = $A71A ;PRINT KEYWORDS
NEWSTT = $A7AE ;NEXT STATEMENT
GONE   = $A7E4 ;EXECUTE A TOKEN
OUTDO  = $AB47 ;PRINT A CHAR
FRMNUM = $AD8A ;GET NEXT PARAMETER
CHKNUM = $AD8D ;CHECK VAR. TYPE
EVAL   = $AE86 ;FUNC. EVALUATION
PARCHK = $AEF1 ;GET VALUE IN ( )
GETNUM = $B7EB ;ADR IN 14,INT IN X
GETADR = $B7F7 ;CONVERT FP TO INT
FLOATC = $BC49 ;CONVERT INT TO FP
;
;PROGRAM VARIABLES & CONSTANTS
;
MAXCOL = $0F   ;MAXIMUM COLOR #
NEWTOK = $CC   ;1ST NEW TOKEN #
DATTOK = $49   ;'DATA' TOKEN-':'
```

```
REMTOK = $55    ;'REM' TOKEN-':'
;
* = $C000
;
;INSTALL NEW INDIRECT VECTORS
;A SYS TO 'INSTAL' ACTIVATES OUR
;NEW KEYWORD COMMANDS
;
INSTAL          ;INSTALL NEW VECTORS
LDX #$07       ;4 TWO-BYTE VECTORS
INSTL1
LDA ICRNCH,X
STA VECSAV,X ;SAVE OLD VECTORS
LDA IVECS,X   ;INSTALL NEW VECTORS
STA ICRNCH,X
DEX
BPL INSTL1    ;KEEP GOING TIL DONE
INSTL2
RTS
;
IVECS
.WORD TOKNIZ
.WORD PRTOK
.WORD EXEST
.WORD EXEFUN
;
KEYTXT ;TEXT OF KEYWORDS
.BYTE 'PAUS',$C5 ;PAUSE
.BYTE 'FCO',$CC ;FCOL
.BYTE 'FIL',$CC ;FILL
.BYTE 'KIL',$CC ;KILL
.BYTE 'LOO',$CB ;LOOK
.BYTE 0  ;END OF TABLE
;
STVEC ;STATMENT DISPATCH VECTORS
.WORD PAUSE-1 ;PAUSE
.WORD FCOL-1 ;FCOL
.WORD FILL-1 ;FILL
.WORD KILL-1 ;KILL
;
FUNVEC ;FUNCTION DISPATCH VECTORS
.WORD LOOK
;
FUNTOK  = FUNVEC-STVEC/2+NEWTOK
;
;PATCH TO TOKENIZATION ROUTINE
;ALLOWS US TO TOKENIZE OUR OWN
;KEYWORDS USING THE UNUSED TOKEN
;NUMBERS 204-254
```

```
;
TOKNIZ
JSR CRNCH   ;TOKENIZE AS USUAL
CRUNCH       ;DO 2ND TOKENIZATION
LDX #$00    ;SET READ INDEX
LDY #$04    ;SET WRITE INDEX
STY GARBFL ;CLEAR 'DATA' FLAG
CRN1
LDA BUF,X ;GET NEXT CHARACTER
;BMI MOVE   ;WRITE IF A TOKEN
CRN2
STA ENDCHR ;FOR END QUOTE TEST
CMP #$22    ;IS THIS A QUOTE?
BEQ SKQUOT ;YES, SKIP TO NEXT "
BIT GARBFL ;IF IN 'DATA STATEMENT
BVS MOVE ;WRITE THE CHARACTER
CMP #'A'    ;< THE LETTER 'A?
BCC MOVE    ;YES, WRITE IT
CMP #$5B    ;> THE LETTER 'Z
BCS MOVE    ;YES, PASS IT THROUGH
STY FBUFPT ;SAVE WRITE INDEX
LDY #NEWTOK-$80 ;# OF 1ST TOKEN
STY COUNT   ;SET TOKEN COUNTER
LDY #$FF
STX TXTPTR ;SAVE READ INDEX
DEX         ;TO OFF?SET THE INX
CRN3
INY         ;ADVANCE WRITE INDEX
INX         ;ADVANCE READ INDEX
CRN4
LDA BUF,X   ;GET BUFFERED CHAR
SEC
SBC KEYTXT,Y ;= NEXT TABLE CHAR?
BEQ CRN3 ;YES, KEEP GOIN'
CMP #$80 ;LAST KEYWORD CHAR?
BNE NEXTKW ;NOPE, TRY NEXT WORD
ORA COUNT ;YEP, GET TOKEN NO.
CRN5
LDY FBUFPT ;RESTORE WRITE INDEX
MOVE
INX ;ADVANCE READ INDEX
INY ;ADVANCE WRITE INDEX
STA BUF-5,Y ;WRITE CHARACTER
LDA BUF-5,Y ; TO TEST FOR EOL
BEQ EXIT ;A ZERO ENDS THE LINE
SEC
SBC #':' ; :STATEMENT TERMINATOR?
BEQ MOVE1 ;YEP, CLEAR 'DATA FLAG
CMP #DATTOK ;TOKEN FOR 'DATA?
```

```
BNE MOVE2 ; DON'T CLEAR FLAG
MOVE1
STA GARBFL ;CLEAR 'DATA FLAG
MOVE2
SEC
SBC #REMTOK    ;TOKEN FOR 'REM?
BNE CRN1    ;NO, NEXT CHARACTER
STA ENDCHR ;YES, FALL THRU
SKIP1
LDA BUF,X  ;GET NEXT CHARACTER
BEQ MOVE   ;KEEP GOIN' TIL EOL
CMP ENDCHR ;OR TERMINATOR
BEQ MOVE
SKQUOT ;SKIP TEXT IN " "
INY ;ADVANCE WRITE INDEX
STA BUF-5,Y ;WRITE CHAR
INX ;ADVANCE READ INDEX
BNE SKIP1 ;ALWAYS--KEEP GOIN'
NEXTKW ;TRY NEXT KEYWORD
LDX TXTPTR ;RESTORE READ INDEX
INC COUNT ;ADVANCE KEYWORD CNTR
NEXT1
INY ;ADVANCE TABLE INDEX
LDA KEYTXT-1,Y ;GET TABLE CHAR
BPL NEXT1 ;SKIP 'TIL NEXT WORD
LDA KEYTXT,Y ;GET 1ST CHAR
BNE CRN4 ;TRY AGAIN
LDA BUF,X ;END OF TABLE
BPL CRN5 ;ALWAYS
EXIT
STA BUF-3,Y ;SET END OF LINE
LDA #$FF ;RESTORE TXTPTR
STA TXTPTR ; TO START OF BUF
RTS
;
;THIS PATCH TO THE 'LIST' ROUTINE
;ALLOWS US TO EXPAND OUR TOKENS
;BACK TO ASCII TEXT, SO THAT THEY
;LIST OUT CORRECTLY
;
PRTOK ;PRINT OUR NEW TOKENS
BPL PRINT1 ;<128, NOT A TOKEN
CMP £$FF ;IS IT PI?
BEQ PRINT1 ;YES, PRINT IT
BIT GARBFL ;ARE WE IN QUOTES?
BMI PRINT1 ;YES, PRINT ANYTHING
CMP #NEWTOK ;IS IT A NEW TOKEN?
BCC OLDPR ;NO, USE OLD ROUTINE
SEC
```

```
SBC #NEWTOK-1 ;GET TOKEN NUMBER
TAX  ;TO USE AS INDEX
STY FORPNT ;SAVE STATEMENT INDEX
LDY #$FF
PRTOK1
DEX ;NEXT KEYWORD
BEQ PRLOOP ;THIS IS THE ONE
PRTOK1
DEX ;NEXT KEYWORD
BEQ PRLOOP ;THIS IS THE ONE
PRTOK2
INY           ;GET NEXT LETTER..
LDA KEYTXT,Y ;IN KEYWORD
BPL PRTOK2    ;END OF KEYWORD?
BMI PRTOK1    ;NO, NEXT LETTER
;
PRLOOP
INY           ;GET NEXT LETTER...
LDA KEYTXT,Y ;IN KEYWORD
BMI PRINT2    ;END OF KEYWORD?
JSR OUTDO     ;NO, PRINT CHAR...
BNE PRLOOP    ;AND REPEAT
;
PRINT1
JMP PLOOP  ;PRINT ONE CHARACTER
PRINT2
JMP PRIT4  ;PRINT LAST CHARACTER
OLDPR
JMP QPLOP  ;USE OLD ROUTINE
;
;THIS PATCH TO THE STATEMENT
;EXECUTION ROUTINE ALLOWS US TO
;CHECK FOR OUR NEW STATEMENT
;TOKENS, AND TO EXECUTE THEM.
;
EXEST
JSR CHRGET  ;GET NEXT CHARACTER
CMP #NEWTOK ;IS IT A NEW TOKEN?
BCC OLDEXE  ;NO, USE OLD ROUTINE
JSR EXE1    ;EXECUTE STATEMENT
JMP NEWSTT  ;AND START OVER
;
EXE1 ;EXECUTE OUR NEW TOKEN
;(CARRY IS ALREADY SET)
SBC #NEWTOK   ;GET TOKEN #
ASL A         ;2*TOKEN #...
TAY           ;IS OUR INDEX TO..
LDA STVEC+1,Y ;THE VECTOR TABLE
PHA           ;PUSH ADDRESS-1...
```

```
LDA STVEC,Y    ;ONTO STACK...
PHA            ;FOR RTS...
JMP CHRGET     ;AT END OF CHRGET
;
OLDEXE
JSR CHRGOT ;GET CHARACTER AGAIN
JMP GONE+3 ;AND USE OLD ROUTINE
;
;THIS PATCH TO THE EVALUATION
;ROUTINE ALLOWS US TO CHECK FOR
;OUR NEW FUNCTION KEYWORDS, AND
;TO EVALUATE THEM, LEAVING THE
;RESULT IN THE FLOATING POINT
;ACCUMULATOR
;
EXEFUN
LDA #$ØØ
STA VALTYP ;SET TO NON-STRING
JSR CHRGET ;GET EVAL. CHAR.
CMP #$FF ;IS IT PI?
BEQ OLDFUN ;YES, DO OLD EVAL.
CMP #FUNTOK ;IS IT A NEW FN?
BCC OLDFUN ;NO, DO OLD EVAL.
;GET TOKEN #
SEC
SBC #FUNTOK
ASL A ;USE AS INDEX
PHA ;SAVE ON STACK
JSR CHRGET ;GET EVAL. CHAR.
JSR PARCHK ;GET EXPRESSION IN ()
PLA ;GET INDEX BACK
TAY
LDA FUNVEC,Y
STA JMPER+1
LDA FUNVEC+1
STA JMPER+2 ;FORM POINTER
JSR JMPER ;EVALUATE FN
JMP CHKNUM ;CHECK VAR. TYPE & RTS
;
OLDFUN
JSR CHRGOT
JMP EVAL+7 ;OLD ROUTINE
;
;THIS SECTION CONTAINS MY NEW
;COMMANDS.  THIS IS WHERE YOU WILL
;INSTALL YOUR OWN CODE.
;
;LOOK (X) FUNCTION PEEKS 2 BYTES
;
```

```
LOOK ;
LDA LINNUM+1
PHA
LDA LINNUM
PHA  ;SAVE LINNUM ON STACK
JSR GETADR ;INTEGER IN 14/15=ARG
LDY #$00 ;SET INDEX
LDA (LINNUM),Y ;GET LOW BYTE
STA FACHO+1
INY
LDA (LINNUM),Y ;GET HIGH BYTE
STA FACHO
PLA
STA LINNUM
PLA
STA LINNUM+1 ;RESTORE LINNUM
LDX #$90 ;SET EXPONENT
SEC
JSR FLOATC ;CONVERT INT TO FP
RTS
;
;'KILL' DISABLES THE NEW COMMANDS
;
KILL
LDX #$07 ;NUMBER OF VECTORS
KILL1
LDA VECSAV,X ;GET SAVED VECTORS
STA ICRNCH,X ;RESTORE THEM
DEX
BPL KILL1    ;DONE?
RTS
;
;'FILL'--FILL X,Y FILLS THE SCREEN
;WITH CHARACTER X IN COLOR Y
;
FILL
JSR GETNUM ;GET ADDR,INT IN X
LDA LINNUM+1 ;CHAR >255?
BNE CHRERR ;YES, ERROR
STA FACHO ;CLEAR POINTER
CPX #MAXCOL+1   ;COL > 15?
BCS COLERR ;YES, ERROR
TXA ;SAVE COLOR
PHA
LDA LDTB1+23 ;FORM POINTER..
AND #03      ;TO TOP...
ORA HIBASE   ;OF SCREEN
STA FACHO+1
LDA LINNUM ;GET FILL CHAR
```

```
JSR FILL1
PLA  ;GET COLOR
;
FCOL1 ;FILL COLOR RAM
;
LDX #$DB   ;POINTER TO..
STX FACHO+1 ;SCREEN RAM
;
;FILL LOOP
;
FILL1
LDX #$03        ;DO 3 PAGES
LDY #$E7        ;AND MOST OF 4TH
FILL2
STA (FACHO),Y   ;FILL 'ER UP
DEY             ;NEXT BYTE
BNE FILL2
STA (FACHO),Y   ;DON'T FORGET ZERO
DEC FACHO+1     ;NEXT PAGE
DEX
BPL FILL2       ;DONE YET?
RTS
;
;OUR NEW ERROR MESSAGE ROUTINE
;STARTS HERE
;
CHRERR
LDA #$00 ;CHARACTER ERROR NO.
.BYT $2C ;SKIP NEXT INSTRUCTION
COLERR
LDA #$01 ;COLOR ERROR NO.
ASL A    ;ERROR NO. * 2
TAX      ;IS USED AS AN INDEX
LDA ERRVEC,X ;TO VECTOR TABLE
STA INDEX ;SET UP TEXT POINTER
LDA ERRVEC+1,X
STA INDEX+1
JMP ERROR+16 ;PRINT ERROR MSG
;
ERRVEC
.WORD ERMSG0
.WORD ERMSG1
;
ERMSG0
.BYT 'NOT A VALID CHARACTE',$D2
ERMSG1
.BYT 'NOT A VALID COLO',$D2
;
;'FCOL'--FCOL X FILLS COLOR RAM
```

```
;WITH COLOR X
;
FCOL
JSR FRMNUM ;GET COLOR #
JSR GETADR ;CONVERT FP TO INT
CMP #ØØ ;COLOR>255?
BNE COLERR ;YES, ERROR
STA FACHO
CPY #MAXCOL+1    ;COLOR>15?
BCS COLERR ;YES, ERROR
TYA ;COLOR TO .A
JMP FCOL1
;
;'PAUSE'--PAUSE X PAUSES PROGRAM
;EXECUTION FOR X JIFFIES (1/6Ø OF
;A SECOND)
;
PAUSE
JSR FRMNUM ;GET # OF JIFFIES
JSR GETADR ;CONVERT FP TO INT
TAX ;HIGH BYTE IN .X, LOW IN .Y
PAUSE1
CPY #$ØØ ;LOW BYTE DONE?
BEQ PAUSE4 ;YES, TRY HIGH BYTE
PAUSE2
DEY
LDA TIME+2 ;SOFTWARE CLOCK...
PAUSE3
CMP TIME+2 ;ON THE SAME JIFFY?
BEQ PAUSE3 ;YES, TRY AGAIN
BNE PAUSE1 ;NO, ONE JIFFY DOWN
PAUSE4
CPX #$ØØ  ;HIGH BYTE DONE?
BEQ PAUSE5 ;YES, EXIT
DEX   ;NO, COUNT DOWN HIGH BYTE
JMP PAUSE2 ;AND DO NEXT LOW BYTE
PAUSE5
RTS
;
.END
```

Program 1. 64 Keywords

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

```
1Ø B=Ø:FOR I=49152 TO 49685:READA:POKEI,A:B=B+A:NE
   XT I                                    :rem 78
2Ø IF B<>64356 THEN PRINT"CHECKSUM ERROR--CHECK YO
   UR TYPING":END                          :rem 133
```

```
30 SYS49152:PRINT"NEW BASIC COMMANDS INSTALLED"
                                                :rem 176
49152 DATA 162, 7, 189, 4, 3, 157               :rem 113
49158 DATA 167, 2, 189, 18, 192, 157             :rem 21
49164 DATA 4, 3, 202, 16, 241, 96               :rem 102
49170 DATA 58, 192, 190, 192, 243, 192          :rem 111
49176 DATA 21, 193, 80, 65, 85, 83              :rem 172
49182 DATA 197, 70, 67, 79, 204, 70             :rem 224
49188 DATA 73, 76, 204, 75, 73, 76              :rem 180
49194 DATA 204, 76, 79, 79, 203, 0              :rem 169
49200 DATA 248, 193, 228, 193, 115, 193         :rem 157
49206 DATA 103, 193, 71, 193, 32, 124            :rem 45
49212 DATA 165, 162, 0, 160, 4, 132             :rem 190
49218 DATA 15, 189, 0, 2, 133, 8                 :rem 55
49224 DATA 201, 34, 240, 79, 36, 15             :rem 202
49230 DATA 112, 38, 201, 65, 144, 34             :rem 245
49236 DATA 201, 91, 176, 30, 132, 113             :rem 39
49242 DATA 160, 76, 132, 11, 160, 255             :rem 42
49248 DATA 134, 122, 202, 200, 232, 189          :rem 141
49254 DATA 0, 2, 56, 249, 26, 192                :rem 110
49260 DATA 240, 245, 201, 128, 208, 48            :rem 96
49266 DATA 5, 11, 164, 113, 232, 200             :rem 241
49272 DATA 153, 251, 1, 185, 251, 1              :rem 199
49278 DATA 240, 54, 56, 233, 58, 240                :rem 9
49284 DATA 4, 201, 73, 208, 2, 133               :rem 149
49290 DATA 15, 56, 233, 85, 208, 179              :rem 15
49296 DATA 133, 8, 189, 0, 2, 240                :rem 109
49302 DATA 223, 197, 8, 240, 219, 200             :rem 44
49308 DATA 153, 251, 1, 232, 208, 240             :rem 39
49314 DATA 166, 122, 230, 11, 200, 185            :rem 84
49320 DATA 25, 192, 16, 250, 185, 26             :rem 255
49326 DATA 192, 208, 180, 189, 0, 2              :rem 209
49332 DATA 16, 190, 153, 253, 1, 169                :rem 0
49338 DATA 255, 133, 122, 96, 16, 42                :rem 5
49344 DATA 201, 255, 240, 38, 36, 15             :rem 253
49350 DATA 48, 34, 201, 204, 144, 36             :rem 249
49356 DATA 56, 233, 203, 170, 132, 73             :rem 49
49362 DATA 160, 255, 202, 240, 8, 200             :rem 37
49368 DATA 185, 26, 192, 16, 250, 48              :rem 16
49374 DATA 245, 200, 185, 26, 192, 48             :rem 60
49380 DATA 8, 32, 71, 171, 208, 245              :rem 209
49386 DATA 76, 243, 166, 76, 239, 166              :rem 79
49392 DATA 76, 26, 167, 32, 115, 0               :rem 160
49398 DATA 201, 204, 144, 21, 32, 0              :rem 193
49404 DATA 193, 76, 174, 167, 233, 204           :rem 109
49410 DATA 10, 168, 185, 49, 192, 72                :rem 8
49416 DATA 185, 48, 192, 72, 76, 115              :rem 17
49422 DATA 0, 32, 121, 0, 76, 231                 :rem 87
49428 DATA 167, 169, 0, 133, 13, 32              :rem 207
```

```
49434 DATA 115, 0, 201, 255, 240, 33      :rem 240
49440 DATA 201, 208, 144, 29, 56, 233      :rem 47
49446 DATA 208, 10, 72, 32, 115, 0        :rem 145
49452 DATA 32, 241, 174, 104, 168, 185    :rem 104
49458 DATA 56, 192, 133, 85, 173, 57       :rem 22
49464 DATA 192, 133, 86, 32, 84, 0        :rem 163
49470 DATA 76, 141, 173, 32, 121, 0       :rem 197
49476 DATA 76, 141, 174, 165, 21, 72       :rem 11
49482 DATA 165, 20, 72, 32, 247, 183        :rem 6
49488 DATA 160, 0, 177, 20, 133, 99       :rem 216
49494 DATA 200, 177, 20, 133, 98, 104      :rem 52
49500 DATA 133, 20, 104, 133, 21, 162      :rem 25
49506 DATA 144, 56, 32, 73, 188, 96       :rem 225
49512 DATA 162, 7, 189, 167, 2, 157       :rem 218
49518 DATA 4, 3, 202, 16, 247, 96         :rem 111
49524 DATA 32, 235, 183, 165, 21, 208      :rem 50
49530 DATA 44, 133, 98, 224, 16, 176        :rem 8
49536 DATA 41, 138, 72, 165, 240, 41        :rem 2
49542 DATA 3, 13, 136, 2, 133, 99         :rem 106
49548 DATA 165, 20, 32, 150, 193, 104      :rem 47
49554 DATA 162, 219, 134, 99, 162, 3       :rem 12
49560 DATA 160, 231, 145, 98, 136, 208    :rem 106
49566 DATA 251, 145, 98, 198, 99, 202      :rem 79
49572 DATA 16, 244, 96, 169, 0, 44        :rem 169
49578 DATA 169, 1, 10, 170, 189, 187       :rem 19
49584 DATA 193, 133, 34, 189, 188, 193    :rem 127
49590 DATA 133, 35, 76, 71, 164, 191       :rem 11
49596 DATA 193, 212, 193, 78, 79, 84       :rem 33
49602 DATA 32, 65, 32, 86, 65, 76         :rem 118
49608 DATA 73, 68, 32, 67, 72, 65         :rem 127
49614 DATA 82, 65, 67, 84, 69, 210        :rem 174
49620 DATA 78, 79, 84, 32, 65, 32         :rem 123
49626 DATA 86, 65, 76, 73, 68, 32         :rem 132
49632 DATA 67, 79, 76, 79, 210, 32        :rem 176
49638 DATA 138, 173, 32, 247, 183, 201    :rem 108
49644 DATA 0, 208, 187, 133, 98, 192       :rem 15
49650 DATA 16, 176, 181, 152, 76, 146      :rem 61
49656 DATA 193, 32, 138, 173, 32, 247      :rem 63
49662 DATA 183, 170, 192, 0, 240, 9       :rem 208
49668 DATA 136, 165, 162, 197, 162, 240   :rem 166
49674 DATA 252, 208, 243, 224, 0, 240      :rem 46
49680 DATA 4, 202, 76, 4, 194, 96         :rem 119
```

Program 2. FILL, FCOL, and PAUSE

```
10 POKE 53281,0                          :rem 239
20 FOR I=1 TO 26                          :rem 11
30 FILL I,1:PAUSE20                       :rem 74
40 FCOL 15:PAUSE 20                        :rem 8
50 FCOL 12:PAUSE 20                        :rem 6
60 FCOL 11:PAUSE 20                        :rem 6
70 FCOL 0:PAUSE 20                       :rem 213
80 NEXT:PAUSE 20:PRINTCHR$(147)          :rem 118
```

6

Utilities

SuperBASIC 64

Martin C. Kees

How would you like to be able to access 41 valuable new commands when you're programming in BASIC? "SuperBASIC 64" adds sprite, color, graphics, sound, and memory management features and enhances eight BASIC commands. It's been made even more powerful than the version which originally ran in COMPUTE! *And it's designed to work as easily and as quickly as any ordinary BASIC instruction. Typing it into your computer is foolproof, for you'll use the Machine Language Editor (MLX) in Appendix I. Once you try SuperBASIC, you'll wonder how you programmed without it—it's an especially valuable addition to any owner's library of programs. As a bonus, the author has included additional sample programs which use SuperBASIC, including "SuperBASIC Sprite Editor," found in another section of this book.*

"SuperBASIC 64"adds commands to BASIC using a special technique. BASIC is automatically copied to its matching RAM and modified to change the STOP command to a wedge vector (similar to Apple's ampersand (&) wedge). The character chosen was the left bracket ([). Then, using four-letter mnemonics following the wedge character, you can select which SuperBASIC command you want to execute.

These machine language routines make it very easy to control virtually all the VIC-II chip special features. Sprites and hi-res graphics can be controlled from BASIC without having to POKE or use Boolean functions to enable special graphics modes. Since BASIC was moved to RAM to implement the [wedge, this made it convenient to enhance a few BASIC commands. I added the use of variable expressions for GOTO and GOSUB, and RESTORE by line number. These

changes to BASIC in RAM don't slow execution as they would have if CHRGET wedging techniques had been used.

SuperBASIC Command Format

The commands can be used in either direct or program mode. The general format is [xxxx <exp>,<exp>... where xxxx represents the four-character mnemonic and <exp> is a number, variable, or a valid BASIC expression. Specific syntax for each command is listed in Table 1, "SuperBASIC Command Summary." When a color is selected, use the standard value ordinarily POKEd to the VIC chip. I've used the same coordinate system for sprite positions as given in Commodore documentation. The hi-res upper left corner is 0,0 and the lower right is 319,199. Commands that switch a function on or off use 0 for off and 1 for on.

SuperBASIC includes two types of changes to normal BASIC, enhanced commands and new commands. Enhanced commands include GOTO and GOSUB and variants with IF and ON. You can use a line number expression for these commands. This can help in program readability, allowing instructions such as GOTO KEY when KEY=1000. This would transfer control to line 1000. RESTORE can also be followed by a line number expression. RESTORE KEY would cause the next READ to use the first DATA statements encountered at or after line 1000. This allows DATA statements to be selected under program control. Small files could be maintained in DATA statements and accessed by line number. When LISTing a program, the SHIFT key pauses the list until released. The ASC function will return a value of zero for null strings.

The new commands can be divided into five categories: sprite, sound, color control, VIC memory mapping, and graphics control. A convenience command [CATA is also included. This lists to the screen all mnemonics defined in SuperBASIC. (I use it to test if SuperBASIC is enabled.)

Loading the Program

To type in SuperBASIC 64 (Program 1) you *must* use the "MLX Machine Language Entry Program," found in Appendix I. Be sure you read the explanation in Appendix I and understand how to use MLX before attempting to enter SuperBASIC.

The numbers you type in create a low memory loader for

SuperBASIC which can be loaded and run as if it were a BASIC program. Because the data for the SuperBASIC loader must go into the same area of memory where BASIC normally resides, a special tactic must be used to prevent the SuperBASIC data from overwriting MLX as it is entered. First, turn the computer off and back on to reset memory pointers to their normal values. Next type in the following line in direct mode (without a line number) and hit RETURN:

```
POKE 44,22: POKE 642,22: POKE 5632,0: NEW
```

This moves up the start of the memory area used by BASIC so that all of the data for SuperBASIC will fit below MLX without interference. Now LOAD and RUN the MLX program in the normal manner. When MLX asks for the starting and ending addresses for SuperBASIC, give 2049 as the start and 5330 for the end. When you finish typing in the data for SuperBASIC, use the MLX Save command to store a copy of the SuperBASIC loader on disk or tape. Be sure to give MLX a unique name for the SuperBASIC program as it won't replace a file. If you do not type in all the data for SuperBASIC in one session, you must repeat the procedure for moving up the start of BASIC before loading MLX to complete your entry.

When you have a complete copy of the loader, you must reset memory to its normal conditions before loading and running SuperBASIC. You can do this by turning the computer off and back on, or with the command SYS 64738. When you run the SuperBASIC loader, it first copies BASIC from ROM into the underlying RAM and makes modifications to certain commands. Then it copies the machine language for the rest of the SuperBASIC routines into memory at $C000–$CC00. No other machine language subroutines which use memory starting at $C000 can be used with SuperBASIC 64, but the DOS Wedge program can be used without conflict. Some of the graphics commands use memory at $02B0–$02C0 for data storage. The loader erases itself from the BASIC memory area after it is run.

The SuperBASIC commands will be enabled until you hit RUN/STOP—RESTORE or POKE 1,55. Once loaded, SuperBASIC can be reenabled with POKE 1,54. The programs you write with SuperBASIC commands are loaded and saved in the normal manner. The only conflict with normal BASIC is

the use of the STOP command. It is not available; use END instead. When SuperBASIC commands are listed while SuperBASIC is disabled, the [character will print as STOP. If the [character is printed and SuperBASIC is disabled, it indicates that the line was entered while SuperBASIC was not in force. That command will appear correct but will produce a syntax error on execution.

Sprite Commands

[DSPR [MOVE [KSPR [ESPR [BSPP. These commands are used in defining sprite characteristics and controlling sprite movement. [DSPR (Define Sprite) is a general setup command that initializes a sprite for the VIC-II chip. The ten arguments in the parameter list (see Table 1) specify most options available for sprite control. [DSPR enables the selected sprite numbered 0–7, stores block address (blk) in current screen pointer table, expands if xexp or yexp=1, determines initial display position (xpos,ypos), and sets sprite color registers (sprcolor). Multicolored sprites are selected by setting multi=1, single color by multi=0. Mc0 and Mc1 are optional arguments in the list which set up multicolor 0 and 1. [MOVE moves the selected sprite to xpos,ypos. Horizontal values greater than 255 are handled automatically. [KSPR and [ESPR kill or enable the selected sprite respectively. [BSPP sets the background/sprite priority for the selected sprite (sel=1 sets background in front of sprite).

Sound Commands

[SSND [PLAY. These commands access some of the features of the SID chip. [SSND (set up sound) produces a sound from one of the three voices of the SID chip. Voice (1–3) selects the voice, ad and sr control the attack/decay and sustain/release registers of the selected voice. Wave controls the waveform, gating, and special effects functions of the sound chip. Wave, ad, and sr use the same values that would normally be POKEd to these registers. Freq controls the frequency of the voice but is a 16-bit value in the range 0–65535. Pwidth is the pulsewidth value for the pulse waveform and is needed only when wave=65. Pwidth is a 12-bit value in the range 0–4095. [SSND sets the volume register to 15. [PLAY is a short form of [SSND that assumes AD/SR values have been set previously. Waveform and voice values are coded into the first parameter argument by wave*256+voice. Freq and Pwidth are

used the same as in [SSND. [PLAY can be used to silence a voice; for instance, [PLAY 1,0 would silence voice one.

VIC Color Control

[BKGD [BKG4 [EXTC [FCOL. These commands control background, border, and text character color. [BKGD sets the background to the selected color. [BKG4 sets all four background color registers (used in extended color and multicolor bitmap modes). [EXTC sets the exterior border to the selected color. [FCOL (fill color memory) fills the color memory block with the selected color. This causes all text on the current screen to be displayed in the selected color. [FCOL is also useful in multicolor bitmap mode to set multicolor pixel color.

VIC Memory Mapping

[BANK [VS1K [CB2K. The VIC-II chip views memory differently than does the 6510 chip. VIC-II sees only 16K at a time and maps the ROM character set into part of this 16K bank at times. These commands allow changes to the normal locations of the screen and character sets. [BANK selects which one of the four banks (0–3) the VIC-II chip sees. Normally this is bank 0. [BANK resets the pointer BASIC uses to locate the screen. [VS1K (Video Screen 1K block) determines which 1K block of the 16 available is used for the text screen.

The blocks are numbered 0–15. The BASIC screen pointer is reset for this location. [CB2K (Character Base 2K block) controls which 2K block of the eight available is used for the character set. The blocks are numbered 0–7. In banks 0 and 2 the ROM set is located at 2K blocks 2 and 3. [CB2K is also used to select which 8K block is used for the bitmap screen. Values 0–3 select the lower 8K block, values 4–7 select the upper 8K block.

These three commands must be used in coordination to smoothly relocate the screen. Caution must be exercised in selecting locations since a system crash will result if the screen overwrites important RAM such as page zero. Banks 2 and 3 must be used with great care. More on bank 3 usage later.

Program 7 demonstrates relocation to PET standard locations for the screen and BASIC.

Graphics/Text Control

[ECGR [MCGR [BMGR. These commands select extended color, multicolor, or bitmap graphics mode. A value of

0 turns the mode off and a value of 1 turns the mode on. Only multicolor and bitmap work in conjunction with each other to form a combined mode. When extended color and bitmap are both on, the screen will appear blank. This effect might be useful for temporarily hiding the screen.

[MXGR [KMXG [CMXV. These commands set up a simple interrupt routine that allows mixed modes to appear in two sections of the screen. [MXGR will change the contents of one VIC register (reg) or part of the contents (the bits OFF in mask) each time the raster counter equals one of the two raster select values (rast1 and rast2). The values in val1 or val2 will be stored into the selected VIC register. You must determine the appropriate value for the particular register. For example, [MXGR 33,240,152,6,252,1 will cause screen lines 49 to 151 to be displayed with background white (color=1) and lines 152 to 250 with background blue.

The visible portion of the screen extends from raster 49 to raster 250 (Commodore documentation says 51–251). [KMXG will kill the interrupt and leave the selected register in an unknown state. [CMXV (Change Mixed-mode Values) allows changing val1 and val2 while mixed mode is in force. By setting them equal, a known state will be in effect after [KMXG. You should not attempt tape or disk I/O while [MXGR is in force. [MXGR mode shouldn't be used in bank 3 VIC mapping if hi-res graphics commands are to be used.

It's possible to set up a text window at the bottom of a hi-res screen using [MXGR. The difficulty is that [MXGR only can change one VIC register. Thus the character base pointer can't be changed as well as the bitmap select bit by [MXGR. This can be solved by locating the character set within the hi-res screen, and putting a text window over the top of the character set.

A six line text window at the bottom of the hi-res screen can easily be created using this technique. In bank 0, using the upper 8K hi-res block, you would first use [CB2K 7 to select both the upper 8K hi-res block and the seventh 2K character set block. Then clear the hi-res screen with [FBMS 0 and download a character set to starting location 7*2048. [MXGR 17,223,0,32,201,0 will complete the setup.

[SIZE [XYSC. These commands help use the smooth scroll registers of the VIC-II chip. [SIZE selects 40 or 38 columns for the text display chosen by setting colsel to 1 or 0

(colsel=1 selects 40 columns) and sets the number of lines to 25 or 24 (rowsel=1 selects 25 lines). [XYSC moves the entire text screen horizontally or vertically up to seven pixels. By setting xpos and ypos to a value in the range 0–7, the screen can be stepped up a pixel at the time to produce a smooth scroll. When used in conjunction with a machine language scroll routine or the automatic scroll up, text can be scrolled smoothly across or up the entire screen.

[DLCS. [DLCS (Download Character Set) assists in using banks without ROM character set images and in designing custom character sets. You can copy the uppercase graphics set, upper- and lowercase set, or both by setting the set equal to 0, 1, or 2 respectively. This is followed by the address of the first location in memory where you wish the ROM set copy to be positioned. This should be on a 2K boundary unless you wish to change the order of the set. When the address is 53248, the set will be copied into the RAM beneath the ROM set for use in bank 3.

[FBMS [FSCR. The current hi-res screen (determined by the last [CB2K command) can be filled with any byte value with [FBMS (Fill Bitmap Screen). [FSCR works in a similar way with the current text screen. The entire screen is filled with a byte value. Since the text screen is used for color control in hi-res mode, [FSCR can be used for hi-res color control.

[PLOT [FLIP [CLPX [MCPL. These commands are used in plotting pixel points in hi-res graphics modes. The first three plot in 320 x 200 resolution two-color mode, the last in 160 x 200 resolution four-color mode. [PLOT sets the selected pixel on, [CLPX turns the pixel off, and [FLIP changes the pixel to the opposite state. [MCPL (Multicolor Plot) accepts horizontal coordinates in the range 0–159 and plots in one of four colors determined by sel with sel in the range 0–3. A value of 0 selects background color, 1 selects text screen low-nybble color, 2 selects text screen high-nybble color, and 3 selects color memory color. Before you execute any of the plotting commands, [CB2K must be used to select the appropriate 8K block and [BMGR 1 must be in force for the plot to be seen. Remember that y coordinates increase as you go down the screen.

[DRAW [UNDR [FLLN [DRW2 [SETP. These commands are used to draw and erase lines to and from the hi-res screen. [DRAW, [UNDR, and [FLLN require a parameter list containing

the start and end points of a line segment. The line is drawn from x1,y1 toward x2,y2. These commands plot the line in three modes. [DRAW turns on the pixels of the line, [UNDR turns off the pixels of the line, and [FLLN flips the state of the pixels of the line. The three commands also set the mode of drawing for the [DRW2 command and save the last plotted point position in a pen position register. [DRW2 (Draw to) only uses an end point in the parameter list. The starting point is obtained from the pen position register contents set by a previous [DRAW, [FLLN, [UNDR, or [SETP command. [DRW2 will plot, flip, or erase the line depending on which line drawing mode was last used. [SETP stores the given x1,y1 coordinate in the pen position register.

[HRCS [CHAR [CHRX [CODE. These commands make it easy to put text on the hi-res screen. [HRCS (Hi-Res Character Set) stores the address for the character set to be used. It need not be located on a 2K boundary or even be the same set as used on the text screen. The address supplied is of the first byte of the set. A value of 53248 will select the ROM set (upper/graphics). [CHAR and [CHRX plot an 8 x 8 character to a selected position on the current hi-res screen.

The character code (char) to select which character to plot corresponds to the screen POKE codes listed in Appendix G. Example: [CHAR 1,100,100 would plot the letter A with position 100,100 being the upper left corner of the 8 x 8 character cell. [CHAR plots the cell to the hi-res screen absolutely while [CHRX uses the exclusive OR function to flip the cell pixels. So [CHRX can be used to unplot a previously plotted character. [CODE helps in translating to the screen POKE code used by [CHAR and [CHRX in character selection.

The argument for [CODE must be the name of a defined string variable. Upon execution, the ASCII values stored in the string will be converted to screen POKE codes. The RVS ON and RVS OFF control characters can be used within the string to select the upper 128 or lower 128 characters of the set. All other control characters will produce unpredictable results. Once the string is converted using [CODE, use the ASC function and MID$ function to read the codes. The ASC function will give correct results for the 0 character of the set. Be careful when using strings not built to high memory because [CODE will modify the actual string data stored within the BASIC text area.

[HRAM [LOOK [STUF. These commands make use of [BANK 3 possible from BASIC. When bank 3 is selected, the VIC-II chip uses RAM from $C000 to $FFFF and ignores ROM located at the same addresses, including the ROM character set. SuperBASIC allows the location of one text screen ([VS1K 3 located at $CC00) in bank 3. RAM from $D000 to $FFFF can be used for character sets, sprites, and a hi-res screen.

The main problem confronting the bank 3 user is the switching required to read and write to these RAM locations. All plotting commands need to read, as well as write, to RAM. These commands can be preceded by [HRAM to accomplish this in bank 3. No embedded banks are allowed following [HRAM and the selected mnemonic. For example, [HRAMDRAW 1,0,100,100 would be used to draw a line to the hi-res screen at $E000 under the Kernal ROMs. [HRAM should be used in this manner with [PLOT, [FLIP, [CLPX, [MCPL, [DRAW, [UNDR, [FLLN, [DRW2, [CHAR, and [CHRX in bank 3. Using the first 3K of bank 3 will crash SuperBASIC, so make sure the text screen is relocated by [VS1K 3. When the transition to bank 3 is made, the 1K block at $0400 can be reclaimed for BASIC program storage. [LOOK and [STUF are PEEK and POKE equivalents that can be used with [HRAM to examine and change RAM. [LOOK is different from PEEK in that a defined variable name is used in the parameter list to return the value read from memory. [STUF works the same as POKE and is primarily useful for storing to block $D000 RAM (for example, [HRAMSTUF 53248,255 writes to RAM under the VIC chip).

Using the Commands

Errors in SuperBASIC commands will give the syntax error message. One difficult error to detect occurs when an embedded BASIC keyword is constructed by part of the command mnemonic and the following parameter. [KSPR INT(X) looks like a valid command but BASIC will find the PRINT ([KS*PR INT*(X)) and tokenize it. Syntax errors are particularly frustrating if you are in hi-res mode when the error occurs. The message will appear as a set of colored blocks on the screen and you will have to type blindly to get back to text mode. It helps to include a line in your programs that restores text mode so that you only need to type a GOTO xxx.

RUN/STOP—RESTORE will kill SuperBASIC. It can be

reenabled with POKE 1,54 most of the time. You should be careful when you have changed banks and screen locations. A RUN/STOP—RESTORE will not reset the default video map so you might wipe out page zero or other important RAM.

SuperBASIC Creations

Using SuperBASIC's powerful functions, you can create complex programs much more easily than you thought possible. The following programs demonstrate SuperBASIC in action. Some, such as Programs 2 through 7, are simple BASIC programs which show how you can use SuperBASIC to create impressive graphic displays, joystick-controlled sprites which draw patterns on the screen, or animated sprites. Another program which uses SuperBASIC, "SuperBASIC Sprite Editor," is more complicated, and thus longer. The program is included in another section, "Colors, Characters, and Motion." All are worth the time it will take you to type them in.

Remember that all the following demonstration programs will only work if SuperBASIC has already been loaded and run. As you type these programs in, you'll come across strange-looking commands, such as [DLCS or [FCOL. Don't worry, the program listing is correct; this is how SuperBASIC notes its new commands. Every time you see the [symbol, just press the SHIFT and colon keys together. This will give you the bracket symbol on the screen. Type in the rest of the command (DLCS, for instance) as you would any other command on the 64.

Although most of these programs can be used without much explanation, since prompts appear frequently on the screen, "Type 64" does need some further description to enter and run properly. Remember, all of these programs require SuperBASIC in memory to operate. If you haven't entered and saved a copy of SuperBASIC, Program 1, do that first.

Type 64

Using these two programs, you can turn your 64 into a 64-column display. No hardware adjustment is necessary; the programs create a new character set that is smaller than the one usually seen on the 64. The letters are still easy to read, and actually look quite nice, especially when you change the background color so that it contrasts with the new set.

This is a two-part program. Type 64, Program 8, is in SuperBASIC, and actually loads and operates the character set. Program 9, "64SET," is the new character set which turns your screen into a 64-column display.

First, type in Program 10. Since it's in SuperBASIC, you shouldn't have any problem if you've read and understood this article. In the listing, you'll come across SuperBASIC commands which always begin with the bracket symbol ([). Whenever you see this character, press SHIFT and the colon keys at the same time; that will produce a [symbol on the screen. The rest of the command (FCOL for instance) you can enter normally, of course. Once you've SAVEd the program, you can begin entering 64SET. You'll notice that it's in machine language, so you'll be using the MLX program from Appendix I again. The starting and ending addresses for 64SET are:

Starting address: 32768
Ending address: 34819

Before you LOAD MLX and begin typing in 64SET, enter the following line in direct mode (without line numbers). This moves BASIC and ensures that it will not interfere with the entry of 64SET. If you type in 64SET in several sessions, make sure you enter this line *before* loading and using MLX.

```
POKE 56,128:CLR
```

Enter 64SET as you would any other program which uses MLX. When you're through, SAVE it to disk, using the filename 64SET. Make sure that this is the filename you use (no spaces between 64 and SET—SET in uppercase); if you name it something else, Program 8 won't be able to load and use it. Be sure that both Program 8 and Program 9 are on the same disk.

When both programs are on one disk, type LOAD"TYPE 64",8 and then RUN it. It will load the character set automatically. All you have to do is type on your new 64-column display.

Table1. SuperBASIC Command Summary

Enhanced BASIC Commands

RESTORE <exp>
GOTO <exp>
IF <exp> GOTO <exp>
IF <exp> GOSUB <exp>
ON <exp> GOTO <exp1>,<exp2>,...
ON <exp> GOSUB <exp1>,<exp2>,...
LIST (Shift Key halts list)
ASC(str$) returns 0 for null string

New SuperBASIC Commands

Sprite Commands
[DSPR spr,blk,xexp,yexp,xpos,ypos,multi,sprcolor,Mc0,Mc1
[MOVE spr,xpos,ypos
[KSPR spr
[ESPR spr
[BSPP spr,sel

Sound Commands
[SSND voice,ad,sr,wave,freq,pwidth
[PLAY 256*wave+voice,freq,pwidth

VIC Color Control
[BKGD col
[BKG4 col0,col1,col2,col3
[EXTC col
[FCOL col

VIC Memory Mapping
[BANK sel
[VS1K sel
[CB2K sel

Graphics/Text Control
[ECGR sel
[MCGR sel
[BMGR sel
[MXGR reg,mask,rast1,val1,rast2,val2
[KMXG
[CMXV val1,val2
[SIZE colsel,rowsel
[XYSC xpos,ypos
[DLCS set,address
[FBMS byte
[FSCR byte
[PLOT x,y
[FLIP x,y
[CLPX x,y

[MCPL x,y,sel
[DRAW x1,y1,x2,y2
[UNDR x1,y1,x2,y2
[FLLN x1,y1,x2,y2
[DRW2 x2,y2
[SETP x1,y1
[HRCS address
[CHAR char,x,y
[CHRX char,x,y
[CODE str$
[LOOK address,variable
[STUF address,byte
[HRAM <SuperBASIC mnemonic> <parameter list>

Program 1. SuperBASIC 64

```
2049 :011,008,000,000,158,050,228
2055 :048,056,048,000,000,000,159
2061 :000,000,000,000,000,000,013
2067 :000,000,000,000,000,000,019
2073 :000,000,000,000,000,000,025
2079 :000,169,039,133,001,169,030
2085 :000,133,020,133,078,169,058
2091 :009,133,021,169,192,133,188
2097 :079,162,012,160,000,177,127
2103 :020,145,078,200,208,249,187
2109 :230,021,230,079,202,208,007
2115 :242,160,008,169,104,032,014
2121 :030,171,169,013,141,119,204
2127 :002,141,120,002,169,002,003
2133 :133,198,169,133,141,001,092
2139 :008,169,020,141,002,008,183
2145 :076,120,008,000,000,000,045
2151 :000,031,147,017,017,048,107
2157 :017,157,082,085,078,019,035
2163 :000,000,000,000,000,169,028
2169 :000,133,020,169,160,133,224
2175 :021,162,032,160,000,177,167
2181 :020,145,020,136,208,249,143
2187 :230,021,202,208,244,162,182
2193 :000,160,003,185,224,160,109
2199 :157,224,160,232,200,224,068
2205 :190,208,244,169,003,141,088
2211 :161,168,169,192,141,162,132
2217 :168,169,074,141,210,166,073
2223 :169,193,141,211,166,141,172
2229 :037,160,169,084,141,036,040
```

```
2235 :160,169,219,141,223,160,235
2241 :169,255,141,044,160,169,107
2247 :194,141,045,160,169,038,178
2253 :133,001,169,005,141,143,029
2259 :183,169,076,141,043,169,224
2265 :141,087,169,169,193,141,093
2271 :045,169,141,089,169,169,237
2277 :200,141,088,169,169,227,199
2283 :141,044,169,096,000,000,173
2289 :000,000,000,000,000,000,241
2295 :000,000,000,000,000,000,247
2301 :000,000,000,032,115,000,144
2307 :032,158,173,032,247,183,060
2313 :096,032,139,192,032,000,244
2319 :192,165,020,166,002,157,205
2325 :248,007,032,000,192,165,153
2331 :020,162,029,032,162,192,112
2337 :032,000,192,165,020,162,092
2343 :023,032,162,192,032,097,065
2349 :192,032,000,192,165,020,134
2355 :072,162,028,032,162,192,187
2361 :032,000,192,165,020,166,120
2367 :002,157,039,208,104,240,045
2373 :117,032,000,192,165,020,083
2379 :141,037,208,032,000,192,173
2385 :165,020,141,038,208,169,054
2391 :001,162,021,032,162,192,145
2397 :096,032,139,192,032,000,072
2403 :192,165,021,072,165,020,222
2409 :072,032,000,192,165,002,056
2415 :010,170,232,165,020,157,097
2421 :000,208,202,104,157,000,020
2427 :208,104,162,016,032,162,039
2433 :192,169,000,141,030,208,101
2439 :141,031,208,096,032,000,131
2445 :192,165,020,041,007,133,187
2451 :002,170,169,001,224,000,201
2457 :240,004,010,202,208,252,045
2463 :133,078,096,164,078,201,141
2469 :000,240,006,152,029,000,080
2475 :208,208,006,152,073,255,049
2481 :061,000,208,157,000,208,043
2487 :096,000,007,014,032,019,095
2493 :199,240,150,032,000,192,234
2499 :165,020,041,003,170,189,015
2505 :183,192,133,078,169,212,144
2511 :133,079,032,000,192,165,040
2517 :020,160,005,145,078,032,141
2523 :000,192,165,020,160,006,250
```

```
2529 :145,078,032,028,193,165,098
2535 :020,133,002,160,004,145,183
2541 :078,032,037,193,169,015,249
2547 :141,024,212,096,032,000,236
2553 :192,165,020,041,003,170,072
2559 :189,183,192,133,078,169,175
2565 :212,133,079,165,021,133,236
2571 :002,169,000,160,004,145,235
2577 :078,032,037,193,165,002,012
2583 :160,004,145,078,096,169,163
2589 :000,160,004,145,078,076,236
2595 :000,192,032,000,192,165,104
2601 :021,160,001,145,078,165,099
2607 :020,136,145,078,165,002,081
2613 :201,065,208,016,032,000,063
2619 :192,165,021,041,015,160,141
2625 :003,145,078,165,020,136,100
2631 :145,078,096,173,141,002,194
2637 :208,251,076,044,168,076,132
2643 :029,168,240,251,032,003,038
2649 :192,032,019,166,056,165,207
2655 :095,233,001,164,096,176,092
2661 :001,136,133,065,132,066,122
2667 :096,032,000,192,160,000,075
2673 :177,020,133,002,032,115,080
2679 :000,032,040,175,164,002,020
2685 :169,000,032,145,179,166,048
2691 :071,164,072,032,215,187,104
2697 :096,032,000,192,165,020,130
2703 :133,078,165,021,133,079,240
2709 :032,000,192,165,020,160,206
2715 :000,145,078,096,173,014,149
2721 :220,041,254,141,014,220,027
2727 :165,001,041,253,133,001,249
2733 :169,193,072,169,184,072,008
2739 :032,115,000,076,000,195,085
2745 :165,001,009,002,133,001,240
2751 :173,014,220,009,001,141,237
2757 :014,220,096,165,101,133,158
2763 :254,104,133,002,198,254,124
2769 :208,005,165,002,076,239,136
2775 :167,032,000,192,032,121,247
2781 :000,201,044,240,237,096,015
2787 :032,121,000,201,137,208,158
2793 :003,076,055,169,201,141,110
2799 :240,249,076,050,169,032,031
2805 :115,000,076,245,196,049,158
2811 :000,000,129,000,000,077,201
2817 :079,086,069,093,192,068,076
```

```
2823 :083,080,082,101,196,083,120
2829 :083,078,068,191,192,080,193
2835 :076,065,089,246,192,066,241
2841 :075,071,068,056,195,069,047
2847 :088,084,067,065,195,075,093
2853 :083,080,082,074,195,069,108
2859 :083,080,082,084,195,066,121
2865 :083,080,080,092,195,083,150
2871 :084,085,070,137,193,069,181
2877 :067,071,082,125,195,077,166
2883 :067,071,082,150,195,066,186
2889 :077,071,082,175,195,083,244
2895 :073,090,069,187,195,088,013
2901 :089,083,067,211,195,067,029
2907 :065,084,065,250,195,066,048
2913 :065,078,075,053,196,086,138
2919 :083,049,075,113,196,067,174
2925 :066,050,075,151,196,068,203
2931 :076,067,083,172,196,077,018
2937 :088,071,082,065,197,075,187
2943 :077,088,071,180,197,067,039
2949 :077,088,086,200,197,070,083
2955 :067,079,076,217,197,080,087
2961 :076,079,084,130,198,070,014
2967 :076,073,080,122,198,067,255
2973 :076,080,088,138,198,077,046
2979 :067,080,076,148,198,070,034
2985 :083,067,082,197,198,070,098
2991 :066,077,083,232,198,068,131
2997 :082,065,087,023,203,072,201
3003 :082,067,083,060,201,067,235
3009 :072,065,082,134,202,067,047
3015 :072,082,088,142,202,067,084
3021 :079,068,069,150,202,076,081
3027 :079,079,075,107,193,066,042
3033 :075,071,052,105,195,072,019
3039 :082,065,077,158,193,070,100
3045 :076,076,078,020,203,085,255
3051 :078,068,082,047,203,068,013
3057 :082,087,050,070,203,083,048
3063 :069,084,080,107,203,255,021
3069 :255,255,255,162,000,134,034
3075 :002,160,000,177,122,221,173
3081 :000,194,208,026,232,200,101
3087 :192,004,208,243,189,001,084
3093 :194,072,189,000,194,072,230
3099 :165,122,024,105,003,133,067
3105 :122,144,002,230,123,096,238
3111 :165,002,024,105,006,133,218
```

```
3117 :002,170,189,000,194,201,033
3123 :255,208,206,096,000,000,048
3129 :032,000,192,165,020,141,095
3135 :033,208,096,032,000,192,112
3141 :165,020,141,032,208,096,219
3147 :032,139,192,169,000,162,001
3153 :021,076,162,192,032,139,191
3159 :192,162,021,076,162,192,124
3165 :032,139,192,032,000,192,168
3171 :165,020,162,027,076,162,199
3177 :192,162,000,134,002,032,115
3183 :000,192,165,020,166,002,144
3189 :157,033,208,232,224,004,207
3195 :208,239,096,032,000,192,122
3201 :165,020,162,017,160,064,205
3207 :032,164,192,165,020,240,180
3213 :239,169,000,162,022,160,125
3219 :016,076,164,192,032,000,115
3225 :192,165,020,162,022,160,106
3231 :016,032,164,192,165,020,236
3237 :240,214,169,000,162,017,199
3243 :160,064,076,164,192,032,091
3249 :000,192,165,020,162,017,221
3255 :160,032,076,164,192,032,071
3261 :000,192,165,020,162,022,238
3267 :160,008,032,164,192,032,015
3273 :000,192,165,020,162,017,245
3279 :160,008,076,164,192,032,071
3285 :000,192,165,020,041,007,126
3291 :133,020,173,022,208,041,048
3297 :248,005,020,141,022,208,101
3303 :032,000,192,165,020,041,169
3309 :007,133,020,173,017,208,027
3315 :041,248,005,020,141,017,203
3321 :208,096,169,032,141,000,127
3327 :002,162,000,142,005,002,056
3333 :134,002,173,141,002,208,153
3339 :251,160,000,189,000,194,037
3345 :153,001,002,232,200,192,029
3351 :004,208,244,169,000,160,040
3357 :002,032,030,171,165,002,175
3363 :024,105,006,133,002,170,219
3369 :189,000,194,201,255,208,064
3375 :215,032,115,000,208,251,100
3381 :096,173,002,221,009,003,045
3387 :141,002,221,032,000,192,135
3393 :165,020,041,003,072,073,183
3399 :003,133,020,173,000,221,109
3405 :041,252,005,020,141,000,024
```

```
3411 :221,104,024,106,106,106,238
3417 :133,020,173,136,002,041,082
3423 :063,005,020,141,136,002,206
3429 :096,173,136,002,024,105,125
3435 :003,141,022,192,076,010,039
3441 :192,032,000,192,165,020,202
3447 :041,063,010,010,133,020,140
3453 :173,136,002,041,192,005,162
3459 :020,141,136,002,165,020,103
3465 :010,010,133,020,173,024,251
3471 :208,041,015,005,020,141,061
3477 :024,208,096,173,024,208,114
3483 :041,241,133,002,032,000,092
3489 :192,165,020,041,007,010,084
3495 :005,002,141,024,208,096,131
3501 :173,014,220,041,254,141,248
3507 :014,220,165,001,041,251,103
3513 :133,001,032,000,192,165,196
3519 :020,041,003,162,008,201,114
3525 :002,208,002,162,016,160,235
3531 :208,201,001,208,002,160,215
3537 :216,132,079,160,000,132,160
3543 :078,134,002,032,000,192,141
3549 :166,002,160,000,177,078,036
3555 :145,020,200,208,249,230,255
3561 :021,230,079,202,208,242,191
3567 :165,001,009,004,133,001,040
3573 :173,014,220,009,001,141,035
3579 :014,220,096,000,000,154,223
3585 :000,006,252,000,006,000,009
3591 :033,240,234,234,173,006,159
3597 :197,073,003,141,006,197,118
3603 :170,189,250,193,141,064,002
3609 :197,172,007,197,185,000,015
3615 :208,045,008,197,029,002,008
3621 :197,153,000,208,173,017,017
3627 :208,041,127,029,001,197,134
3633 :141,017,208,189,000,197,033
3639 :141,018,208,169,001,141,221
3645 :025,208,076,129,234,120,085
3651 :169,000,141,014,220,032,131
3657 :000,192,165,020,141,007,086
3663 :197,032,000,192,165,020,173
3669 :141,008,197,032,000,192,143
3675 :165,020,141,003,197,165,014
3681 :021,041,001,240,002,169,059
3687 :128,141,004,197,032,000,093
3693 :192,165,020,141,002,197,058
3699 :032,000,192,165,020,141,153
```

```
3705 :000,197,165,021,041,001,034
3711 :240,002,169,128,141,001,040
3717 :197,032,000,192,165,020,227
3723 :141,005,197,173,017,208,112
3729 :041,127,013,004,197,141,156
3735 :017,208,173,003,197,141,122
3741 :018,208,169,003,141,006,190
3747 :197,169,241,141,026,208,121
3753 :169,011,141,020,003,169,170
3759 :197,141,021,003,088,096,209
3765 :120,169,049,141,020,003,171
3771 :169,234,141,021,003,169,156
3777 :240,141,026,208,088,076,204
3783 :244,193,032,000,192,165,001
3789 :020,141,002,197,032,000,085
3795 :192,165,020,141,005,197,163
3801 :096,032,000,192,165,020,210
3807 :041,015,162,000,157,000,086
3813 :216,157,000,217,157,000,208
3819 :218,157,000,219,232,208,245
3825 :241,141,134,002,096,032,119
3831 :000,192,165,020,041,007,160
3837 :133,002,165,020,041,248,094
3843 :133,251,165,021,133,252,190
3849 :032,000,192,165,020,133,039
3855 :078,041,248,133,020,133,156
3861 :253,169,000,133,254,024,086
3867 :006,253,038,254,006,253,069
3873 :038,254,165,020,101,253,096
3879 :133,253,144,002,230,254,031
3885 :024,006,253,038,254,006,114
3891 :253,038,254,006,253,038,125
3897 :254,165,078,041,007,005,095
3903 :253,133,253,024,165,251,118
3909 :101,253,133,251,165,252,200
3915 :101,254,133,252,173,136,100
3921 :002,041,192,005,252,133,194
3927 :252,173,024,208,041,008,025
3933 :010,010,005,252,133,252,243
3939 :166,002,189,107,198,160,153
3945 :000,096,128,064,032,016,185
3951 :008,004,002,001,192,048,110
3957 :012,003,000,085,170,255,130
3963 :032,246,197,081,251,145,051
3969 :251,096,032,246,197,017,200
3975 :251,145,251,096,032,246,132
3981 :197,073,255,049,251,145,087
3987 :251,096,032,000,192,165,115
3993 :020,041,003,024,105,008,098
```

```
3999 :133,002,006,020,038,021,123
4005 :032,255,197,133,002,032,048
4011 :000,192,165,020,041,003,080
4017 :170,189,119,198,037,002,124
4023 :133,020,165,002,073,255,063
4029 :160,000,049,251,005,020,162
4035 :145,251,096,032,000,192,143
4041 :173,136,002,133,252,169,042
4047 :000,133,251,168,162,003,156
4053 :165,020,145,251,200,208,178
4059 :251,230,252,202,208,246,072
4065 :145,251,200,192,232,208,173
4071 :249,096,032,000,192,173,205
4077 :136,002,041,192,133,252,225
4083 :173,024,208,041,008,010,195
4089 :010,005,252,133,252,169,046
4095 :000,133,251,162,032,160,225
4101 :000,165,020,145,251,200,018
4107 :208,251,230,252,202,208,082
4113 :246,096,032,121,000,208,208
4119 :001,096,104,104,076,070,218
4125 :192,169,000,141,176,002,197
4131 :141,178,002,141,179,002,166
4137 :173,167,002,013,168,002,054
4143 :208,002,056,096,162,024,083
4149 :046,176,002,046,177,002,246
4155 :046,178,002,046,179,002,000
4161 :056,173,178,002,237,167,110
4167 :002,168,173,179,002,237,064
4173 :168,002,144,006,140,178,203
4179 :002,141,179,002,202,208,049
4185 :219,046,176,002,046,177,243
4191 :002,024,096,032,000,192,185
4197 :165,020,141,193,002,165,019
4203 :021,141,194,002,032,000,241
4209 :192,165,020,141,197,002,062
4215 :032,000,192,165,020,141,157
4221 :195,002,165,021,141,196,077
4227 :002,032,000,192,165,020,030
4233 :141,198,002,169,000,141,020
4239 :202,002,056,173,198,002,008
4245 :237,197,002,141,199,002,159
4251 :176,014,169,255,141,202,088
4257 :002,077,199,002,141,199,013
4263 :002,238,199,002,169,000,009
4269 :141,203,002,056,173,195,175
4275 :002,237,193,002,141,200,186
4281 :002,173,196,002,237,194,221
4287 :002,141,201,002,176,027,228
```

```
4293 :169,255,141,203,002,077,020
4299 :200,002,141,200,002,169,149
4305 :255,077,201,002,141,201,062
4311 :002,238,200,002,208,003,100
4317 :238,201,002,169,000,141,204
4323 :204,002,173,199,002,205,244
4329 :200,002,169,000,237,201,018
4335 :002,176,076,173,199,002,099
4341 :208,005,141,205,002,240,022
4347 :105,141,177,002,173,200,025
4353 :002,141,167,002,173,201,175
4359 :002,141,168,002,169,255,232
4365 :141,205,002,032,030,199,110
4371 :144,003,076,058,201,173,162
4377 :176,002,013,177,002,208,091
4383 :020,169,255,141,176,002,026
4389 :141,177,002,169,000,141,155
4395 :208,002,169,025,141,209,029
4401 :002,208,049,169,000,141,106
4407 :208,002,141,209,002,240,089
4413 :039,169,255,141,204,002,103
4419 :173,200,002,024,109,201,008
4425 :002,240,171,173,199,002,092
4431 :141,167,002,169,000,141,187
4437 :168,002,173,200,002,141,003
4443 :177,002,169,255,141,205,016
4449 :002,076,016,200,238,200,061
4455 :002,238,199,002,173,193,142
4461 :002,041,007,133,002,173,211
4467 :193,002,041,248,133,251,215
4473 :173,194,002,133,252,173,024
4479 :197,002,032,014,198,017,075
4485 :251,145,251,173,204,002,135
4491 :208,095,173,203,002,240,036
4497 :016,056,173,193,002,233,050
4503 :001,141,193,002,176,013,165
4509 :206,194,002,144,008,238,181
4515 :193,002,208,003,238,194,233
4521 :002,056,173,200,002,233,067
4527 :001,141,200,002,176,003,186
4533 :206,201,002,024,173,200,219
4539 :002,109,201,002,240,120,093
4545 :173,205,002,240,165,024,234
4551 :173,176,002,109,208,002,101
4557 :141,208,002,173,177,002,140
4563 :109,209,002,141,209,002,115
4569 :144,144,173,202,002,240,098
4575 :006,206,197,002,076,107,049
4581 :200,238,197,002,076,107,025
```

```
4587 :200,173,202,002,240,006,034
4593 :206,197,002,076,250,200,148
4599 :238,197,002,206,199,002,067
4605 :240,058,173,205,002,240,147
4611 :040,024,173,176,002,109,015
4617 :208,002,141,208,002,173,231
4623 :177,002,109,209,002,141,143
4629 :209,002,144,019,173,203,003
4635 :002,240,017,056,173,193,196
4641 :002,233,001,141,193,002,093
4647 :176,003,206,194,002,076,184
4653 :107,200,238,193,002,208,225
4659 :248,238,194,002,208,243,160
4665 :096,198,122,096,032,000,089
4671 :192,165,020,141,075,201,089
4677 :165,021,141,076,201,096,001
4683 :143,183,000,169,000,141,199
4689 :193,002,141,196,002,032,135
4695 :000,192,165,020,141,197,034
4701 :002,032,000,192,169,056,032
4707 :197,020,169,001,229,021,224
4713 :176,005,169,255,141,193,020
4719 :002,165,020,041,007,133,223
4725 :002,165,020,041,248,133,214
4731 :251,165,021,133,252,032,209
4737 :000,192,169,192,197,020,131
4743 :176,005,169,255,141,196,053
4749 :002,165,020,041,007,141,005
4755 :194,002,141,195,002,165,078
4761 :020,032,014,198,165,251,065
4767 :041,248,133,251,173,197,178
4773 :002,133,020,169,000,133,110
4779 :021,006,020,038,021,006,027
4785 :020,038,021,006,020,038,064
4791 :021,024,173,075,201,101,010
4797 :020,133,020,165,021,109,145
4803 :076,201,133,021,024,165,047
4809 :251,105,008,141,177,002,117
4815 :165,252,105,000,141,178,024
4821 :002,165,021,041,208,201,083
4827 :208,208,007,120,165,001,160
4833 :041,251,133,001,169,000,052
4839 :141,176,002,166,002,240,190
4845 :005,056,106,202,208,251,041
4851 :141,179,002,172,176,002,147
4857 :177,020,166,002,240,004,090
4863 :074,202,208,252,032,077,076
4869 :202,208,238,044,193,002,124
4875 :048,056,056,169,008,229,065
```

```
4881 :002,133,002,201,008,240,091
4887 :045,173,177,002,133,251,036
4893 :173,178,002,133,252,169,168
4899 :000,141,176,002,173,194,209
4905 :002,141,195,002,173,179,221
4911 :002,073,255,141,179,002,187
4917 :172,176,002,177,020,166,254
4923 :002,010,202,208,252,032,253
4929 :077,202,208,240,169,004,197
4935 :005,001,133,001,088,096,139
4941 :172,195,002,044,077,201,000
4947 :048,012,133,254,173,179,114
4953 :002,049,251,005,254,076,214
4959 :099,202,081,251,145,251,100
4965 :200,140,195,002,192,008,070
4971 :208,017,160,064,140,195,123
4977 :002,230,252,044,196,002,071
4983 :016,005,169,007,141,176,121
4989 :002,238,176,002,173,176,124
4995 :002,201,008,096,169,000,095
5001 :141,077,201,076,078,201,143
5007 :169,255,141,077,201,076,038
5013 :078,201,032,115,000,032,095
5019 :040,175,234,234,234,234,026
5025 :234,234,165,071,133,020,250
5031 :165,072,133,021,160,000,206
5037 :177,020,240,213,056,165,020
5043 :020,233,002,133,020,176,251
5049 :002,198,021,177,020,197,032
5055 :069,208,196,200,177,020,037
5061 :197,070,208,189,160,003,000
5067 :177,020,133,251,200,177,137
5073 :020,133,252,169,000,133,148
5079 :253,133,002,133,254,160,126
5085 :000,177,071,170,164,002,037
5091 :177,251,201,018,208,007,065
5097 :169,128,133,253,076,009,233
5103 :203,201,146,208,007,169,149
5109 :000,133,253,076,009,203,151
5115 :041,191,016,002,073,192,254
5121 :005,253,164,254,145,251,049
5127 :230,254,230,002,202,208,109
5133 :211,165,254,160,000,145,180
5139 :071,096,169,081,044,169,137
5145 :017,141,132,200,076,098,177
5151 :199,169,049,141,132,200,153
5157 :169,065,141,130,200,169,143
5163 :203,141,131,200,096,032,078
5169 :032,203,032,098,199,169,014
```

```
5175 :014,141,130,200,169,198,139
5181 :141,131,200,096,032,014,163
5187 :198,073,255,096,173,195,033
5193 :002,141,193,002,173,196,012
5199 :002,141,194,002,173,198,021
5205 :002,141,197,002,173,132,220
5211 :200,201,049,208,009,032,022
5217 :037,203,032,119,199,076,251
5223 :054,203,076,119,199,032,018
5229 :000,192,165,020,141,195,054
5235 :002,165,021,041,001,141,230
5241 :196,002,032,000,192,165,196
5247 :020,141,198,002,096,000,072
5253 :163,020,010,000,153,034,001
5259 :154,147,083,085,080,069,245
5265 :082,066,065,083,073,067,069
5271 :032,086,051,032,048,049,193
5277 :049,048,056,052,034,000,140
5283 :187,020,015,000,153,034,060
5289 :066,089,032,077,065,082,068
5295 :084,073,078,032,067,032,029
5301 :075,069,069,083,034,000,255
5307 :208,020,020,000,153,034,110
5313 :091,067,065,084,065,034,087
5319 :058,144,067,065,084,065,170
5325 :058,162,000,000,000,013,182
```

Program 2. Moiré Pattern

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

```
1 REM MOIRE TITLE PAGE DEMO                    :rem 85
5 [EXTC0                                       :rem 244
10 [CB2K4:[BMGR1:[FBMS0 :[FSCR1                :rem 193
15 FORJ=0 TO318 STEP2                          :rem 177
20 [DRAWJ,198,160,100 :NEXT                    :rem 252
22 FORJ=0 TO318 STEP2                          :rem 175
23 [DRAWJ,0,160,100 :NEXT                      :rem 141
24 FORJ=0 TO198 STEP2                          :rem 183
25 [DRAW160,100,318,J{3 SPACES}:NEXT           :rem 251
26 FORJ=0 TO198 STEP2                          :rem 185
27 [DRAW161,100,0,J{3 SPACES}:NEXT             :rem 146
29 [EXTC4                                       :rem 46
30 M$="SUPERBASIC":[HRCS53248:M$=M$+""         :rem 217
40 X=120:Y=80:GOSUB50                          :rem 227
45 M$="{RVS}BY MCSOFT":M$=M$+"":X=124:Y=120:GOSUB5
   0                                           :rem 132
47 [CHRX54,152,89:[CHRX52,160,89                :rem 86
48 FORJ=1TO800:NEXT                            :rem 191
49 [FSCR16:{5 SPACES}GOTO100                    :rem 97
```

```
50 [CODEM$:FORJ=1TOLEN(M$)                    :rem 105
60 [CHRXASC(MID$(M$,J,1)),X,Y                 :rem 186
70 X=X+8:NEXT                                  :rem 48
80 RETURN                                      :rem 72
100 GETA$:IFA$=""THEN100                       :rem 69
110 [BMGR0:[CB2K2                              :rem 14
```

Program 3. Stars

```
1 REM STAR DEMO                               :rem 116
10 PI=2*↑                                     :rem 146
20 INPUT"{CLR}STAR POINTS WANTED (0 TO END)";PW
                                              :rem 128
21 IFPW=0THENEND                               :rem 12
22 INPUT"SKIP TRY VALUE CLOSE TO HALF # POINTS)";S
   K                                           :rem 33
23 INPUT"RADIUS <100 ";R                       :rem 91
30 P=PI/PW                                     :rem 95
50 [BMGR1:[CB2K4:[FBMS0:[FSCR1                :rem 197
60 X=160:Y=100-R:TL=0                         :rem 185
70 FORJ=1TOPW                                  :rem 80
80 TH=TL+SK                                   :rem 170
90 TL=TH:TH=TH*P-(PI/4)                       :rem 133
100 X2=COS(TH)*R+160                          :rem 104
110 Y2=SIN(TH)*R+100                          :rem 105
120 [DRAWX,Y,X2,Y2                            :rem 102
130 X=INT(X2):Y=INT(Y2):NEXT                  :rem 255
140 GETA$:IFA$=""THEN140                       :rem 77
150 [BMGR0:[CB2K2:PRINT"{CLR}":GOTO20         :rem 133
```

Program 4. Circles

```
5 REM CIRCLE DEMO                             :rem 240
10 INPUT"CENTER X,Y";A,B                      :rem 189
20 INPUT"RADIUS";R                            :rem 139
40 [FSCR1:[CB2K4:[BMGR1:[FBMS0                :rem 196
50 PH=0:Y1=0:X1=R                             :rem 237
60 PY=PH+Y1+Y1+1                              :rem 170
70 PX=PY-X1-X1+1                              :rem 189
80 [PLOT A+X1,B+Y1                             :rem 26
90 [PLOT A-X1,B+Y1                             :rem 29
100 [PLOTA+X1,B-Y1                             :rem 69
110 [PLOTA-X1,B-Y1                             :rem 72
120 [PLOTA+Y1,B+X1                             :rem 69
130 [PLOTA-Y1,B+X1                             :rem 72
140 [PLOTA+Y1,B-X1                             :rem 73
150 [PLOTA-Y1,B-X1                             :rem 76
160 PH=PY:Y1=Y1+1                             :rem 252
170 IFABS(PX)<ABS(PY)THENPH=PX:X1=X1-1        :rem 149
```

```
180 IFX1>=Y1THEN60                                    :rem 75
200 GETA$:IFA$=""THEN200                              :rem 71
210 [BMGR0:[CB2K2                                     :rem 15
220 INPUT"{CLR}CENTER X,Y";A,B                        :rem 131
230 INPUT"RADIUS";R                                   :rem 190
240 [FSCR1:[CB2K4:[BMGR1:GOTO50                       :rem 225
```

Program 5. Joystick-Controlled Sprites

```
1 REM DOODLE                                          :rem 204
5 GOSUB900:[DSPR1,13,0,0,160+16,100+44,0,0:GOSUB14
  0                                                   :rem 65
10 [BANK0:[CB2K4:[BMGR1:[FBMS0:[FSCR1:[BSPP1,1
                                                      :rem 250
20 E=1:X=160:Y=100:C=-1:FORQ=1TO100:NEXT              :rem 129
30 IFPEEK(203)=60THEN130                              :rem 99
31 IFPEEK(203)=4THENE=-E:IFE>0THEN[DSPR1,13,0,0,0,
   0,0,0                                              :rem 186
32 IFE<0THEN[DSPR1,13,0,0,X+16,Y+44,0,12:[CLPXX,Y
                                                      :rem 163
35 JV=PEEK(56320):FR=JVAND16                          :rem 164
40 JV=15-(JVAND15)                                    :rem 254
50 IFJV=0ANDFR=16THEN30                               :rem 162
60 IFJV=1ORJV=5ORJV=9THENY=Y-1:IFY<0THENY=199
                                                      :rem 223
70 IFJV=2ORJV=6ORJV=10THENY=Y+1:IFY>199THENY=0
                                                      :rem 10
80 IFJV>=4ANDJV<=6THENX=X-1:IFX<0THENX=319:rem 208
90 IFJV>=8ANDJV<=10THENX=X+1:IFX>319THENX=0 :rem 0
100 IFFR=0ANDJV=0THENC=-C:E=1:FORQ=1TO100:NEXT:IFC
    >0THEN[KSPR1:POKE53288,0                          :rem 226
105 IFE<0THEN[ESPR1:[MOVE1,X+16,Y+44:[CLPXX,Y:GOTO
    30                                                :rem 117
110 IFC>0THEN[PLOTX,Y:GOTO30                          :rem 78
120 IFC<0THEN[ESPR1:[MOVE1,X+16,Y+44:GOTO30
                                                      :rem 199
130 [BANK0:[BMGR0:[CB2K2:POKE198,0:PRINT"{CLR}":[K
    SPR1:END                                          :rem 13
140 PRINT"{CLR}DOODLE 64"                             :rem 26
150 PRINT"{DOWN}USE JOYSTICK IN PORT 2"               :rem 227
160 PRINT"BUTTON TURNS INK ON/OFF"                    :rem 105
165 PRINT"F1 TURNS ERASE MODE ON/OFF"                 :rem 188
170 PRINT"HIT A KEY TO START"                         :rem 169
180 PRINT"HIT {RVS}SPACE{OFF} TO STOP"                :rem 72
185 PRINT"THE BLACK + IS YOUR CURSOR WHEN INK=OFF"
                                                      :rem 205
186 PRINT"THE GREY + IS YOUR CURSOR WHEN ERASE=ON"
    :[BKGD1:[FCOL0                                    :rem 191
190 GETA$:IFA$=""THEN190                              :rem 87
```

```
200 IFA$=" "THENRETURN                          :rem 22
210 RETURN                                      :rem 115
900 X=13*64                                     :rem 38
910 READY:IFY<ØTHENRETURN                       :rem 172
920 POKEX,Y:X=X+1:GOTO91Ø                       :rem 55
1000 DATA1,192,Ø,1,192,Ø,1,192,Ø,1,192,Ø,1,192,Ø
                                                :rem 52
1010 DATAØ,128,Ø,126,63,Ø,Ø,128,Ø,1,192,Ø,1,192,Ø
                                                :rem 102
1020 DATA1,192,Ø,1,192,Ø,1,192,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
                                                :rem 92
1030 DATAØ,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø      :rem 22
1040 DATAØ,Ø,Ø,-1                               :rem 81
```

Program 6. Sprite Animation

```
1 REM FALLING SHAMROCKS                         :rem 189
2 REM HIT A KEY TO STOP PROGRAM                 :rem 38
5 [EXTC13:[CB2K4:[BMGR1:[FSCR5:[FBMS171         :rem 47
10 X=832:V=53265:R=128                          :rem 201
20 READA:IFA<ØTHEN35                            :rem 204
30 POKEX,A:X=X+1:GOTO2Ø                         :rem 175
35 FORJ=ØTO7                                    :rem 224
40 [DSPRJ,13,1,1,Ø,Ø,Ø,5+J{2 SPACES}:NEXT       :rem 239
50 FORJ=1TO256:FORK=1TO8:[MOVEK-1,J+K*K,J*K+K:NEXT
   :WAITV,R:[FSCRJ/2                            :rem 90
55 GETA$:IFA$<>""THEN3ØØ                        :rem 93
56 NEXT                                         :rem 170
60 X=PEEK(8192)+1:[FBMSX:GOTO5Ø                 :rem 142
100 DATAØ,102,Ø,Ø,255,Ø,1,255,128,3,255,192:rem 81
110 DATA3,255,192,25,255,152,60,126,60,126,126,126
                                                :rem 198
120 DATA255,60,255,255,255,255,127,255,254,255,255
    ,255,255                                    :rem 112
130 DATA24,255,126,24,126,60,24,60,24,24,24,Ø,24,Ø
    ,Ø,24,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,-1               :rem 190
300 [CB2K2:[BMGRØ:FORJ=ØTO7:[KSPRJ:NEXT         :rem 89
```

Program 7. Simple PET Emulator

```
10 REM ROUTINE TO SET BASIC MEMORY AND SCREEN TO P
   ET STANDARD LOCATIONS                        :rem 81
20 REM SCREEN AT 32768                          :rem 165
30 REM BASIC 1024 TO 32767                      :rem 28
40 REM ASSUME IN C-64 STANDARD MAP              :rem 182
50 [FSCR Ø:[VS1K Ø :[BANK 2 :PRINT"{CLR}"       :rem 137
60 POKE44,4:POKE 45,3:POKE46,4                  :rem 193
70 POKE55,Ø:POKE56,128                          :rem 247
80 NEW                                          :rem 82
```

Program 8. Type 64

```
2 IFA=ØTHENA=1:LOAD"64SET",8,1                    :rem 2
3 PRINT"{CLR}USE F1 TO END PROGRAM":PRINT"HIT ANY
  {SPACE}KEY TO BEGIN"                            :rem 86
4 GETA$:IFA$=""THEN4                              :rem 139
5 POKE56,128:POKE55,Ø:CLR                         :rem 224
10 [BMGR1:[CB2K4:[FBMSØ:[FSCR1                    :rem 193
20 [HRCS32768                                     :rem 247
25 X=Ø:Y=Ø:W=5:RV=Ø                               :rem 126
30 BL=Ø                                           :rem 94
35 FORJ=1TO3Ø:GETK$:IFK$=""THENNEXT               :rem 189
40 IFJ=31ANDBL=ØTHEN[CHRX16Ø,X,Y:BL=1:GOTO35
                                                  :rem 28
42 IFJ=31THEN[CHRX16Ø,X,Y:GOTO3Ø                  :rem 21
44 J=31:NEXT:IFBL=1THENBL=Ø:[CHRX16Ø,X,Y          :rem 37
50 IFK$="{CLR}"THEN[FBMSØ:GOTO25                  :rem 51
55 IFK$=CHR$(13)THENX=Ø:Y=Y+8:GOTO14Ø             :rem 227
60 IFK$="{HOME}"THEN25                            :rem 142
65 IFK$="{RVS}"THENRV=128:GOTO3Ø                  :rem 129
66 IFK$="{OFF}"THENRV=Ø:GOTO3Ø                    :rem 151
70 IFK$="{DOWN}"THENY=Y+8:Y=-Y*(Y<193):J=31:GOTO4Ø
                                                  :rem 119
80 IFK$="{UP}"THENY=Y-8:Y=-Y*(Y>=Ø):J=31:GOTO4Ø
                                                  :rem 204
90 IFK$="{RIGHT}"THENX=X+W:X=-X*(X<316):J=31:GOTO4
   Ø                                              :rem 156
100 IFK$="{LEFT}"THENX=X-W:X=-X*(X>=Ø):J=31: GOTO4
    Ø                                             :rem 27
105 IFK$="{F1}"THEN2ØØ                            :rem 91
106 IFK$=" "THEN[CHAR32+RV,X,Y:GOTO12Ø             :rem 5
110 [CODEK$:CR=ASC(K$):[CHRX CR+RV,X,Y            :rem 85
120 X=X+W:X=-X*(X>=ØANDX<316):IFX=ØTHENY=Y+8
                                                  :rem 228
140 Y=-Y*(Y<193)                                  :rem 94
150 GOTO3Ø                                        :rem 50
200 [BMGRØ:[CB2K2:PRINT"{CLR}"                    :rem 172
```

Program 9. 64Set

```
32768 :ØØØ,224,176,Ø16,2Ø8,2Ø8,Ø64
32774 :112,ØØØ,Ø96,240,144,240,Ø7Ø
32780 :144,144,144,ØØØ,224,144,Ø44
32786 :144,224,144,144,224,ØØØ,13Ø
32792 :Ø96,144,128,128,128,144,Ø24
32798 :Ø96,ØØØ,192,16Ø,144,144,254
32804 :144,16Ø,192,ØØØ,240,128,132
32810 :128,224,128,128,240,ØØØ,122
32816 :240,128,128,224,128,128,ØØØ
32822 :128,ØØØ,Ø96,144,128,176,214
```

```
32828 :144,144,096,000,144,144,220
32834 :144,240,144,144,144,000,114
32840 :112,032,032,032,032,032,088
32846 :112,000,048,016,016,016,030
32852 :144,144,096,000,144,144,244
32858 :160,192,160,144,144,000,122
32864 :128,128,128,128,128,128,096
32870 :240,000,144,240,240,144,086
32876 :144,144,144,000,144,208,124
32882 :208,208,176,176,144,000,002
32888 :096,144,144,144,144,144,168
32894 :096,000,224,144,144,224,190
32900 :128,128,128,000,096,144,244
32906 :144,144,144,096,048,000,202
32912 :224,144,144,224,144,144,144
32918 :144,000,096,144,128,096,246
32924 :016,144,096,000,240,064,204
32930 :064,064,064,064,064,000,226
32936 :144,144,144,144,144,144,008
32942 :096,000,144,144,144,144,078
32948 :144,096,096,000,144,144,036
32954 :144,144,240,240,144,000,074
32960 :144,144,096,096,240,144,032
32966 :144,000,144,144,144,240,246
32972 :096,096,096,000,240,016,236
32978 :032,064,128,128,240,000,034
32984 :192,128,128,128,128,128,024
32990 :192,000,224,112,064,240,030
32996 :064,112,224,000,048,016,180
33002 :016,016,016,016,048,000,090
33008 :000,096,096,240,096,096,096
33014 :096,096,016,032,064,240,022
33020 :064,032,016,000,000,000,108
33026 :000,000,000,000,000,000,002
33032 :032,032,032,032,032,000,168
33038 :032,000,000,144,144,000,078
33044 :000,000,000,000,112,096,228
33050 :096,240,096,096,224,000,010
33056 :032,096,144,192,048,144,176
33062 :096,032,128,144,032,064,022
33068 :144,016,000,000,224,160,076
33074 :192,160,144,144,112,000,034
33080 :096,192,128,000,000,000,216
33086 :000,000,096,192,128,128,094
33092 :128,192,096,000,192,096,004
33098 :048,048,048,096,192,000,250
33104 :000,144,096,240,096,144,032
33110 :000,000,000,064,064,224,182
33116 :064,064,000,000,000,000,220
```

```
33122 :000,000,000,096,096,192,226
33128 :000,000,000,240,000,000,088
33134 :000,000,000,000,000,000,110
33140 :000,096,096,000,000,000,052
33146 :008,016,032,064,128,000,114
33152 :096,144,176,144,208,144,016
33158 :096,000,096,032,032,032,166
33164 :032,032,112,000,096,144,044
33170 :016,032,064,128,240,000,114
33176 :096,144,016,096,016,144,152
33182 :096,000,032,160,160,160,254
33188 :240,032,032,000,240,128,068
33194 :128,224,016,144,096,000,010
33200 :096,128,128,224,144,144,016
33206 :096,000,240,016,016,032,070
33212 :032,032,032,000,096,144,012
33218 :144,096,144,144,096,000,050
33224 :096,144,144,112,016,016,216
33230 :096,000,000,000,096,000,142
33236 :000,096,000,000,000,000,052
33242 :096,000,000,096,096,192,186
33248 :016,032,064,128,064,032,048
33254 :016,000,000,000,240,000,230
33260 :240,000,000,000,128,064,156
33266 :032,016,032,064,128,000,002
33272 :096,144,016,032,032,000,056
33278 :032,000,000,224,176,016,190
33284 :208,208,112,000,096,240,100
33290 :144,240,144,144,144,000,058
33296 :224,144,144,224,144,144,016
33302 :224,000,096,144,128,128,230
33308 :128,144,096,000,192,160,236
33314 :144,144,144,160,192,000,050
33320 :240,128,128,224,128,128,248
33326 :240,000,240,128,128,224,238
33332 :128,128,128,000,096,144,164
33338 :128,176,144,144,096,000,234
33344 :144,144,144,240,144,144,000
33350 :144,000,112,032,032,032,166
33356 :032,032,112,000,048,016,060
33362 :016,016,144,144,096,000,242
33368 :144,144,160,192,160,144,008
33374 :144,000,128,128,128,128,238
33380 :128,128,240,000,144,240,212
33386 :240,144,144,144,144,000,154
33392 :144,208,208,208,176,176,208
33398 :144,000,096,144,144,144,022
33404 :144,144,096,000,224,144,108
33410 :144,224,128,128,128,000,114
```

```
33416 :096,144,144,144,144,096,136
33422 :048,000,224,144,144,224,158
33428 :144,144,144,000,096,144,052
33434 :128,096,016,144,096,000,122
33440 :240,064,064,064,064,064,208
33446 :064,000,144,144,144,144,038
33452 :144,144,096,000,144,144,076
33458 :144,144,144,096,096,000,034
33464 :144,144,144,144,240,240,216
33470 :144,000,144,144,096,096,046
33476 :240,144,144,000,144,144,244
33482 :144,240,096,096,096,000,106
33488 :240,016,032,064,128,128,048
33494 :240,000,192,128,128,128,006
33500 :128,128,192,000,224,112,236
33506 :064,240,064,112,224,000,162
33512 :048,016,016,016,016,016,104
33518 :048,000,000,096,096,240,206
33524 :096,096,096,096,016,032,164
33530 :064,240,064,032,016,000,154
33536 :000,000,000,000,000,000,000
33542 :000,000,032,032,032,032,134
33548 :032,000,032,000,000,144,220
33554 :144,000,000,000,000,000,162
33560 :112,096,096,240,096,096,248
33566 :224,000,032,096,144,192,206
33572 :048,144,096,032,128,144,116
33578 :032,064,144,016,000,000,042
33584 :224,160,192,160,144,144,048
33590 :112,000,096,192,128,000,070
33596 :000,000,000,000,096,192,092
33602 :128,128,128,192,096,000,226
33608 :192,096,048,048,048,096,088
33614 :192,000,000,144,096,240,238
33620 :096,144,000,000,000,064,132
33626 :064,224,064,064,000,000,250
33632 :000,000,000,000,000,096,192
33638 :096,192,000,000,000,240,118
33644 :000,000,000,000,000,000,108
33650 :000,000,000,096,096,000,050
33656 :000,000,008,016,032,064,240
33662 :128,000,096,144,176,144,046
33668 :208,144,096,000,096,032,196
33674 :032,032,032,032,112,000,122
33680 :096,144,016,032,064,128,112
33686 :240,000,096,144,016,096,230
33692 :016,144,096,000,032,160,092
33698 :160,160,240,032,032,000,018
33704 :240,128,128,224,016,144,024
```

```
33710 :096,000,096,128,128,224,078
33716 :144,144,096,000,240,016,052
33722 :016,032,032,032,032,000,074
33728 :096,144,144,096,144,144,192
33734 :096,000,096,144,144,112,022
33740 :016,016,096,000,000,000,076
33746 :096,000,000,096,000,000,146
33752 :000,000,096,000,000,096,152
33758 :096,192,016,032,064,128,238
33764 :064,032,016,000,000,000,084
33770 :240,000,240,000,000,000,202
33776 :128,064,032,016,032,064,064
33782 :128,000,096,144,016,032,150
33788 :032,000,032,000,248,024,076
33794 :072,232,040,040,136,248,002
33800 :152,008,104,008,104,104,232
33806 :104,248,024,104,104,024,110
33812 :104,104,024,248,152,104,244
33818 :120,120,120,104,152,248,122
33824 :056,088,104,104,104,088,064
33830 :056,248,008,120,120,024,102
33836 :120,120,008,248,008,120,156
33842 :120,024,120,120,120,248,034
33848 :152,104,120,072,104,104,200
33854 :152,248,104,104,104,008,014
33860 :104,104,104,248,136,216,212
33866 :216,216,216,216,136,248,042
33872 :200,232,232,232,104,104,160
33878 :152,248,104,104,088,056,070
33884 :088,104,104,248,120,120,108
33890 :120,120,120,120,008,248,066
33896 :104,008,008,104,104,104,024
33902 :104,248,104,040,040,040,174
33908 :072,072,104,248,152,104,100
33914 :104,104,104,104,152,248,170
33920 :024,104,104,024,120,120,112
33926 :120,248,152,104,104,104,198
33932 :104,152,200,248,024,104,204
33938 :104,024,104,104,104,248,066
33944 :152,104,120,152,232,104,248
33950 :152,248,008,184,184,184,094
33956 :184,184,184,248,104,104,148
33962 :104,104,104,104,152,248,218
33968 :104,104,104,104,104,152,080
33974 :152,248,104,104,104,104,230
33980 :008,008,104,248,104,104,252
33986 :152,152,008,104,104,248,194
33992 :104,104,104,008,152,152,056
33998 :152,248,008,232,216,184,222
```

```
34004 :120,120,008,248,056,120,116
34010 :120,120,120,120,056,248,234
34016 :024,136,184,008,184,136,128
34022 :024,248,200,232,232,232,118
34028 :232,232,200,248,248,152,012
34034 :152,008,152,152,152,152,242
34040 :232,216,184,008,184,216,008
34046 :232,248,248,248,248,248,190
34052 :248,248,248,248,216,216,148
34058 :216,216,216,248,216,248,090
34064 :248,104,104,248,248,248,192
34070 :248,248,136,152,152,008,198
34076 :152,152,024,248,216,152,204
34082 :104,056,200,104,152,216,098
34088 :120,104,216,184,104,232,232
34094 :248,248,024,088,056,088,030
34100 :104,104,136,248,152,056,084
34106 :120,248,248,248,248,248,138
34112 :152,056,120,120,120,056,176
34118 :152,248,056,152,200,200,054
34124 :200,152,056,248,248,104,060
34130 :152,008,152,104,248,248,226
34136 :248,184,184,024,184,184,072
34142 :248,248,248,248,248,248,046
34148 :248,152,152,056,248,248,180
34154 :248,008,248,248,248,248,074
34160 :248,248,248,248,248,152,224
34166 :152,248,248,248,248,232,214
34172 :216,184,120,248,152,104,124
34178 :072,104,040,104,152,248,082
34184 :152,216,216,216,216,216,088
34190 :136,248,152,104,232,216,206
34196 :184,120,008,248,152,104,196
34202 :232,152,232,104,152,248,250
34208 :216,088,088,088,008,216,096
34214 :216,248,008,120,120,024,134
34220 :232,104,152,248,152,120,156
34226 :120,024,104,104,152,248,162
34232 :008,232,232,216,216,216,024
34238 :216,248,152,104,104,152,142
34244 :104,104,152,248,152,104,036
34250 :104,136,232,232,152,248,026
34256 :248,248,152,248,248,152,224
34262 :248,248,248,248,152,248,070
34268 :248,152,152,056,232,216,252
34274 :184,120,184,216,232,248,130
34280 :248,248,008,248,008,248,216
34286 :248,248,120,184,216,232,206
34292 :216,184,120,248,152,104,244
```

```
34298 :232,216,216,248,216,248,090
34304 :248,024,072,232,040,040,144
34310 :136,248,152,008,104,008,150
34316 :104,104,104,248,024,104,188
34322 :104,024,104,104,024,248,114
34328 :152,104,120,120,120,104,232
34334 :152,248,056,088,104,104,014
34340 :104,088,056,248,008,120,148
34346 :120,024,120,120,008,248,170
34352 :008,120,120,024,120,120,048
34358 :120,248,152,104,120,072,102
34364 :104,104,152,248,104,104,108
34370 :104,008,104,104,104,248,226
34376 :136,216,216,216,216,216,008
34382 :136,248,200,232,232,232,078
34388 :104,104,152,248,104,104,132
34394 :088,056,088,104,104,248,010
34400 :120,120,120,120,120,120,048
34406 :008,248,104,008,008,104,070
34412 :104,104,104,248,104,040,044
34418 :040,040,072,072,104,248,178
34424 :152,104,104,104,104,104,024
34430 :152,248,024,104,104,024,014
34436 :120,120,120,248,152,104,228
34442 :104,104,104,152,200,248,026
34448 :024,104,104,024,104,104,096
34454 :104,248,152,104,120,152,006
34460 :232,104,152,248,008,184,060
34466 :184,184,184,184,184,248,050
34472 :104,104,104,104,104,104,024
34478 :152,248,104,104,104,104,222
34484 :104,152,152,248,104,104,020
34490 :104,104,008,008,104,248,250
34496 :104,104,152,152,008,104,048
34502 :104,248,104,104,104,008,102
34508 :152,152,152,248,008,232,124
34514 :216,184,120,120,008,248,082
34520 :056,120,120,120,120,120,104
34526 :056,248,024,136,184,008,110
34532 :184,136,024,248,200,232,228
34538 :232,232,232,232,200,248,074
34544 :248,152,152,008,152,152,080
34550 :152,152,232,216,184,008,166
34556 :184,216,232,248,248,248,092
34562 :248,248,248,248,248,248,210
34568 :216,216,216,216,216,248,056
34574 :216,248,248,104,104,248,158
34580 :248,248,248,248,136,152,020
34586 :152,008,152,152,024,248,250
```

```
34592 :216,152,104,056,200,104,096
34598 :152,216,120,104,216,184,006
34604 :104,232,248,248,024,088,220
34610 :056,088,104,104,136,248,018
34616 :152,056,120,248,248,248,104
34622 :248,248,152,056,120,120,238
34628 :120,056,152,248,056,152,084
34634 :200,200,200,152,056,248,106
34640 :248,104,152,008,152,104,080
34646 :248,248,248,184,184,024,198
34652 :184,184,248,248,248,248,172
34658 :248,248,248,152,152,056,178
34664 :248,248,248,008,248,248,072
34670 :248,248,248,248,248,248,062
34676 :248,152,152,248,248,248,132
34682 :248,232,216,184,120,248,090
34688 :152,104,072,104,040,104,192
34694 :152,248,152,216,216,216,054
34700 :216,216,136,248,152,104,188
34706 :232,216,184,120,008,248,130
34712 :152,104,232,152,232,104,104
34718 :152,248,216,088,088,088,014
34724 :008,216,216,248,008,120,212
34730 :120,024,232,104,152,248,026
34736 :152,120,120,024,104,104,032
34742 :152,248,008,232,232,216,246
34748 :216,216,216,248,152,104,060
34754 :104,152,104,104,152,248,034
34760 :152,104,104,136,232,232,136
34766 :152,248,248,248,152,248,222
34772 :248,152,248,248,248,248,068
34778 :152,248,248,152,152,056,202
34784 :232,216,184,120,184,216,096
34790 :232,248,248,248,008,248,182
34796 :008,248,248,248,120,184,012
34802 :216,232,216,184,120,248,178
34808 :152,104,232,216,216,248,136
34814 :216,248,013,013,013,013,002
```

Copyfile

Gregor Larson

Copying files—both BASIC and machine language programs—is simple and fast when you use this program. A short machine language routine, "Copyfile" allows you to make file copies using only one disk drive.

One drawback of a single disk drive is its inability to copy files from one disk to another. BASIC programs can be copied by loading them into the computer, then saving them out to another disk, but sequential files, such as user or machine language program files, can be difficult to copy with just one drive.

"Copyfile" allows you to duplicate these files. It reads the whole file into the machine and then waits until you press the C key. Then it writes the entire file back to another disk. The program is written in machine language, so it's fast. It also makes good use of the 64's memory, and can copy a file of more than 170 blocks.

Enter and Use

To enter Copyfile, use the MLX program in Appendix I. MLX makes it easier to enter the sometimes complicated machine language code. Before you begin to type in Copyfile, read Appendix I.

Because Copyfile and MLX use some of the same memory area on the 64, you'll have to enter a POKE statement in direct mode (without a line number) before you use MLX to type in Copyfile. This statement moves BASIC, and is:

```
POKE 44,PEEK(44)+2:POKE(PEEK(44))*256,0:NEW
```

If you enter Copyfile in more than one session (which is unlikely, since it's so short), you would have to type in the above statement each time before beginning to use MLX with this program.

Once you've typed in the POKE statement and loaded

MLX, RUN it. It will ask you for the beginning and ending addresses of Copyfile. They are:

Beginning address: 2049
Ending address: 2300

You can then begin to type in the numbers you see in the Copyfile listing at the end of this article. Once you've finished, SAVE it to disk or tape using the MLX program. Now you're ready to make copies of any file.

To use Copyfile, simply LOAD it and type RUN. With the proper disk in the drive, enter the name of the file to be copied. The filename should be in the form:

filename for PRG (program) file
filename, S for SEQ (sequential) file
filename, U for USR (user) file

You don't need to place the filename within quotation marks, as when you load a BASIC program. If you don't specify the type of file using a comma and appropriate letter, Copyfile by default will create it as a program file.

If there is any kind of error in reading the file into memory, an error message will display and the program will stop. If there is no error, the file will read into the computer. When the drive stops, remove the source disk and place the destination disk into the drive. Press the *C* key. The file then writes to the destination disk, using the original name of the file. An error at this point will show on the screen, and the program will wait for another press of the *C* key to try to read the file again, or a press of the RUN/STOP key to quit the program. If all's gone well, you've now got a copy of your original, ready to use.

You can even make multiple copies of the same program to different disks, simply by pressing the C key again (once another destination disk has been placed in the drive). Using this function, you can make as many copies of a file as you want. Pressing the RUN/STOP key at any time stops the process and lets you begin copying *another* file. Just type RUN, and you're ready to start again.

Copyfile

```
2049 :029,008,010,000,158,050,000
2055 :048,055,057,058,040,067,076
2061 :041,032,049,057,056,051,043
```

```
2067 :032,067,079,077,080,085,183
2073 :084,069,033,000,000,000,211
2079 :169,054,133,001,160,000,036
2085 :032,207,255,201,013,240,217
2091 :006,153,000,002,200,208,100
2097 :243,132,063,032,210,255,216
2103 :160,002,032,179,008,032,212
2109 :207,008,208,065,162,002,201
2115 :032,198,255,160,000,032,232
2121 :183,255,041,064,208,012,068
2127 :032,228,255,145,251,200,166
2133 :208,002,230,252,208,237,198
2139 :132,061,165,252,133,062,128
2145 :164,063,169,044,153,000,178
2151 :002,200,169,087,153,000,202
2157 :002,200,132,063,032,207,233
2163 :008,032,240,008,032,228,151
2169 :255,201,067,240,014,201,075
2175 :003,208,245,032,240,008,095
2181 :169,055,133,001,108,002,089
2187 :160,160,001,032,179,008,167
2193 :032,207,008,208,219,162,213
2199 :002,032,201,255,160,000,033
2205 :177,251,032,210,255,200,002
2211 :208,002,230,252,196,061,088
2217 :208,242,165,062,197,252,015
2223 :208,236,240,190,169,002,196
2229 :162,008,032,186,255,165,221
2235 :063,162,000,160,002,032,094
2241 :189,255,032,192,255,169,005
2247 :249,133,251,169,008,133,118
2253 :252,096,169,008,032,180,174
2259 :255,169,111,032,150,255,159
2265 :032,165,255,201,048,240,134
2271 :015,208,003,032,165,255,133
2277 :072,032,210,255,104,201,079
2283 :013,208,244,168,096,169,109
2289 :002,032,195,255,032,231,220
2295 :255,096,013,013,013,013,138
```

Merging Programs on the 64

John A. Winnie

For intermediate programmers, "Merger" allows you to build up large programs by working on smaller portions separately and then linking them together later. This approach is used by many professionals.

If you do much BASIC programming, sooner or later you'll need to merge two short programs to form a larger one. Or perhaps you'll need to append onto a program a series of DATA statements—DATA for sprites, redefined characters, sound and music, or whatever. Here is a quick and easy way to add those DATA statements—or any other BASIC statements—onto the end of your programs.

Of course, various techniques for merging programs have been around for some time. When all that is needed is a simple append, however, the method presented here does the job nicely. The program below, "Merger," is designed to merge with any programs which are appended to it, and it allows you to keep on appending indefinitely.

Using Merger

After typing and saving Merger, load it in the usual way. Next, run Merger, and then load in your main program. Now, as Merger instructs, POKE locations 43 and 44 with 1 and 8, respectively. Your main program is now appended to Merger and ready for any DATA statements you may want to add later.

Remember, Merger allows you to append programs only, not to insert them. So to prepare for using Merger later, begin your programs with a line number greater than five. For the same reason, all DATA statements to be added should begin with a line number higher than those already present in the

program. When you have finished, just erase Merger by deleting lines 1 through 5.

How Merger Works

First, clear out your Commodore 64 by typing NEW and pressing RETURN. Then enter the following simple program:

```
10 REM
```

Press RETURN, and the one-line program is now entered into memory beginning at address 2048 and running on upward. To see just how the program is stored, enter:

```
FOR I=2048 TO 2056:PRINT PEEK (I):NEXT I
```

If all this has been done correctly, you now should see a list of memory contents which looks like this:

```
0,7,8,10,0,143,0,0,0
```

The 0 in address 2048 is invariable: all BASIC programs begin with zero. They also always end with a zero; in fact, they always end with exactly three zeros—which is just what we see here in memory locations 2054 through 2056. From this point on in memory, BASIC will store any variables and other information that it may need to execute the program.

In general, when a BASIC line is stored, it will end with a single zero, not three zeros. When a new line is appended to the program, its code begins immediately after that single zero. So in the example above, if the line

```
20 REM
```

were now added to our sample program, the (link of the) new line would now come in at address 2055—the address of the middle zero in the triplet; a new triplet of zeros would appear later in memory, signaling the end of line 20 and the new end of the program. (Try this later to see for yourself.) So, to merge programs, we simply have to make sure that we load the new section at the address of the middle zero (2055, in our example) within the three zeros which signal the end of our original program. What we need to do is raise the floor of BASIC to this new address, load the section to be merged, and then lower the floor to its original value (here, 2049).

Tinkering with BASIC

Raising the floor of BASIC is easy. The new address is simply POKEd into addresses 43 and 44 in low-byte, high-byte order.

(HI=INT (ADDRESS#/256) :LO=ADDRESS#-256*HI.) Finding this new address is another matter, but fortunately, this turns out to be easy as well.

As I mentioned above, BASIC needs to know where it is safe to begin to store its variables. In other words, BASIC needs to know the first address to come after the three zeros which end the program. Hence, the computer stores this address in a pair of memory locations in the usual low-byte, high-byte form. In the 64, these locations are addresses 45 and 46.

To see this, enter PRINT PEEK(45),PEEK(46), and out should come the pair 9,8. Since the address 2057 is the first address to follow our sample program, and 2057=256*8+9, we have the expected result.

Now that we have the address of the first location after the end of the program, the rest is easy. The new program is simply loaded into memory two places before this location. In our example, we load at location 2055 (2057–2). And that's all there is to it.

The basic idea behind Merger should now be clear. Everything of interest is packed into line 5. First, for any program which begins with these lines, the new floor for BASIC is computed using the contents of locations 45 and 46, as described above. Next, the floor of BASIC is raised to the new location. As a result, any new program now loaded will start right at the tail end of the previous program—just where we want it.

64 Merger

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

```
1 C=53280:POKEC,6:POKEC+1,8                          :rem 45
2 INPUT "{BLU}{CLR}{4 DOWN}{8 RIGHT}MERGE (Y/N)";A
  $                                                  :rem 27
3 IFA$<>"Y"THENEND                                   :rem 68
4 PRINT"{10 DOWN}{3 RIGHT}LOAD YOUR ADDITION":PRIN
  T"{3 RIGHT}THEN POKE 43,1 AND 44,8."               :rem 18
5 E=256*PEEK(46)+PEEK(45)-2:H=INT(E/256):L=E-256*H
  :POKE43,L:POKE44,H:END                            :rem 181
```

64 Program Lifesaver
"UNNEW" Rescues Lost Programs

Vern Buis

If you have ever lost a BASIC program by accidentally typing NEW, then read on. This short machine language routine for the Commodore 64 provides an easy means of recovering BASIC programs that have been "erased"—and it loads and executes in only ten seconds.

Sooner or later—practically every programmer does it—thinking a program has been saved, you type NEW to clear out the memory, and a split-second after pressing RETURN, you wind up screaming.

But on the Commodore 64, typing NEW does not really erase the program from memory. NEW just makes the computer (and the programmer) *think* the program is gone. As long as you don't start typing another program or switch off the machine, the program is still there. To get it back, all you have to do is fool the computer into remembering where in its memory the program begins and ends.

That's what "64 Program Lifesaver" does. By loading and running this short machine language utility immediately after committing the grievous error, you can save your lost program, save your hours of work, and even save your sanity.

Entering the Lifesaver

The Lifesaver is listed as a BASIC loader, a BASIC program that creates a machine language program. Be sure to read the following special instructions before typing the program. The procedure is somewhat different from most and requires that certain steps be followed exactly.

First, if you are using tape instead of disk, enter line 60 as follows:

```
60 CLR:SAVE"UNNEW",1,1
```

After typing the listing, *do not RUN it.* Instead, save it on disk or tape with a filename such as "LIFESAVER/BASIC" or "UNNEW/BASIC". Do not use the filename "UNNEW". This filename must be reserved.

Now enter RUN. The BASIC loader creates the machine language program and automatically saves it on disk or tape under the filename "UNNEW". This is what you'll actually use to rescue lost programs; the BASIC loader can be set aside as a backup in case you need to create another copy.

Using the Lifesaver

OK, let's say you've just typed NEW and wiped out hours of valuable labor. (To test the Lifesaver, you can load a BASIC program and erase it with NEW.) Recovering it is easy.

To load the Lifesaver from tape, enter:

```
LOAD"UNNEW",1,1
```

To load the Lifesaver from disk, enter:

```
LOAD"UNNEW",8,1
```

Either way, it loads pretty fast, because the program is short. Now, to activate the Lifesaver, enter:

```
SYS 525 [RETURN]
CLR [RETURN]
```

(Incidentally, CLR means to type the keyword CLR, not to press the CLR/HOME key.)

That's all there is to it. When you enter LIST, the BASIC program you thought was forever lost is back, safe and sound.

The Lifesaver itself also remains in memory, but probably not for long. It's tucked away in memory which is unprotected (locations used by the input buffer and BASIC interpreter), so you'll have to load it again each time you want to use it. But unless you're either very unlucky or (shall we say) prone to inadvertent actions, the Lifesaver isn't something you should be needing often.

Why It Works

Instead of erasing the program in memory when you type NEW, the 64 simply resets two key pointers in such a way that the operating system doesn't "see" that the program is still there. These pointers keep track of where in memory a

BASIC program begins and ends. NEW moves the top-of-program pointer down to the bottom of BASIC memory, and the first two bytes of BASIC memory are set to zero. These first two bytes serve as a pointer to the address for the second line of BASIC code. When they are set to zero, the operating system believes that no program is in memory.

The Lifesaver works by skipping the first two bytes of BASIC memory (the address pointer) and the next two bytes (BASIC line number). It scans upward for a zero byte—the end-of-line indicator. Upon finding the zero byte, the routine POKEs its address, plus one, into the second-line-of-BASIC address pointer. One of the erased pointers is thereby restored.

Next, the Lifesaver scans byte-by-byte through the BASIC memory area until it finds three consecutive zero bytes. This is the end-of-program indicator. Once it locates these zeros, the routine POKEs the address of the third zero, plus one, into the top-of-BASIC/start-of-variables pointer at locations 45–46. This completely restores the erased program.

For those who might want to relocate the Lifesaver to a safer memory area—to preserve it for frequent use or to combine it with other utility routines—the machine language program is written to be fully relocatable. It uses no absolute JMP or JSR instructions. The area used here was chosen to make it load easily into a 64 and to minimize the danger of it loading atop a BASIC program.

64 Program Lifesaver

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

```
10 I=525                                          :rem 131
20 READ A:IF A=256 THEN 40                         :rem 54
30 POKE I,A:I=I+1:GOTO 20                         :rem 130
40 POKE 43,525 AND 255:POKE 44,2:REM SET BOTTOM OF
    MEMORY                                        :rem 173
50 POKE 45,578 AND 255:POKE 46,2:REM SET TOP OF ME
   MORY                                           :rem 216
60 CLR : SAVE"0:UNNEW",8                           :rem 79
70 REM FOR TAPE USE SAVE"UNNEW",1,1                 :rem 3
525 DATA 160,3,200,177,43,208,251                  :rem 82
532 DATA 200,200,152,160,0,145,43                  :rem 64
539 DATA 165,44,200,145,43,133,60                  :rem 87
546 DATA 160,0,132,59,162,0,200                   :rem 231
553 DATA 208,2,230,60,177,59,208                   :rem 45
560 DATA 245,232,224,3,208,242,200                :rem 126
567 DATA 208,2,230,60,132,45,164                   :rem 37
574 DATA 60,132,46,96,256                         :rem 220
```

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.

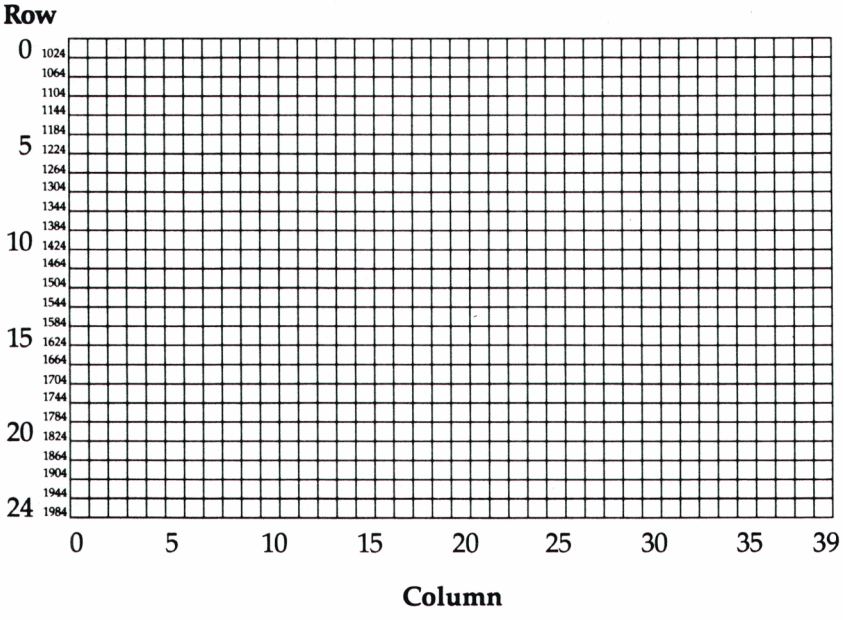
You also go into quote mode when you INSerT spaces into a line. In any case, the easiest way to get out of quote mode is to just press RETURN. You'll then be out of quote mode and you can cursor up to the mistyped line and fix it.

Use the following table when entering cursor and color control keys:

When You Read:	Press:	See:
{CLR}	SHIFT + CLR/HOME	
{HOME}	CLR/HOME	
{UP}	SHIFT + CRSR ↑	
{DOWN}	CRSR ↓	
{LEFT}	SHIFT + CRSR ←	
{RIGHT}	CRSR →	
{RVS}	CTRL + 9	
{OFF}	CTRL + 0	
{BLK}	CTRL + 1	
{WHT}	CTRL + 2	
{RED}	CTRL + 3	
{CYN}	CTRL + 4	
{PUR}	CTRL + 5	
{GRN}	CTRL + 6	
{BLU}	CTRL + 7	
{YEL}	CTRL + 8	

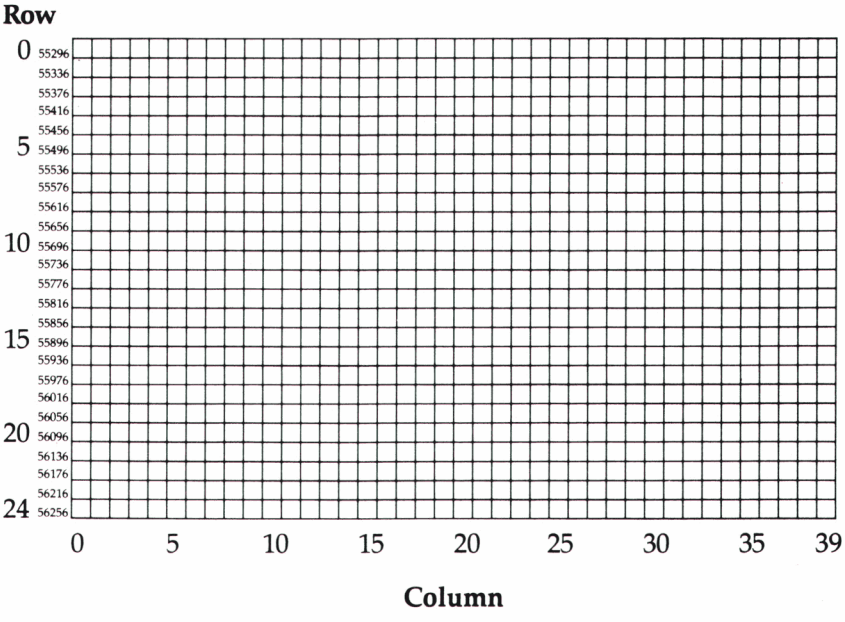
When You Read:	Press:	See:
{1}	COMMODORE + 1	
{2}	COMMODORE + 2	
{3}	COMMODORE + 3	
{4}	COMMODORE + 4	
{5}	COMMODORE + 5	
{6}	COMMODORE + 6	
{7}	COMMODORE + 7	
{8}	COMMODORE + 8	
{F1}	f1	
{F2}	SHIFT + f1	
{F3}	f3	
{F4}	SHIFT + f3	
{F5}	f5	
{F6}	SHIFT + f5	
{F7}	f7	
{F8}	SHIFT + f7	

Screen Location Table



Appendix D

Screen Color Memory Table



Screen Color Codes

Value To POKE For Each Color

Color	Low nybble color value	High nybble color value	Select multicolor color value
Black	0	0	8
White	1	16	9
Red	2	32	10
Cyan	3	48	11
Purple	4	64	12
Green	5	80	13
Blue	6	96	14
Yellow	7	112	15
Orange	8	128	—
Brown	9	144	—
Light Red	10	160	—
Dark Gray	11	176	—
Medium Gray	12	192	—
Light Green	13	208	—
Light Blue	14	224	—
Light Gray	15	240	—

Where To POKE Color Values For Each Mode

Mode*	Bit or bit-pair	Location	Color value
Regular text	0	53281	Low nybble
	1	Color memory	Low nybble
Multicolor	00	53281	Low nybble
text	01	53282	Low nybble
	10	53283	Low nybble
	11	Color memory	Select multicolor
Extended	00	53281	Low nybble
color text †	01	53282	Low nybble
	10	53283	Low nybble
	11	53284	Low nybble
Bitmapped	0	Screen memory	Low nybble [‡]
	1	Screen memory	High nybble [‡]

Multicolor	00	53281	Low nybble [‡]
bitmapped	01	Screen memory	High nybble [‡]
	10	Screen memory	Low nybble [‡]
	11	Color memory	Low nybble

* For all modes, the screen border color is controlled by POKEing location 53280 with the low nybble color value.

† In extended color mode, Bits 6 and 7 of each byte of screen memory serve as the bit-pair controlling background color. Because only Bits 0–5 are available for character selection, only characters with screen codes 0–63 can be used in this mode.

[‡] In the bitmapped modes, the high and low nybble color values are ORed together and POKEd into the *same location* in screen memory to control the colors of the corresponding *cell* in the bitmap. For example, to control the colors of cell 0 of the bitmap, OR the high and low nybble values and POKE the result into location 0 of screen memory.

ASCII Codes

ASCII	CHARACTER	ASCII	CHARACTER
5	WHITE	50	2
8	DISABLE	51	3
	SHIFT COMMODORE	52	4
9	ENABLE	53	5
	SHIFT COMMODORE	54	6
13	RETURN	55	7
14	LOWERCASE	56	8
17	CURSOR DOWN	57	9
18	REVERSE VIDEO ON	58	:
19	HOME	59	;
20	DELETE	60	<
28	RED	61	=
29	CURSOR RIGHT	62	>
30	GREEN	63	?
31	BLUE	64	@
32	SPACE	65	A
33	!	66	B
34	"	67	C
35	#	68	D
36	$	69	E
37	%	70	F
38	&	71	G
39	'	72	H
40	(	73	I
41	)	74	J
42	*	75	K
43	+	76	L
44	,	77	M
45	–	78	N
46	.	79	O
47	/	80	P
48	0	81	Q
49	1	82	R

Appendix F

ASCII	CHARACTER	ASCII	CHARACTER
83	S	120	
84	T	121	
85	U	122	
86	V	123	
87	W	124	
88	X	125	
89	Y	126	
90	Z	127	
91	[	129	ORANGE
92	£	133	f1
93	]	134	f3
94	↑	135	f5
95	←	136	f7
96		137	f2
97		138	f4
98		139	f6
99		140	f8
100		141	SHIFTED RETURN
101		142	UPPERCASE
102		144	BLACK
103		145	CURSOR UP
104		146	REVERSE VIDEO OFF
105		147	CLEAR SCREEN
106		148	INSERT
107		149	BROWN
108		150	LIGHT RED
109		151	GRAY 1
110		152	GRAY 2
111		153	LIGHT GREEN
112		154	LIGHT BLUE
113		155	GRAY 3
114		156	PURPLE
115		157	CURSOR LEFT
116		158	YELLOW
117		159	CYAN
118		160	SHIFTED SPACE
119		161	

COMPUTE!'s Second Book of Commodore 64

COMPUTE!'s Second Book of Commodore 64 continues the popular tradition of presenting some of the best articles and programs from *COMPUTE!* magazine and *COMPUTE!'s Gazette*, as well as several others which have never before appeared in print.

Like the best-selling *COMPUTE!'s First Book of Commodore 64*, this book includes dozens of complete, ready to type in programs. There are even utilities which make your typing of BASIC and machine language programs mistake-proof.

Some of the articles and programs included are:

- "Speedscript," a machine language word processor of commercial software quality.
- Educational games for children which are fun to play, yet teach skills from spelling to mathematics.
- A program which adds 41 new commands to BASIC.
- A detailed explanation of the computer's SID sound chip.
- How to add your own new BASIC keywords.
- Arcade-style games, from the fast-paced, all-machine-language "Spike," to a text adventure game where you explore an alien spaceship.
- A sprite editor.
- Financial applications, including an electronic spreadsheet program and a program to keep track of your files or mailing lists.
- Programs which make mistake-free entry of machine language and BASIC programs simple and easy.

If you've purchased a COMPUTE! Book before, you already know about the high quality programs and clear explanations. If this is your first COMPUTE! Book, you're in for some pleasant surprises.

1284
NR13423
239,25

ISBN 0-942386-44-2



This was brought to you

from the archives of

http://retro-commodore.eu