A **COMPUTE! Books** Publication $12.95

VIC 3

COMPUTE!'s Third Book of VIC

Games, utilities, tutorials, applications, and more for users of the VIC-20 home computer.

COMPUTE!'s Third Book of VIC

VIC 3

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

The following article was originally published in *COMPUTE!* magazine, copyright 1982, Small System Services, Inc.: "VIC's Perpetual Calendar"(June).

The following articles were originally published in *COMPUTE!* magazine, copyright 1983, Small System Services, Inc.: "Automatic VIC Appending" (March), "Musical Scales on the VIC" (March), "Major & Minor: VIC Music Theory" (April), "Programming Multicolor Characters on the VIC" (May).

The following articles were originally published in *COMPUTE!* magazine, copyright 1983, COMPUTE! Publications, Inc.: "SpeedSki" (July), "VIC Musician" (July), "Relocating VIC Loads" (August), "Ultrasort for Commodore" (September), "Spiralizer" (October), "PEEK and PRINT for the VIC-20" (November), "Art Museum" (December).

The following articles were originally published in *COMPUTE!'s Gazette*, copyright 1983, COMPUTE! Publications, Inc.: "Quickfind" (July), "VIC/64 Mailing List" (August), "Disk Menu" (August), "Demon Star for VIC and 64" (September), "Using the Function Keys: A BASIC Tutorial" (September), "How to Use Tape and Disk Files" (October), "Introduction to Custom Characters for VIC and 64" (November), "How to Make Custom Characters on the VIC" (November), "VIC/64 Program Lifesaver" (November), "Tricks for Saving Memory" (December), "Custom Characters on the Expanded VIC" (December), "VIC Music Writer" (December).

The following articles were originally published in *COMPUTE!'s Gazette*, copyright 1984, COMPUTE! Publications, Inc.: "SpeedScript" (January), "Graph Plotter" (January).

The following article was originally published in *COMPUTE!* magazine, copyright 1983, Jim Butterfield: "Visiting the VIC-20 Video, Parts 1–4."

Printed in the United States of America

ISBN 0-942386-43-4

10 9 8 7 6 5 4 3 2 1

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. VIC-20 is a registered trademark of Commodore Electronics Limited.

Contents

Chapter 1

Getting to Know Your VIC

How to Use Tape and Disk Files

Richard Mansfield

In the past, programming the use of files often seemed too difficult. It is, though, an excellent tool that all programmers can learn to use.

After a few weeks with their computers, many people find themselves trying (and failing) to make *files* on tape or disk. Files are quite useful, even necessary in many kinds of programs, but you do have to be a bit patient with them at first. They're not as immediately obvious as other aspects of BASIC.

We'll take it step by step, and you'll soon have files going in and out of the computer like a pro. But before getting down to specifics, a brief historical note will demonstrate that any confusion you might have experienced when working with files is fully justified and puts you in good company. In the early days, files mystified nearly everyone.

Charming, But Slim

In 1978, the first true consumer computers—the venerable model 2001 Commodore PETs—were shipped with a charming, but slim, user's manual. This booklet, 49 pages long, was called *An Introduction to Your New PET (Revised).* It included instructions on using the reverse field key, the cursor controls, and some elementary BASIC, along with tips on how to clean the PET. Nothing about files.

Also, there were very few books or magazines about personal computers at that time. And they said nothing useful about files either. The best sources of specific information were the few mimeographed user-group newsletters. These early publications were full of techniques and debates about how to make files work.

To get a firm grip on OPEN, CLOSE, PRINT#, and

INPUT# (BASIC's file-handling words), the first thing we should do is clearly understand the general differences between programs and files.

Telling Them Apart

Tapes or disks can store two entirely different things—*programs* and *files*. (Don't be confused if you should read something like this in a book: "Store your program files on tape." That terminology is both redundant and confusing. There is a crucial distinction to be made between programs and files.)

A BASIC program is a collection of lines, and each line contains instructions to the computer. These instructions are to be carried out during a RUN of the program. That is, the instructions are followed in order, from the lowest line number to the highest, after you type the word RUN. A data file, by contrast, is raw information, like a page in a telephone book, without any instructions about what to do with that information.

When programs are saved onto a disk or tape, they can later be loaded back into the computer to be run any time in the future. Programs you type into the computer will stay there only as long as the computer is turned on. So, to build a library of programs, you must save them on tape or disk. (Let's refer to tape or disk storage as *magnetic memory* from here on.)

When programs are saved to magnetic memory, it's as if the tape or disk were given a photo of the program that was in the computer at the time of the SAVE. BASIC keeps track of how large a program is—where it starts and ends in the computer's memory cells—so it knows just what to "photograph" when you ask for a SAVE.

BASIC doesn't help you out this way with your files; BASIC doesn't thoroughly supervise file storage and recall. You must do several things to create a file on magnetic memory and several things to get it back into the computer later. You establish the size of the file, the divisions between items in the file (called *delimiters*), and the order of the items. We'll illustrate this in a minute, but first let's visualize how programs and files differ:

A typical can of soup will have both a *file* and a *program* written on the label:

Tomato Soup
Water, tomatoes, salt,
monosodium glutamate,
red color #7, oleoresin.

1. Open carefully.
2. Empty contents into pan.
3. Add one can of water.
4. Heat to a simmer.

Steps 1 through 4 are clearly a program of sorts. One clue that these steps are program-like is that each item starts with a number, indicating the order in which the operations are to be performed. The ingredients—standing by themselves as raw data—are a file. And, just as the ingredients "file" above is *acted upon* by the cooking instructions program, a computer program acts upon a data file.

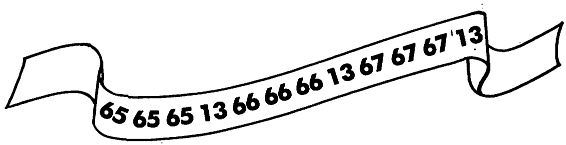
Here's a simple program which will create a tape file (there's a different format if you want to make a disk file: You change the device number in line 20 by typing OPEN 1,8,8,"0:FILE,S,W"—we'll get to the reason for the "0:" and the "S,W" later):

```
10 DATA AAA,BBB,CCC
20 OPEN 1,1,1,"FILE"
30 FOR I=1 TO 3
40 READ D$
50 PRINT#1,D$
60 NEXT I
70 CLOSE 1
```

PRINT# (usually pronounced "print-number") is an entirely different command from PRINT, and the punctuation, as always in programming, must be exact. Line 40 is interesting because we keep READing D$ over and over to use it as a temporary holding place until we can PRINT# to a magnetic memory. D$ isn't anything in itself (it varies, it's a *variable*). READ will pick out each datum from the DATA in line 10 in turn, keeping track of the last one that was READ.

In any case, after this program is run, the magnetic memory would contain a file. If we could look at that file on the tape the way we would look at a photograph, we would see a row of numbers. The number 65 stands for the letter A, and 13 represents a carriage return. Here's what the photograph would look like:

A Strip of Tape Containing Our Example File



A *data base manager* is a program that manipulates data in files. Writing a large, flexible data base managing program is not a simple task—it can involve sorting, searching, and other complex programming techniques. Nonetheless, handling Christmas card lists is not beyond beginning programming skills. Files do, though, represent something of a challenge. Your computer's manual contains information on the necessary punctuation and syntax for the BASIC commands which manipulate files. However, a brief overview might be of help.

OPEN, PRINT#, INPUT#, and CLOSE

While a program would be stored by the simple SAVE instruction, a file is stored by a combination of OPEN, PRINT#, and CLOSE. Likewise, a program is just LOADed, but a file is "loaded" into the computer with OPEN, INPUT#, and CLOSE. Files *are* a bit more complicated, but the tradeoff is that you can do more manipulating with files, easier *appending* (adding to them), easier *merging* (making two files into one), and so on.

The command OPEN is generally used to communicate with a disk or tape drive. It's like pulling open a file cabinet drawer—once a file is OPENed, you can then get at the records inside. Here's what you would do to OPEN the file we created earlier. This time, instead of writing to it, we'll read from it.

OPENing Commodore Files

```
10 OPEN 1,8,8,"0:FILE,S,R"
```

The first number (1) means that this OPEN will hereafter be called #1. When you pull something out of it, you would use INPUT#1 (you can keep open up to ten files at one time). The second number (8) means disk drive (a 1 in this position would mean to open a file on the cassette drive). The second 8 is a secondary address which allows you to give additional instructions. With disk drives, just use 8.

The 0: specifies drive zero and the S means *sequential* file. The Commodore disks can create two other kinds of files, random and relative, but sequential is the simplest. Finally, the R means *read*, so you will be using INPUT# to get things out of this file. (A *W* here would mean *write*, and you would PRINT# to the file.) To make this "reading or writing" distinction for tape files, the secondary address is used: A 1 means write and a 0 means read. (10 OPEN 1,1,0,"FILE" would be the same as the example above, except it's for a cassette file. No drive number is specified, and the S is not necessary since cassette files can only be sequential files.)

Taking Something Out

Recall that we put things into a file with PRINT#. Going the other way, you get something out of an OPENed file by using INPUT# in combination with a string variable to "hold" whatever comes from the file (they come back to the computer in the order they were PRINT#ed in a sequential file). To get the AAA back:

```
20 INPUT#1,A$
```

Then you could print A$ to see the data AAA it holds:

```
30 PRINT A$
```

To get the rest of the data we stored, you could:

```
40 INPUT#1,B$:PRINT B$
50 INPUT#1,C$:PRINT C$
```

After you finished INPUT#ing or PRINT#ing from a file which had been OPENed as file #1, you would close it:

```
60 CLOSE1
```

When you've finally CLOSEd the file, you are free to use that file number (#1 in these examples) for some other file, with a different name. CLOSE is essential, however. Without it you could permanently lose part or all of a file, or even damage other files. Don't leave files open after you're finished with them.

INPUT# and PRINT# Hints

The INPUT# and PRINT# work very similarly to the way INPUT and PRINT work from the keyboard and to the screen. The only catch is that PRINT# needs some special handling. It's best to give it a line to itself:

```
20 PRINT#1,A$
30 PRINT#1,B$
```

The reason for putting PRINT# on its own line is that this is an easy way to separate items in a file: with carriage returns. Just as

```
20 PRINT A$
30 PRINT B$
```

causes B$ to be on the line below A$ on the screen (since using a new line forces a carriage return to take place)—a separate program line puts a carriage return symbol onto the tape or disk and keeps the data from running into each other. This kind of "separator" symbol is called a *delimiter.*

Manipulating Files

Our example program above, which reads data from magnetic memory into the computer, does work, but it's cumbersome. Files are usually written to or read from within loops. Here's a simple program to write a file to tape:

```
10 DATA BILL, SANDY, DORIS, LARRY
20 OPEN 1,1,1,"NAMES":REM (A TAPE FILE)
25 PRINT#1,4
30 FOR I=1 TO 4
40 READ A$
50 PRINT#1,A$
60 NEXT I
70 CLOSE 1
```

Since there are four names in this file, the loop counts up to four, READing a new A$ from the DATA line each time through. Since PRINT#1 is by itself on line 50, it will send carriage returns to the tape each time it PRINT#s, separating the names on tape with delimiters. This way, there will be no question of something like BILLSANDY being stored.

When this file is later read into the computer, it would be very useful to know when the entire file ends, how big it is. There are two ways to do this. You could add the word "END" to the DATA line and then change line 30 to read: FOR I=1

TO 5. Or, you could put the *count* (the number of records for this file) on the tape or disk itself, as part of the file. See line 25 above for an example of this.

Here's a *reader* program which first pulls the count out of the file and then brings the records into the computer.

```
10 OPEN 1,1,0,"NAMES":REM (A TAPE FILE)
20 INPUT#1,COUNT:REM THIS IS THE FIRST ITEM ON THE
    FILE
30 FOR I=1 TO COUNT
40 INPUT#1,A$
50 PRINT A$:REM (TO THE SCREEN)
60 NEXT I
70 CLOSE 1
```

If you use the "END" technique, the reader program would not use line 20 and would add line: 45 IF A$="END" THEN GOTO 70. If you plan to do significant manipulations with the file data, you might want to call the items into an array so that they can be worked with easily.

One final note about something which might not be immediately obvious: If you update a file, you cannot put it back on a disk using the same name. Here's why: It's first read off the disk and into memory because you want to make some changes. Before you OPEN–PRINT#–CLOSE it back onto the disk, you must first *scratch* (remove) the original file (it's in the computer now) so you can replace it with the updated one. For obvious reasons, you can't have two files on disk with the same name. This *scratching* is unnecessary for tape files, because the recorder will write over the old file (if you rewind the tape).

There are numerous ways to manipulate files. We've been dealing with *sequential* files, the most straightforward type of files; using INPUT#, the most straightforward access command; and delimiting with carriage returns, the simplest punctuation. Your manual contains information about more complex, sophisticated filing techniques, including special types of disk files, using GET#, and delimiting with commas or even using semicolons between items.

Using the Function Keys

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 (you held down a special ConTRoL key while pressing 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 labeled keys to cut, paste, copy, edit, etc. Since computers are general-purpose, the keys had to be unlabeled 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, unlabeled function keys are defined by whatever program is currently running. Frequently, programmers assign powerful functions to the keys. This gives the user a feeling of power—pressing one key unleashes raw computing power. Of course, 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 the spaceship based on the position of the joystick.

When you run commercial software, the keys do everything from changing border colors to shifting the screen, selecting difficulty, restarting a game, etc. If you buy the Super Expander cartridge, the keys will type out certain BASIC commands for you. But the real power 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 IFA$="" 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 less than that of 0 or greater than that of the character 9, then loop back to line 10. (The ASCII value is 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.)

Incidentally, the ASCII code for the null string (quote-quote) is zero, which is less than 48, the code for the numeral zero, so the loop will also wait for a key. If you're curious about ASCII, check out the BASIC commands ASC and CHR$ in the appendix. You can also find a table of the ASCII codes and their character equivalents in your user's guide.

Strictly Logical?

So if you just want to accept a yes or no answer (Y for yes, N for no), 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 <>, <, >, =, etc.

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$="{F5}" THEN PRINT"OOO! FUNCTION FIVE"
70 IF F$="{F6}" THEN PRINT"FUNCTION SIX"
80 IF F$="{F7}" THEN PRINT"FUNCTION SEVEN"
90 IF F$="{F8}" THEN PRINT"FUNCTION EIGHT"
```

The {F1}, {F2}, etc., mean for you to press the appropriate function key inside the quotes. You'll get the aforementioned symbols. Line 60 is just to remind you that every program has a spark of spontaneity. 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$(34);A$,VAL(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 the symbol for it 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 to f7, and f2 to f8, separately. So you could use a statement like this to check for f6:

```
342 IF F$=CHR$(139) THEN PRINT"FUNCTION SIX"
```

or

```
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 it do it.

Tricks for Saving Memory

John Stilwell

Writing programs to fit in an unexpanded VIC-20 is not easy—there's only 3.5K of free memory to work with. You should find the following tricks very useful.

Trick 1

Always use keyword abbreviations when entering a program (see the table below for a list of abbreviations). This won't save any memory because of the abbreviations, but it will allow you to cram more statements into a line. This is important because every line takes up five bytes, then you start counting the statements. The only problem with this trick is that if the line, when listed, exceeds 88 characters on the VIC, you can't edit it. If something needs to be changed, you will have to retype the entire line. Also, if you submit the program to a magazine which publishes the listing, other people won't be able to enter your program without also using the abbreviations—something they may not know.

VIC Keyword Abbreviations

A	SHIFT-B for ABS
A	SHIFT-N for AND
A	SHIFT-T for ATN
C	SHIFT-H for CHR$
CL	SHIFT-O for CLOSE
C	SHIFT-L for CLR
C	SHIFT-M for CMD
C	SHIFT-O for CONT
D	SHIFT-A for DATA
D	SHIFT-E for DEF
D	SHIFT-I for DIM
E	SHIFT-N for END
E	SHIFT-X for EXP

F	SHIFT-O for FOR
F	SHIFT-R for FRE
G	SHIFT-E for GET
GO	SHIFT-S for GOSUB
G	SHIFT-O for GOTO
I	SHIFT-N for INPUT#
LE	SHIFT-F for LEFT$
L	SHIFT-E for LET
L	SHIFT-I for LIST
L	SHIFT-O for LOAD
M	SHIFT-I for MID$
N	SHIFT-E for NEXT
N	SHIFT-O for NOT
O	SHIFT-P for OPEN
P	SHIFT-E for PEEK
P	SHIFT-O for POKE
?	for PRINT
P	SHIFT-R for PRINT#
R	SHIFT-E for READ
RE	SHIFT-S for RESTORE
RE	SHIFT-T for RETURN
R	SHIFT-I for RIGHT$
R	SHIFT-N for RND
R	SHIFT-U for RUN
S	SHIFT-A for SAVE
S	SHIFT-G for SGN
S	SHIFT-I for SIN
S	SHIFT-P for SPC(
S	SHIFT-Q for SQR
ST	SHIFT-R for STR$
ST	SHIFT-E for STEP
S	SHIFT-T for STOP
S	SHIFT-Y for SYS
T	SHIFT-A for TAB(
T	SHIFT-H for THEN
U	SHIFT-S for USR
V	SHIFT-A for VAL
V	SHIFT-E for VERIFY
W	SHIFT-A for WAIT

Trick 2

If the last character on a logical line is an ending quotation mark of a PRINT statement, leave it off. It won't hurt anything as long as it's the last thing on the line. Besides less typing for you, it saves one byte for each quote you leave off. This may not seem like much, but everything adds up. Remember, the average line statement is 40 bytes long.

Trick 3

This one will save the greatest amount of memory. Use cursor controls whenever possible. Here are some examples:

```
10 PRINT
20 PRINT
30 PRINT
40 PRINT"HI MOM"
```

This program uses 34 bytes of memory. If the PRINT statements are replaced by down-cursor controls, there is a significant saving.

```
10 PRINT"{3 DOWN}HI MOM"
```

This accomplishes the same thing but uses 19 bytes, so we saved 15 bytes. Now we're getting somewhere. Look through your program and see how many times you can do this. You may be amazed. Oh, don't forget to leave the ending quotation mark off.

```
10 PRINT "{3 DOWN}HI MOM
```

This saves one extra byte.

Trick 4

This is a modification of Trick 3. Always use TABs instead of cursor controls if there are a lot of cursor controls. However, with TABs you are limited to moving from left to right and down.

To move to the right five columns, use TAB(5). To move down, add 22 for every row. For example, we will move to the right five columns and down ten rows:

(10 rows*22)+5 columns=225, so use TAB(225).

Unfortunately, the TAB number must be less than 256. If you need to TAB further than 255, it is legal to stack TABs—TAB(255)TAB(25).

Instead of this:

10 PRINT "{10 DOWN} {5 SPACES}HI MOM"

Memory usage is 31 bytes. Try it this way:

(10 rows*22)+5 columns=225

10 PRINT TAB(225)"HI MOM"

This now uses only 22 bytes. In comparison to Trick 3, nine bytes may not seem like much, but if the above program were written with ten PRINT statements, it would use approximately 77 bytes. So we would have saved 56 bytes by using TABs.

To know when to use TABs instead of cursor controls, you must look at the memory requirements. Cursor controls take one byte each. TABs take two bytes plus one byte for each digit in the TAB number.

Trick 5

If something looks strange with the TAB above, you are right. There is no semicolon between the TAB and the quote. It is not necessary. Since it doesn't affect the spacing, why use it? After all, it uses up one byte. You can also eliminate the semicolon between quotes and variables.

10 PRINT "A=";A

can be written as

10 PRINT"A="A

Note that the semicolon must be used if the PRINT is changed to an INPUT.

10 INPUT"A=";A

Trick 6

This trick is frowned upon by traditional programmers. Nevertheless, you can number a program by ones. You won't want to do this unless you have a renumber program. If you renumber the program by ones, starting with zero as the first line number, the program will take up less space. This is because all branching commands such as GOTO take one byte plus one for every digit of the address.

This trick has on occasion saved me a couple of hundred bytes. Unfortunately, modifying this program will be hard, since you can't insert any lines without renumbering.

Trick 7

Trick 7 does not hold for most computers. With the VIC, use PRINT statements instead of POKEs whenever possible. This is for three reasons.

First of all, POKE statements are very slow. I recently rewrote the graphics in a program, changing the POKEs into PRINT statements. I was amazed. You would think that it was written in machine language. The speed difference is that great.

Second, POKE statements take up more memory than PRINT statements (in most cases). A POKE takes two bytes plus one for every digit of the numbers that go with it. That's an average of eight bytes for every character POKEd on the screen. In contrast, it takes one byte for the PRINT and one for each of the quotes and characters in between. So, if you are creating graphics, you might save a lot of memory by using PRINT statements.

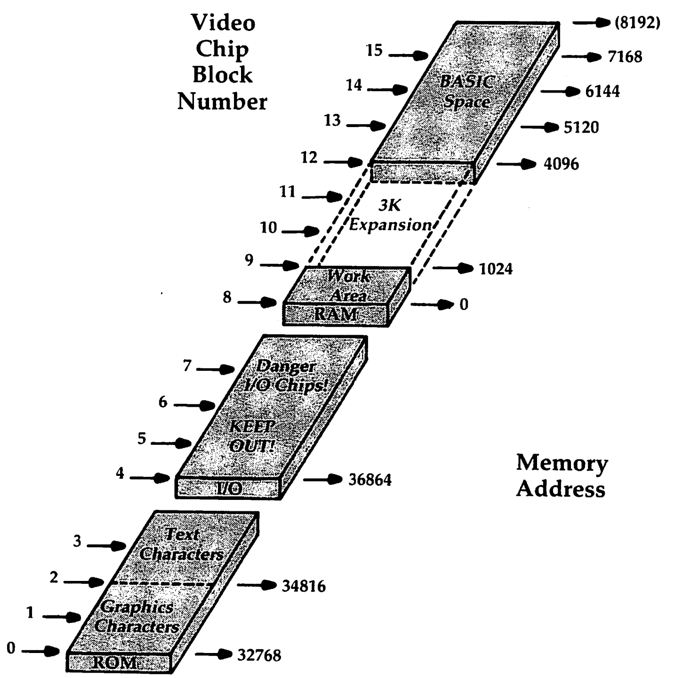
Third, when POKEing directly into screen memory on the VIC, a corresponding POKE to color memory is necessary to make the character appear on the screen. This then requires two POKE statements for each character. It will be more economical (memory-wise) to use PRINT, which automatically takes care of color memory.

Visiting the VIC-20 Video

Jim Butterfield

By traveling through the video chip and looking at things from the chip's point of view, you'll learn more about the structure and use of the VIC-20 video chip.

If we want to put the VIC-20 video chip to work, we must learn to see things from its standpoint. It sees the computer memory in a way that differs significantly from the way the processor chip sees it. Let's look at what the video chip sees:



How the video chip sees memory.

How the Video Chip Sees Memory

The video chip sees only the memory shown above. Even if you have expanded your computer to include lots of extra RAM above address 8191, the chip can't see it. The chip sees only the character ROM, in blocks 0, 1, 2, and 3; and the lowest 8K of RAM (in blocks 8 to 15). Blocks 4, 5, 6, and 7 would look at the Input/Output area, but take my advice: Don't do it—no good will come from these addresses.

What the Chip Wants

The video chip wants to dig out two things from memory and deliver them to the screen. It wants to look at screen memory—usually the characters you have typed. On a minimum 5K VIC, that's block 15.5, which corresponds to decimal address 7680 or hexadecimal 1E00. Did I mention that for screen memory, we can look at half blocks? It makes sense, since only 500-odd characters are needed to fill the screen.

By the way, the official name for screen memory is the *video matrix.* Whatever you call it, if you POKE 7680,1 on an unexpanded VIC, you'll see the letter A appear at the start of the screen. Unless, of course, you're printing white on white, in which case you need very good vision to see it.

The second thing that the chip wants from memory is the *character set*—instructions on how to draw each character on the screen. On a typical VIC, this will be either block 0 for the graphics character set or block 2 for the text mode (upper- and lowercase). You can change it, but you'll usually want to stay with even numbers: A full character set including the reversed characters takes up 2048 bytes of memory.

The official name for the character set is *character cells,* although the term *character base* is coming into use. Whatever you call it, you can't POKE 32768,55 and expect anything to happen—the standard characters are in ROM and cannot be changed. They're carved in stone, or silicon, to be more exact. If you want to switch to custom characters, you'll need to stage them in RAM and tell the chip which block to take them from.

There's a third thing that the chip uses, but it doesn't come from regular memory in the usual way. That's the screen colors (the *color matrix).* This color information for each character comes through the back door, so to speak, and we won't worry about the details too much here. When we need to, we'll set the color and assume everything will work.

Architecture

Looking at the diagram, we can begin to see why the VIC does its odd screen switch when you add memory. In the 5K VIC, the screen sits at the top of memory—and that's the highest address that the video chip can see (block 15.5). If we add 3K RAM expansion, the screen can stay where it is above the BASIC RAM area. But if we add 8K or more, the video chip can't see that high, and the screen memory must flip down to the bottom where it won't get in the way of your BASIC program. Which bottom, you may ask? It turns out to be block 12, which is memory address 4096 or hexadecimal 1000, even if the 3K expansion is in place.

You can move this around yourself, of course, and we'll be doing that in just a few moments.

The trick is mostly location 36869, which contains instructions on which blocks to use for screen and characters. We do it this way: Select which blocks you want for each. Now, multiply the screen block (not including the .5 if you're using it) by 16 and add the character block. POKE the result into 36869, and the job's done. We'll need to do a couple of other things for sanity's sake, but that's the main job.

The half page for the screen memory goes into location 36866; you invoke it by adding 128 to the column count if you want to go the extra distance. That means that under normal circumstances (22 columns), you want to POKE 36866,22 for an exact block number, and POKE 36866,150 to nudge to the extra half page.

An Adventure

Let's do something useless, but fun. We'll move the screen memory down to address zero (that's block 8). We can't play with this area—too many important things are happening there—but we can watch interesting things in progress, like the timer and the cursor doing their peculiar things.

First, the calculation. We want the character set to stay the way it is (block 0 for graphics), and we want to move the screen memory to block 8, so 8*16+0=128. No half block, so 36866 should be 22.

A preliminary step: Let's make sure that we don't print white-on-white by clearing the screen and typing:

```
FOR J=37888 TO 38911:POKE J,0:NEXT J
```

Ready? Here goes. Enter:

POKE 36869,128:POKE 36866,22

Press RETURN. No, we haven't crashed, but we'll have to type blind from now on.

First, examine the fascinating busy things that are under way. The timer is working away in three bytes. At first glance, only one byte seems to be changing. The cursor flash is being logged and timed somewhat below. And if you start typing, you'll see a whole new series of working values coming into play. Indeed, if you can type blind, you might try PRINT 1234+5678 and watch the flurry of activity.

If you type a lot, the screen will start to scroll, and the display will start to vanish as the colors are rolled off the top.

Restore everything to normal by holding down RUN/STOP and tapping the RESTORE key.

This has been a first exploration, but you may feel that you understand better what the video chip is up to. Indeed, you may feel that you have gained some measure of control.

There's much more to be learned. This is a start.

Minimum Memory

It's worthwhile making an observation about the minimum VIC configuration here. We know that the video chip sees memory in an unusual way:

Some users have memory expansion permanently connected to their VIC machines. They don't want to plug and unplug the memory units. Yet some programs call for a minimum VIC with only 5K. A few POKEs can reconfigure any machine to this minimum configuration.

First, we set the Limit-of-BASIC:

POKE 55,0:POKE 56,30:CLR

And then put the screen into place (block 15.5):

POKE 36869,240:POKE 36866,150:PRINT CHR$(147)

This takes care of the high end of memory. It's not always necessary, but we can also set up the low end. Hold down the RESTORE key and press the RUN/STOP key to clear the screen. Then enter:

POKE 4096,0:POKE 43,1:POKE 44,16:NEW

Small Character Sets

A full character set, 256 characters, takes up 2048 bytes of memory; there are eight bytes for each character. We have tried copying over the whole set. On a small VIC, it takes up a lot of our available RAM and starts to cramp our program space. Can we omit some of the characters and save space? Yes, we can.

Our program may not need the reverse video characters. If so, there's a savings of 1024 bytes. Be careful: Reverse video is used for flashing the cursor. If you give it up, the cursor may not work in quite the same way.

But there's more. Which are the characters that we use the most? Well, the alphabetic characters A to Z, the space character, of course, and the numbers 0 to 9. What luck! These characters are bunched together within the first 58 of the character set, including a few spares. 58 times 8 gives us 464 bytes of storage—not bad for a functional character set.

We could do better than this if we had a specialized display that could work from very few characters. For example, a game might use only four characters: a ball, a ninepin, a gutter, and the all-important space character to give us blank space. Even so, we might be tempted to go the whole alphanumeric set—to display scores, instructions, and the like.

A little arithmetic shows us a convenient arrangement. The character set must start on a block boundary. Screen memory may start on a half-block boundary. If we put them one behind the other, this would give us 512 bytes for the character set, enough for 64 characters.

In fact, let's try this, with the partial character set at block 15 and the screen at its usual block 15.5. We can write a simple graphics demonstration program.

A Little Program

```
100 POKE 56,28:CLR          (lower Limit-of-BASIC)
110 FOR J=0 TO 63           (copy of 64 characters)
120 J1=J*8                  (8 bytes per character)
130 FOR K=0 TO 7            (copy each byte)
140 POKE J1+K+7168,PEEK(J1+K+32768)
150 NEXT K,J
```

Here come our custom characters. We'll draw a ship in two characters: the left half in character 27 and the right half in character 28. The "pixels" of the drawing are in the DATA statements:

```
160 DATA 0,0,4,4,127,63,31,0
170 DATA 0,0,192,192,252,248,240,0
180 FOR J=27 TO 28 (two specials)
190 J1=J*8
200 FOR K=0 TO 7
210 READ X:POKE J1+K+7168,X
220 NEXT K,J
```

Now we put our new character set in gear:

```
230 POKE 36869,255
240 POKE 36866,150
```

And we'll draw our little ship with a simple demonstration program. Note that screen character 27 corresponds to ASCII character 91.

```
300 PRINT CHR$(147);"SHIP GRAPHIC"
310 FOR J=2 TO 18 (left to right)
320 PRINT CHR$(19)
330 PRINT TAB(J);CHR$(32);CHR$(91);CHR$(92)
340 FOR K=1 TO 99
350 NEXT K,J
360 GET X$:IF X$=""GOTO 300
```

The program ends here. Restore the regular character set:

```
370 POKE 36869,240
```

Run the program. After the initial pause for generating the new character set, a ship will move across the screen. You can adjust its speed by changing the value of 99 in line 340. The program will terminate if you hold down any key.

If you press RUN/STOP, the program will break with the custom character set still in place. You'll notice the lack of a cursor; apart from that, you can type most alphanumeric characters without problems. You might like to look around to find out which keys now represent left and right halves of the ship. When you are finished playing, type CONT to allow the program to continue, and then terminate by holding a key down.

You may notice that the program does not restore the 512 bytes that it takes for the character generator. So the character set is protected, and you can try going back to it if you wish with a POKE 36869,255. Eventually, you may wish to make the program complete by adding line 380, with a POKE 56 and a CLR. I'll leave you to work out the proper details.

Here's a question that may cross your mind: If the character generator starts at block 15, where would the video chip go for the reverse characters? They would be in the next block, but we don't have a block 16. What happens? The video chip address *wraps around,* and we go to block 0. The characters in block 0 are not reversed, of course, and that's why the cursor doesn't flash.

We can do some good work with a very small character set. It doesn't necessarily have to be big to be useful.

Another thing that we have spotted in this episode—we can build effective graphic pictures by using more than one character. Our program used two separate characters which together drew a ship, but we can use three, four, six, or more as needed.

Creating Your Own Characters

Suppose we want to lay out our own screen and characters. It seems simple enough: Choose the locations for screen memory and character set, and POKE the block numbers (screen block times 16 plus character block) into address 36869. If the screen is positioned at an exact block boundary, we put a low number (such as 22) into 36866; otherwise, we place a high number there (such as 22 plus 128, or 150). The 22, by the way, is for 22 columns—standard for the VIC.

However, we have two major tasks to perform. First, we must make sure that the memory we are using to feed the video chip isn't needed by somebody else. Second, we must tell the VIC-20 operating system about our new screen location. Changing the video chip isn't enough—the parts of the computer that print to the screen must be told that the screen is somewhere else.

Let's try an example: We'd like to put our own character set into a 5K VIC. Things will get a little crowded, since we need to use 2K for the extra character set. But we can make it work.

Finding Room

Almost all the spare RAM memory of the computer is assigned to BASIC. This is to allow you to write programs as large as possible. We must take memory away from BASIC to make room for the new video stuff.

BASIC memory is a single continuous block. It goes from

Start-of-BASIC (whose address is logged in locations 43 and 44) to Limit-of-BASIC (whose address is logged in locations 55 and 56). No breaks: You can't pop a screen in the middle and have BASIC memory skip around it. You can find the Start-of-BASIC address on your machine by typing

PRINT PEEK(43)+PEEK(44)*256

or the Limit-of-BASIC address by typing

PRINT PEEK(55)+PEEK(56)*256

Remember these; they are a good way to check the values after you've changed things around.

Making Room

We have a choice. We can move down the Limit-of-BASIC, which will give us room at the top. We can move up the Start-of-BASIC, which will make room at the bottom. Or we can do both, if we don't mind the extra work. Whatever we do, we must realize that we're trimming back the area available for BASIC.

If we move down the Limit-of-BASIC, we must say CLR after we do so. This gets rid of variables and strings that might be in embarrassing places. Don't forget this.

Moving the Start-of-BASIC upwards takes a good deal of care. Rule 1: We must POKE a value of 0 into the first available location. Rule 2: We must set the Start-of-BASIC pointer so that it points to the next location behind the 0. Rule 3: When we're finished, we must type NEW to make sure that BASIC is cleanly set up in the new memory area.

How do we set up these pointers? Divide the desired address by 256: The remainder goes into the first byte, and the quotient into the second byte. For example, we want to move the Limit-of-BASIC down to 6144. 6144 divided by 256 gives 24 with zero remainder, so we POKE 55,0:POKE 56,24:CLR.

Another example: We want BASIC to start at 5120. First, place the zero: POKE 5120,0. Now, the pointer must be set to 5121 (behind the zero); since 5121 divided by 256 gives 20 with a remainder of 1, we POKE 43,1:POKE 44,20:NEW.

Planning

We want to set up a complete character set, including the reverse characters. That will take 2K of memory—we could do it in 1K if we were willing to skip the reverse characters. Let's plan to put this at the top of memory, starting at block 14.

The screen takes up half a block, of course, and it seems to make sense to set this up just below the characters; so we'll pick block 13.5 (we can set the screen on a half-block boundary, remember?). This calls for a Limit-of-Memory of 5632. You may have noticed, by the way, that the Limit-of-Memory pointer is set one location beyond the last usable value. In other words, BASIC can use 5631, but it can't use 5632, the Limit value.

Arithmetic time. 5632 divided by 256 gives 22 with zero remainder; so type:

POKE 55,0:POKE 56,22:CLR

and the space is allocated. You can try PRINT FRE(0) and see what a puny amount of memory you have left.

We haven't yet told the video chip to use this area. We're not ready to point the chip towards the new character set area; we haven't put any characters there yet. So let's move characters in—but wait a moment.

The new character set would go into the same area of RAM as the present screen location. This would give us an odd-looking screen. We could live with that part, but the screen would also do odd things like scrolling, which would move the character set we had so carefully placed. We'd better move the screen to a clear area first.

Moving the Screen: Video and System

The character set can remain as block 0 for the moment; we'll want to shift the screen to block 13.5, with POKEs to 36869 and 36866. But we need to do two extra things at the same time: tell the computer system where to find the new screen, and clean up the screen area.

The POKEs to 36869 and 36866 tell the video chip all it needs to know about delivering the screen memory to the video output circuits. But unless we tell the computer system about the change, it will continue to put new characters into the old screen area. We tell it with a POKE to location 648. Here's how the arithmetic goes.

Divide the new screen memory address by 256, and POKE the result into address 648. Our example puts the screen at 5632, which gives 22 when divided by 256; so we'll POKE 648,22. But we need to do everything together. Let's work out the other POKEs. The screen goes to block 13.5, and the character set remains at block 0 for the moment. Since 13 * 16

− 0 = 208, we'll need to POKE 36869,208. The half-block is logged into the system with POKE 36866,128+22, and so we move the screen with:

POKE 648,22:POKE 36869,208:POKE 36866,150: PRINT CHR$(147)

CHR$(147) is the clear-screen character, by the way.

Making Characters

Now we can copy the character set from its fixed appearance in 32768 to our planned new area at 6144 and up. If we copy the character set exactly, we've wasted a lot of memory; we'll get the same characters as before. To show we have control, we'll vary the normal character set slightly.

Instead of the normal graphics set—uppercase and graphics—we'll mix the two as we copy them over. Not too useful, perhaps, but when we cut over to the new character set, you'll be able to see that something new has happened. Enter the following program:

```
100 FOR J=0 TO 255 STEP 2
110 J1=J*8
120 FOR K=0 TO 7
130 POKE J1+K+6144,PEEK(J1+K+32768)
140 NEXT K
150 FOR K=8 TO 15
160 POKE J1+K+6144,PEEK(J1+K+34816)
170 NEXT K
180 NEXT J
```

Run this program; it will take a minute or two.

The Final Touch

The screen has already been moved, and the character set is in place and ready to go. Let's cut it in, and the project will be complete.

The screen is still at block 13.5, and the new character set will be at block 14. So we do 13 * 16 + 14 and get 222; we'll want to POKE 36869,222. Since we're not moving the screen this time, the half-block value in 36866 is still good; we won't need to change that. We're ready. Enter:

POKE 36869,222

Now try typing or listing the previous program, and look at the odd combination of characters we've created. We must tie things together neatly—BASIC, the operating system, the video chip—to make it all work properly. But with good planning, we can make the screen do marvelous things.

Hi-Res

We've meddled with the character set, both built-in and homebrewed. But we haven't seemed to deal with achieving that mystic goal—high-resolution screen control.

We've dealt with custom characters. And as Glinda the Good Witch could have said to Dorothy, "If you had known their powers, you could have done it the very first day." In other words, we've been looking at high resolution all along without recognizing it.

Here's the trick: If every position on the screen contained a different character, and if we can define any character at will, we can define any spot on the screen as we wish.

Mechanically, we do it this way: The first cell on the screen will contain character 0; the next will contain character 1; and so on. To change the upper-leftmost pixel on the screen, we modify the upper left pixel of character 0, and the screen immediately shows the change.

This is a change from our usual use of screen and character set. Our screen memory is now totally fixed and must not change. Normal printout and things like scrolling must stop. The characters, on the other hand, are now completely variable, with pixels turning on and off according to what the picture needs.

Wait—there's a problem. It seems that the screen has room for 506 characters; yet we know that we can make only 256 individual characters. Something doesn't fit. How can we resolve this problem?

There are two ways. One is to use *double characters*—the jumbo-sized characters that we get when we POKE an odd number into address 36867. Each of our 256 characters now occupies twice the space on the screen, so that we can cover the screen easily. The character set table now becomes huge, of course. Each character takes 16 bytes to describe, making the whole table up to 4096 bytes long.

Since we're trying to describe things you can achieve in an unexpanded VIC, this becomes impractical—it's hard to take 4K away from a machine that has only 3.5K available to start with. On a machine with memory expansion, however, this is quite practical; read on, for we'll use tricks on the small machine that will come in handy even on the big ones.

The other method is this: Cut the size of the screen so that it contains only 256 characters or less. We can store the number of columns and rows we want into 36866 and 36867. POKE 36866,16 will set 16 columns; and POKE 36867,32 will set 16 rows (we must multiply the number by 2 here). How many characters can we store? 256—and that number may sound familiar by now.

By the way, BASIC won't know how to cope with the peculiar row and column counts if you do this as a direct command, so be prepared for an odd-looking screen. Neatness fanatics will want to center the remaining display by appropriate POKEs to 36864 and 36865, but I'll leave this as an exercise for you.

Diving In

Enough of this abstract theory. Let's dive into a program to prove that even the humble minimum VIC can do high-resolution graphics.

```
100 POKE 56,22:CLR          (Drop top of BASIC)
110 POKE 36869,222          (Relocate screen...)
120 POKE 36866,144          (and character set)
```

Note that the above line sets the screen to a half-block (128) and sets up 16 columns instead of the normal 22 (128 plus 16 gives 144). We may as well go ahead and change the rows:

```
130 POKE 36867,32           (16 rows times 2)
200 FOR J=6144 TO 8191
210 POKE J,0:NEXT J
```

We've cleared the entire character set to zero (all pixels off). Now let's set up the screen with character 0 in the first slot, etc.:

```
300 FOR J=0 TO 255
310 POKE J+5632,J
320 NEXT J
```

Let's set all characters to color black:

```
330 FOR J=37888 TO 38911
340 POKE J,0:NEXT J
```

Our screen is now ready. Serious graphics takes quite a bit of math (dividing by 16 to find the row and column; dividing by 8 for the pixel position), but we'll substitute a little simple coding to draw a triangle:

```
400 FOR J=6792 TO 6816 STEP 8
410 POKE J,255                (horizontal line)
410 NEXT J
500 FOR J=6280 TO 6664 STEP 128
510 FOR K=J TO J+7
520 POKE K,128                (vertical line)
530 NEXT K,J
600 FOR J=6280 TO 6704 STEP 136
610 X=128                     (leftmost pixel)
620 FOR K=J TO J+7
630 POKE K,PEEK(K) OR X
640 X=X/2                     (move pixel right)
650 NEXT K,J
700 GOTO 700
```

The program is now complete. It will wait in a loop at line 700 until you press RUN/STOP. When you do so, a number of odd things will happen. The computer will try to print the word READY into screen memory, but screen memory is intended for a different usage now, and all that will result is screen clutter.

Bring everything back to sanity by holding down RUN/STOP and hitting the RESTORE key.

Extra Ideas

Effective graphics calls for the use of a fair bit of mathematics. To place (or clear) a pixel, you need to find the row and column by dividing the X and Y coordinates by the appropriate scaling factor. You need to change this to a screen character number by multiplying the row number by the total number of columns and then adding the column number. Multiply this by 8, and you'll get the position where the character is located within the character set. Now we must go for the pixels within this character: The bits within a byte are pixels *across*, and the eight consecutive bytes are pixels *down*. Now you know why people buy a Super Expander—to save them from the math.

Even when you have plenty of memory available, which allows you to use double characters and gets lots of pixels on the screen, it's usual to trim the screen a little. The normal 22 columns by 23 rows are usually trimmed back to 20 columns by 20 rows (actually ten rows of double characters). This does two things: It makes the arithmetic a little easier, and it drops the memory requirements from 4096 bytes for a full deck down to only 3200 bytes. This, in turn, gives us space to pack screen memory into the same 4K block. That's handy because we cannot be sure that the video chip will have access to any more than 4K of RAM. BASIC, of course, will long since have been moved to occupy memory from 8192 and up.

If you want to add text to the high-resolution display, it's a snap. Just copy the characters you want from the character set ROM and transfer them to the appropriate character slots on the screen.

Don't forget that you can POKE appropriate values into 36864 and 36865 to center the graphics neatly. Our example looked a little lopsided; try your hand at making it neater.

High resolution is there and waiting.

Yes, you can do it on a small screen PET. There's a good bit of math needed. You may find this a challenge: After all, isn't that what a computer does best?

Even if the mathematics befogs your mind and causes you to go out and buy a Super Expander, you'll have learned a few new things. First, the Super Expander doesn't make graphics possible—they were there all the time—it just makes them easier. Second, you'll have a better idea of what's going on inside your marvelous VIC computer.

Relocating VIC LOADs

Tony Valeri

When you need to relocate a program in the VIC's memory, you can use this simple technique.

As most VIC users know, the VIC relocates all programs to the start of BASIC memory unless told otherwise. For example, LOAD 1,1 tells the computer to load the program into the area of memory specified by the tape.

So we have two choices; we can either load a program into the start of BASIC memory (usually $1000) or load a program back into its original location in memory. But what if we want to place a previously prepared subroutine at the end of a program, or relocate a machine language program to some novel place in memory? There's not much we could do short of retyping it.

Basically, what happens during a LOAD is that, after a few pointers are stored (buffer location, program name, etc.), a routine is called that searches the tape for the next program header, and then reads it into the cassette buffer. The LOAD routine next checks the buffer to find out whether the program being loaded is to be placed into the locations specified in the buffer or is to be relocated to the start of BASIC. Now, if we could bypass the routine that does this, things would be much simpler.

In the table, you'll see the locations necessary to relocate a program anywhere in the VIC's memory.

Use a SYS 63407. The computer will prompt with the usual PRESS PLAY ON TAPE. The difference is that the computer now prints READY as soon as the program is found. What has happened is that the SYS 63407 tells the computer to load the next program header and store the information in the cassette buffer.

To find out the original start and end locations of your program, type in:

PRINT PEEK(829) +PEEK(830)*256,PEEK(831)+
256*PEEK(832)

Increasing the value in locations 829 and 831 by one will place the program one byte higher in memory. Increasing the value in locations 830 and 832 by one will place the program 256 bytes higher in memory. Decreasing the values in these locations will have the opposite effect.

After the buffer has been changed, a SYS 62980 will return control of the computer to the LOAD routine. Now load the main body of the program into memory, but load it into the new locations just specified.

See It Work

To demonstrate this technique, we'll fill the screen with data from tape. The demonstration is for the unexpanded VIC, so you'll need to remove or disable any memory expansion. To prepare, type in the following line in direct mode:

POKE 46,PEEK(46) + 2

This reserves two pages (512 bytes) at the end of your BASIC program for data.

Type in the following one-line program exactly as it appears. Any additional spaces will cause errors. The program will fill the space between the end of the program and the start of variables with the screen POKE value for the ball character.

10 FORA=4124TO4629:POKEA,81:NEXT

After checking your typing, RUN the program, then SAVE it to cassette.

Next, rewind the tape and reset the VIC with a SYS 64802. Start the relocatable load by typing:

SYS 63407

After the VIC reads the tape header into its buffer, you can check the original start and end addresses by PEEKing addresses 829–832 as indicated above. The starting and ending addresses should be 4097 and 4636. Instead we want to put the block of 506 ball characters into screen memory, which starts at location 7680. To accomplish this, type in the following series of POKEs:

POKE 829,229:POKE 830,29:POKE 831,0:POKE 832,32

You'll need to prepare the screen by changing the colors to make the balls visible. Try POKE 36879,76. Finally, complete the tape LOAD by typing:

SYS 62980

The data coming in from tape will be directed to the screen memory area and will fill the display with ball characters.

	HEX	DEC
Routine To Load Header	$F7AF	63407
Buffer Start Of Prog.	$033D & $033E	829 & 830
Buffer End Of Prog.	$033F & $0340	831 & 832
Continue Load	$F607	62980

Locations necessary to place a program anywhere in the VIC's memory.

Chapter 2

Recreations

Demon Star

Stan McDaniel

Packing arcade-style quality into the unexpanded VIC is a tough BASIC programming challenge, but you'll find that "Demon Star" comes through with flying colors (plus sound and custom graphics).

Your universe is being invaded by Demon Stars, which appear at unpredictable intervals and can destroy all life forms in their vicinity. A defense shield protects your home area, but vast amounts of energy are needed to maintain it.

Your mission: to penetrate lifeless areas already devastated by Demon Stars and to transmit energy units back to your home planet.

Your starship moves horizontally, vertically, and diagonally, controlled by a joystick. The joystick fire button shoots matter conversion torpedoes. When you are not busy defending yourself from Demon Stars or chasing dangerous Quasars, you convert celestial objects into valuable energy units by scoring direct hits with your torpedoes. Nebulae net you five energy units, single-spiral galaxies ten units, stars 15, and double-spiral galaxies 20.

Blasting Quasars

Quasars appear frequently, accompanied by a deep-pitched warning sound. If you do not destroy a Quasar before it disappears, you lose 500 energy units. An energy drop below zero destroys your ship, as does a collision with any celestial object. Demon Stars show up less frequently than Quasars, but they are far more dangerous. Their negative energy field restricts the range of your torpedoes, and failing to hit a Demon Star within the allotted time costs you your ship.

When you shoot down a Quasar, you receive 100 energy units. A Demon Star gains you a whopping 500 units. In addition, you receive a 100-unit bonus every time you manage to collect 1000 units on your own. But watch out! When your energy store exceeds 3000, your ship will begin moving to a more densely populated area of the universe, and Demon Stars will appear with greater frequency. To help you keep track, your total accumulated energy units are displayed continuously.

A timer also appears whenever a Quasar or Demon Star enters your area.

Preparing Demon Star

"Demon Star" consists of two programs: the main program, and a second program which records a special data file on the cassette tape following the main program (disk users note the changes needed on line 115 of Program 1 and line 10 of Program 2). Turn on your VIC, type NEW, and then type in the main program (Program 1). It is fairly long and very compact, so you must type with care. (Be sure to read the article "Automatic Proofreader," Appendix J, to help you with your typing.)

After the main program has been typed, copy it to cassette with a SAVE "DEMON STAR" command or to disk with SAVE "DEMON STAR",8. Tape users should then rewind the tape and issue a VERIFY "DEMON STAR" command. When the verification is complete, press the STOP button on the recorder. Now clear memory with NEW and type in the "DATA" program (Program 2). When this is completed accurately, type RUN. Tape users will be prompted to press PLAY and RECORD on the recorder. Program 2 will create a data file entitled "DF" on your disk or immediately following "DEMON STAR" on your tape. Finally, SAVE the data program under the name "DATA", just in case you need it again. A spare copy of the main program also is a good idea.

Once you have completed the process, this is what you should have: If you are using tape, your tape should have the main program (Program 1), followed by the data file called "DF," followed by a copy of Program 2 called "DATA"; disk users should have three files on their disks, Program 1 ("DEMON STAR"), "DF," and "DATA."

To play the game, plug a joystick into the VIC game port. Insert the disk or tape containing Demon Star into the drive or cassette player (be sure to rewind the tape). Then LOAD the main program, Program 1. When the program completes loading, tape users should leave the PLAY button down. RUN the program. The screen color will change and a WAIT message will appear.

Wait patiently while the program loads the data from the data file. Tape users will see the tape stop and start several times. Do not interfere with this process. The loading is complete when a screen prompt, LEVEL?, asks you for your choice of skill levels (there are ten levels of play). Be sure to press STOP on the recorder at this point. Type a number (the skill level you want) between 1 and 10, and press RETURN.

I prefer to start at level four, but a first-time player might want to pick level two or three. As soon as the level is entered, the screen color will shift to deep blue, and after a few moments the Demon Star universe will scroll majestically into view. Your starship will be resting at the bottom left of the screen. After a brief pause (giving you time to look things over), the ship will take off, bent upon its energy-gathering mission.

Demon Star Strategy

If there are any celestial objects immediately in front of the ship which might interfere with your takeoff, press the fire button immediately and hold it down. The obstructing objects will be destroyed as your ship gets underway. Yellow double-spiral galaxies are worth the most, so head for them (controlling direction with the joystick) and start firing. Your normal firing range is about one-third the screen width. When a Demon Star is on the screen, the range drops to slightly less than one-fourth the screen width.

When a Quasar appears, you must find your way to it and fire your torpedo within the time limit, making some split-second decisions about the shortest path through the maze of objects. You will have time to blast one or two objects out of your way, if necessary. An unvanquished Quasar drains 500 energy units, so you will want to accumulate more than 500 units as soon as you can to keep from blowing up (which happens, remember, anytime your energy drops below zero). The best way to insure survival is to destroy five Quasars right away.

When you reach higher levels of play, blast strategically placed objects out of your way quickly, carving out paths which give your ship easier access to all areas of the screen. Your ship and your torpedoes can move across the screen border and come back on the other side; do not forget this when you are in a tight spot.

The entry of a Demon Star is heralded by a blinding flash, followed by the roaring of pure negative energy. Keep calm enough during this nerve-rattling display to find the Demon Star, get into firing position, and destroy it before your time runs out. Demon Stars are the same color as regular stars, so even though they have a different shape, you have to be sharp-eyed to spot them in time.

The skill levels automatically advance as you score more points. A beginning player starting at skill level four will find it difficult, but not impossible, to reach a score of 1000. If you manage to accumulate more than 6000 energy units, you will be playing at level eight. An expert will reach 10,000 units at this level. Master players will be able to play at skill levels nine and ten. The game will not advance automatically to these highest levels, so the player must select them when the game starts.

If your ship is destroyed, your total score will be displayed, and you will be asked if you want to play another round. Typing Y will start a new round, and typing N will exit the program. If you exit the program by accident and you want to play again, type RUN 130, not just RUN.

Customizing the Program

For those who would like to change some of the characteristics of play, the following information about Program 1 will prove helpful. To make changes in the program, follow the editing procedures in the first two chapters of *Personal Computing on the VIC-20* (the manual that came with your computer).

Lines 155–175 lower the screen out of visible range, print the game universe on the screen, and scroll the screen up again. The expression (3*SK) in line 160 determines the number of celestial objects which will appear on the screen for any selected skill level (SK). If you want to make the game easier, you can change this to (2*SK). Note the FOR/NEXT loop at the end of line 175; it is important because the delay it creates gives the player time to look over the situation before the action begins. To create a longer delay, make the loop maximum greater than 1000.

The main program loop is in lines 180–255. The IF statement at the end of line 180 sends the program (at random intervals) to the Quasar/Demon Star routine at lines 405–420. If you want the Quasars and Demon Stars to appear with greater frequency, decrease the argument of the function FNR. For example, change FNR(10) to FNR(8). Increasing the value will decrease the frequency.

If a player runs out of time when a Quasar or Demon Star is on the screen, the IF statement at the end of line 185 sends the program to a time-out routine at lines 450–460. As the program stands, a player has about 12 seconds before running out of time. To increase the amount of time, increase the figure 700 in line 185.

As you know, the torpedo firing range is inhibited by the appearance of a Demon Star. This is accomplished in line 405 by the expression RG=5, lowering the range (RG) to five screen locations or about one-fourth of the screen width. Changing the value of RG at this point will change the Demon Star's effect upon the range.

The normal range of the torpedo is set by RG=8 in line 135. The current value of eight screen locations seems just right. The longer you make the range, the easier the game will be. If you decide to try out a different range, you must change the value of RG not only in line 135, but also in line 320 (this is where the range is reset to normal after a Demon Star has been destroyed).

Scoring for objects hit is accomplished by lines 320–330. The formula PT=PT+(L−41*5) in line 325 sets the point scores 5, 10, 15, and 20 for each of the four types of objects. Changing the 5 to 6 here would change the scores to 6, 12, 18, and 24. Line 330 sets the value of a Quasar, and line 320 establishes the 500-point value of the Demon Star. The bonus points are controlled by lines 67–69.

Another scoring control appears in line 455, where PT=PT−500 decreases the score when a Quasar times out. To vary the amount of decrease, change the figure 500 in this line.

Program 1. Demon Star—Main Program

Please read the section entitled "Preparing Demon Star" and the article "Automatic Proofreader," Appendix J, before typing in Demon Star.

```
100 POKE36879,253:PRINT"{CLR}WAIT"            :rem 106
105 P=28:POKE56,P:POKE52,P:POKE51,PEEK(55):CLR
                                               :rem 31
110 FORI=7168TO7679:POKEI,PEEK(I+32768-7168):NEXT
                                              :rem 163
114 REM FOR DISK USE OPEN1,8,0,"DF" FOR LINE 115
                                               :rem 60
115 OPEN1,1,0,"DF"                            :rem 173
```

```
120 INPUT#1,X:IFX=999THEN130                    :rem 199
125 FORJ=XTOX+7:INPUT#1,Y:POKEJ,Y:NEXT:GOTO120:DIM
    MX%(2,2),O%(4)                              :rem 249
130 CLOSE1:PT=0:BB=1000:INPUT "LEVEL (1-10)";SK:IF
    SK<1ORSK>10THEN130                          :rem 38
135 D0=37154:D1=D0-3:D2=D0-2:CD=30720:C=22:R=23:SP
    =32:FL=0:RG=8                               :rem 20
140 V=36878:VN=V-1:VS=V-2:E=42                  :rem 254
145 FORI=0TO2:FORJ=0TO2:SP=SP+1:MX%(I,J)=SP:NEXTJ,
    I:SP=32                                     :rem 125
150 DEFFNA(W)=7680+X+C*Y:DEFFNB(W)=PEEK(FNA(W)):DE
    FFNR(W)=INT(RND(1)*W)                       :rem 22
155 PRINT"{CLR}":POKEV+3,128:POKEV+1,14:POKEV-9,25
    5                                           :rem 124
160 FORI=0TO3:O%(I)=E:E=E+1:NEXT:K=4:FORH=0TO3:FOR
    J=1TO(3*SK):GOSUB440                        :rem 102
165 POKERP,O%(H):POKERC,K:NEXTJ:K=K+1:NEXTH     :rem 8
170 MC=0:MR=-1:SX=3:SY=22:SH=8167:POKESH,36:POKESH
    +CD,3                                       :rem 123
175 FORT=128TO24STEP-1:POKEV+3,T:FORD=1TO10:NEXTD,
    T:FORH=1TO1000:NEXT                         :rem 184
180 PRINT"{HOME}{WHT}"PT"{BLU}":IFFL=0ANDFNR(10)=3
    THEN405                                     :rem 244
185 IFFL=2THENPRINT"{HOME}{DOWN}{GRN} ";RIGHT$(TI$
    ,2);"{BLU}":IFTI>700THEN450                 :rem 94
190 GOSUB390:IFFBTHENX=SX:Y=SY:PX=MC:PY=MR:GOTO260
                                                :rem 107
195 U=0:W=0:IFJ0THENU=1                         :rem 148
200 IFJ2THENU=-1                                :rem 188
205 IFJ1THENW=1                                 :rem 149
210 IFJ3THENW=-1                                :rem 192
215 IFU=0ANDW=0THENU=MC:W=MR                    :rem 62
220 MC=U:MR=W:SX=SX+MC:SY=SY+MR:IFSY<0THENSY=R
                                                :rem 143
225 IFSY>RTHENSY=0                              :rem 172
230 IFSX>21THENSX=0                             :rem 183
235 IFSX<0THENSX=21                             :rem 186
240 X=SX:Y=SY:J=FNB(0):IFJ<48ANDJ>41THENFL=1:GOTO3
    15                                          :rem 32
245 POKESH,SP:POKESH+CD,0:SH=FNA(0):POKESH,MX%(MC+
    1,MR+1):POKESH+CD,3                         :rem 33
250 IFFL<>2THENPRINT"{HOME}{DOWN}{BLK}{3 SPACES}
    {BLU}"                                      :rem 55
255 GOTO180                                     :rem 110
260 FORI=1TORG:POKEV,15:X=X+PX:Y=Y+PY:POKEVS,241
                                                :rem 143
265 IFI<>1THENPOKET,SP:POKET+CD,0               :rem 108
270 IFX>21THENX=0                               :rem 21
275 IFX<0THENX=21                               :rem 24
```

```
280 IFY>RTHENY=Ø                                       :rem 7
285 IFY<ØTHENY=R                                       :rem 1Ø
29Ø J=FNB(Ø):IFJ=SPTHEN3ØØ                             :rem 46
295 IFJ<48ANDJ>41THENI=RG:NEXTI:GOTO315               :rem 253
3ØØ T=FNA(Ø):POKET,37:POKET+CD,1:POKEV,Ø:FORH=1TO2
    5:NEXTH:NEXTI                                      :rem 24
3Ø5 POKET,SP:POKET+CD,Ø:IFPEEK(RP)=47THENPOKEV,15
                                                       :rem 132
31Ø GOTO18Ø                                            :rem 1Ø2
315 L=FNB(Ø):XP=FNA(Ø):IFFL=1THEN34Ø                   :rem 125
32Ø IFL=47THENPT=PT+5ØØ:FL=Ø:RG=8:GOTO34Ø              :rem 23
325 IFL<46ANDL>41THENPT=PT+((L-41)*5)                  :rem 155
33Ø IFL=46THENPT=PT+1ØØ:FL=Ø                           :rem 193
335 IFPEEK(RP)=47THEN35Ø                               :rem 177
34Ø POKEXP,58:POKEXP+CD,2:POKEVN,22Ø                   :rem 245
345 FORM=15TOØSTEP-1:POKEV,M:FORN=1TO25:NEXTN,M:PO
    KEVN,Ø:POKEVS,Ø                                    :rem 22
35Ø FORM=1TO3:POKEXP,58:POKEXP+CD,2:FORN=1TO25:NEX
    TN:POKEXP,SP:POKEXP+CD,Ø                           :rem 136
355 FORN=1TO25:NEXTN,M:IFFL=1THEN37Ø                   :rem 28
36Ø IFPT>BBTHEN425                                     :rem 88
365 POKET,32:POKET+CD,Ø:GOTO18Ø                        :rem 137
37Ø POKEVN,Ø:POKEV-9,24Ø:POKEV+1,17Ø:IFPT<ØTHENPRI
    NT"{CLR}{BLK}ENERGY LOSS: ";PT:GOTO38Ø:rem 189
375 PRINT"{CLR}{BLK}SHIP DESTROYED. ENERGYTRANSMIT
    TED: ";PT                                          :rem 218
38Ø INPUT"ANOTHER GAME{SHIFT-SPACE}(Y/N) ";A$:IFA$
    ="Y"THEN13Ø                                        :rem 2Ø4
385 END                                                :rem 119
39Ø POKEDØ,127:P=PEEK(D2)AND128:JØ=-(P=Ø):POKEDØ,2
    55                                                 :rem 247
395 P=PEEK(D1):J1=-((PAND8)=Ø):J2=-((PAND16)=Ø)
                                                       :rem 91
4ØØ J3=-((PAND4)=Ø):FB=-((PANDSP)=Ø):RETURN
                                                       :rem 252
4Ø5 GOSUB44Ø:IFRP<77Ø6THENGOSUB44Ø:POKERP,47:POKER
    C,6:POKEVN,241:GOSUB465:RG=5:GOTO415               :rem 24Ø
41Ø POKERP,46:POKERC,2:POKEV-4,197                     :rem 69
415 POKEVS,Ø:POKEV,15:FORH=1TO2ØØ:NEXT:POKEV-4,Ø
                                                       :rem 32
42Ø FL=2:TI$="ØØØØØØ":GOTO185                          :rem 68
425 POKEVS,Ø:POKEVN,Ø:PRINT"{CLR}{YEL}BONUS 1ØØ":F
    ORH=1TO1ØØØ:NEXT                                   :rem 64
43Ø PT=PT+1ØØ:BB=BB+1ØØØ:IFPT>3ØØØANDSK<8THENSK=SK
    +1                                                 :rem 154
435 GOTO135                                            :rem 11Ø
44Ø X=FNR(C):Y=FNR(R):IFFNB(Ø)<>SPTHEN44Ø              :rem 4
445 RP=FNA(Ø):RC=RP+CD:RETURN                          :rem 76
45Ø POKEV,Ø:IFPEEK(RP)=47THEN37Ø                       :rem 2Ø4
```

```
455 POKERP,SP:POKERC,Ø:PT=PT-5ØØ:IFPT<ØTHEN37Ø
                                            :rem 187
46Ø PRINT"{HOME}{BLK}{6 SPACES}":FL=Ø:GOTO18Ø
                                             :rem 83
465 FORI=1TO6:POKEV+1,3Ø:FORJ=1TO3Ø:NEXT:POKEV+1,1
    4:FORJ=1TO3Ø:NEXT:NEXTI:RETURN          :rem 248
```

Program 2. Demon Star—Data Program

```
9 REM FOR DISK USE OPEN1,8,1,"DF" FOR LINE 1Ø
                                            :rem 17Ø
1Ø OPEN1,1,1,"DF"                           :rem 12Ø
2Ø READX:IFX<ØTHEN5Ø                         :rem 247
3Ø PRINT#1,X                                :rem 2ØØ
4Ø GOTO2Ø                                   :rem 255
5Ø CLOSE1                                    :rem 12
6Ø GOTO999                                   :rem 74
1ØØ DATA7432,128,126,121,112,1Ø4,68,64,32     :rem 9
11Ø DATA744Ø,6,8,16,252,252,16,8,6          :rem 183
12Ø DATA7448,32,64,68,1Ø4,112,121,126,128    :rem 18
13Ø DATA7456,24,24,24,24,6Ø,9Ø,153,129      :rem 128
14Ø DATA7464,Ø,Ø,36,24,24,36,Ø,Ø             :rem 66
15Ø DATA7472,129,153,9Ø,6Ø,24,24,24,24      :rem 128
16Ø DATA748Ø,1,126,158,14,22,34,2,4         :rem 226
17Ø DATA7488,96,16,8,63,63,8,16,96          :rem 219
18Ø DATA7496,4,2,34,22,14,158,126,1         :rem 235
19Ø DATA75Ø4,128,82,164,8Ø,21,17Ø,21,34     :rem 177
2ØØ DATA7512,24,6Ø,98,73,93,89,5Ø,4         :rem 248
21Ø DATA752Ø,Ø,42,28,62,28,42,Ø,Ø           :rem 115
22Ø DATA7528,Ø,112,152,188,189,25,14,Ø      :rem 127
23Ø DATA7536,6Ø,66,129,153,153,129,66,6Ø    :rem 242
24Ø DATA7544,56,16,84,254,84,16,56,Ø         :rem 43
25Ø DATA7632,162,116,124,56,6Ø,1Ø6,81,144    :rem 21
26Ø DATA999,-1                              :rem 231
999 END                                     :rem 13Ø
```

Galactic Code

Stanley E. Ingertson

Do you enjoy a secret, something a little mysterious, a little amazing? Here is a secret code program for the unexpanded VIC that codes, decodes, and provides a performance to amaze your friends.

You can use this "Galactic Code" to exchange messages, write a secret diary, amaze your friends, or just enjoy its displays.

First, here is an explanation of how to use the code program. Then, for the adventurous, there is a journey into the very heart of the program.

How to Use Your Galactic Code

Once you type in the program using the instructions found in "How to Type In Programs" (Appendix B) and "The Automatic Proofreader" (Appendix J), you enter the realm of the *Top Secret*. A secret message appears that only those from Alpha Centauri or Pluto, or those owning the decoder book (*COMPUTE!'s Third Book of VIC*) can decipher. To actually enter the program you must use a password. For now the password is SPY CODE, but later there is an explanation of how to change it. Just like in large computer systems, the password will *not* appear on the screen as you type. So type carefully and if you make a mistake, just press RETURN and type again.

Next a real computer MENU appears. A menu is a screen display that allows you to make the proper selection of the features in a computer program. In this program you can select: CODE a message, DECODE a message from your friends, see another GALACTIC message (each one is different), or STOP. If you choose to stop you will have to type RUN again to use the program. The stop feature makes it easy for you to LIST parts of the program and make the changes discussed later. By the way, try selecting something *not* on the menu and see what happens.

When you are using the CODE portion of the program, spell out any numbers which are actually in your message, and use the VIC's numbers to add mystery to your code. Any number you type will appear as a letter in your coded message. When someone is decoding, they will know that the letters are extra and will ignore them.

For the Daring

For those who would like to modify this program to suit themselves, have no fear. It's not that hard to do. If you type LIST 20 you can change the code's name to MIKE'S SECRET CODE or MARY'S DIARY CODE or whatever will make it personal. The portion of the line that says TAB(200) is used to position the title in the center of the screen. If your new title is longer or shorter, just decrease or increase the number in parentheses until it is centered. This number can be up to 255.

If you LIST 80 you can change the password. Just type over the words SPY CODE. Use the INSERT function (SHIFT and INST) if your new password is longer than SPY CODE. Be sure to always press RETURN after making a program change. This tells the computer you really do want to change something.

Line 190 looks a little complicated, but you can change the message at the end of it quite easily. Keep it friendly.

If you've already run the program, you know there's a portion where the computer *thinks* while it is coding or decoding your message. You can change the thinking display by using different characters in program line 420, or you can change the sound by changing the STEP value in line 430. A smaller number will slow down the sound and action, and a larger number will speed things up—try substituting STEP 10 or STEP 60 for STEP 25. If you want an unusual effect, change the step amount to STEP XX and add this line to the program:

```
425 XX=INT(RND(1)*40)+8
```

This will change the step value to a random number. You can experiment with this further by varying the values of 40 and 8. Try leaving the eight off completely and run it a few times—sooner or later you will have a glitch because the step value will equal zero.

You can modify the Galactic Code display by experimenting with lines 510 and 560. In line 510 the value you pick for T will determine how many random characters you see. Line 560 varies the number of lines printed in order to stop the VIC before it is forced to scroll upward. This gives you the maximum code display. Try the following values for T and R; 4 and 105, 11 and 38, 15 and 29.

This program uses DATA statements to produce the code symbols. If you LIST 1010, you will see the numbers used. You can change these code characters to other numbers, letters, or symbols. Each complete code character must have a comma after it. Some changes the VIC will allow with no problem. Sometimes, however, you will have to place your new code character in quotes. For example, if you wanted the VIC to print a heart symbol when you type an A, change the first data character on the line (the 1). Insert two spaces and type a shifted S within quotes.

For the True Adventurer

Now's the time to put your hiking boots on. This portion of the article is for those who dare to go right into the heart of the computer.

A computer's possibilities are limited only by another, greater computer—the human mind. If you are one of those people who look at the maze of symbols called a computer program and immediately wonder, "How does it work?," you're my kind of explorer. One enjoyable feature of the BASIC language is the variety of ways of *telling* the computer to do something. This flexibility is similar to the amazing power of English adjectives. In English you can say *huge* or *enormous* or *gigantic.* In BASIC there are often several ways to give the computer the same instruction. However, just like in English, if you say it any old way, you'll get a syntax error.

One example of this variety in BASIC, especially with your VIC, is the different ways to tell the computer to leave a blank line between displays on your screen. Three different ways are illustrated in program lines 120–150, 205–225, and 310–320. The first way uses commas.

Commas can be used in several ways after the PRINT command. If the statement being printed is over ten characters long, a comma will cause the next statement to begin on a new line rather than at the center of the same line.

Try these examples:

```
5 PRINT "{CLR}BEGIN HERE"
10 PRINT,,,,"DROPS 3 LINES--WHY?"
15 PRINT"DROP 3 AND PRINT",,,,"HI!"
20 PRINT"DROP 3 AND",,,,,,"PRINT--HO!"
25 PRINT"DROP 5 AND",,,,,,,,,,
30 PRINT "PRINT--HUM!"
35 PRINT "CENTER",,,"END"
```

Because line 5 did not end in a semicolon, the PRINT command in line 10 *adds* to the effect of the four commas. Lines 15 and 20 show the effect of more or less than ten characters before the commas.

You can also print a series of blank lines by stringing together a number of PRINT commands. For example:

100 PRINT:PRINT:PRINT:PRINT "HELLO"

The VIC has yet another way which is easier on the computer but harder on a typewriter. Once you type the first quotation marks in a PRINT statement, the cursor controls will produce special reversed field symbols representing the movements you want the cursor to make when the program is run. For example,if you hit the cursor-down key three times after a quotation mark, you would see three reversed-field Q's on the screen. In the following example, line 40 hides the cursor and forces you to press STOP to stop running the program.

```
5 PRINT "{CLR}{3 DOWN}";
10 FOR X=1 TO 19
20 PRINT "OVER{DOWN}{3 RIGHT}";
30 NEXT
40 GOTO40
```

Into the Jungle

Actually this program was written to illustrate the use of the DATA statement and the RESTORE command. It would have been possible to write this program using 52 IF statements. For example, to change an A to a 1, you could say: IF X$="A" THEN T$="1". You would need a statement for each code character and each decode character, plus some for spaces and ending symbols. However, there is a more challenging way which will help you understand the DATA statement.

When a computer reaches a READ command, it immediately jumps to the DATA line located somewhere in the program. It will READ the items in the DATA statement in order, *one* item *each* time it's commanded to READ. If you send it to the DATA line too many times, you will receive an OUT OF DATA error. When you use the RESTORE command, the computer will reset to the *beginning* of the DATA line.

Try the simple program below—you don't have to type line 10.

```
10 REM JERRY-1,BILL-2,JOE-3,PETE-4
20 INPUT A
30 FOR X=1 TO A
40 READ B$
50 NEXT X
60 PRINT B$
70 RESTORE
80 GOTO 20
100 DATA JERRY,BILL,JOE,PETE
```

If you type 3 the computer will print JOE; a 4 will give you PETE. Here's what actually happens. When you type a 3, line 30 becomes FOR X=1 TO 3. The computer begins going around a loop. Each time it's told to READ B$, the computer will advance one position along the DATA statement. The first time B$ becomes JERRY, then BILL, and finally the third time it becomes JOE. The computer has now completed the loop and moves on to line 60. Since B$'s final value is JOE, that is what is printed. Line 70 tells the VIC to start at the beginning of the DATA statements the next time a READ command appears. Finally, line 80 sends the computer back for another number from you.

By simply substituting A, B, C, D, ... for the names here, we have the portion of our program that does the decoding. This is actually found in lines 325–360. To keep from losing the values of DE$ (DECODE STRING) as we input them, they are added together to form T$ (TOTAL STRING) in line 350. If you type a 10, the computer will go around the loop ten times, stepping along the DATA line until it reaches J. It will transfer the J to T$, then *reset* the DATA pointer to the beginning so you can enter another number.

Things become even more interesting when it comes to the coding portion (changing the letters into numbers). In line 340 we used the number input as the value for A, but a letter must be input as a *string* variable. Unfortunately, the computer wouldn't understand if we said FOR X=1 TO IN$, so somehow we must change that letter into a number.

The solution to this problem is found in the very interesting function called ASC(X$). Appendix G, "ASCII Codes," will tell you the value for each of the VIC's characters. When you type in a letter or character and ask the VIC for its ASCII code, it will give you its ASCII code *number*. This is just what we want. Try this example on your VIC.

IN$="A"
PRINT ASC(IN$)

This would be the same as typing an A during the CODE portion of the program. You will get the number 65. In fact, the entire alphabet is in proper sequence from 65 (A) to 90 (Z). So now we can type a letter and receive a number to use in our loop. The only thing we need to do is change the range of our numbers. Instead of going from 65 to 90, we want to go from 27 to 56 so we can add the code values to our DATA statement right after the end of the decode values. If you check lines 230–280, you will see we just subtract 38 from the ASCII value. Now if an A is pressed, the computer takes the ASCII value (65) and subtracts 38, which gives 27. The computer will run around the loop 27 times, moving along the various DATA values, until the final time C$ becomes a 1. This is added to T$ (line 265), thus putting together a string of numbers for the letters you type.

Galactic Code

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

```
10 REM INTRODUCTION                                    :rem 231
20 PRINT "{CLR}":{2 SPACES}PRINT TAB(200)"GALACTIC
   SPACE CODE"                                          :rem 74
30 PRINT:PRINT TAB(6)"TOP SECRET"                       :rem 18
40 FOR T=1 TO 4000:NEXT                                :rem 237
50 GOSUB 500                                           :rem 122
60 PRINT:PRINT " ENTER PASSWORD"                       :rem 239
70 PRINT"{WHT}":INPUT P$                               :rem 123
80 IF P$<>"SPY CODE"THEN70                             :rem 214
90 PRINT"{BLU}"                                         :rem 89
100 REM: SELECTIONS                                    :rem 168
110 PRINT "{CLR}":PRINT "SELECT ONE"                   :rem 163
120 PRINT,,,,"A{3 SPACES}CODE"                         :rem 112
130 PRINT,,,,"B{3 SPACES}DECODE"                       :rem 251
140 PRINT,,,,"C{3 SPACES}GALACTIC CODE"                :rem 172
150 PRINT,,,,"D{3 SPACES}STOP"                         :rem 161
155 Q$=""                                              :rem 145
160 INPUT Q$                                           :rem 156
165 IF Q$="A"THEN 200                                   :rem 35
170 IF Q$="B"THEN 300                                   :rem 33
175 IF Q$="C"THEN GOSUB 500                            :rem 169
180 IF Q$="C"THEN PRINT:PRINT" PRESS RETURN":INPUT
    R$:GOTO 110                                        :rem 218
185 IF Q$="D"THEN END                                  :rem 109
```

```
190 IF Q$<>"A"OR Q$<>"B" OR Q$<>"C" OR Q$<>"D"THEN
     PRINT"{CLR}"TAB(225)"BE SERIOUS"          :rem 43
195 FOR X=1 TO 2500:NEXT:GOTO110               :rem 52
200 REM:TO CODE WORDS                          :rem 253
205 PRINT "{CLR}{DOWN}ENTER MESSAGE TO BE","CODED
     {SPACE}LIKE THIS:"                        :rem 31
210 PRINT "{2 DOWN}THIS IS A TEST"             :rem 219
215 PRINT "{2 DOWN}OR TYPE ANY NUMBER","AT RANDOM
     {SPACE}BETWEEN","WORDS, LIKE THIS:"       :rem 177
220 PRINT "{2 DOWN}THIS7IS8A9TEST              :rem 98
225 PRINT "{2 DOWN}END WITH{3 SPACES}+"        :rem 202
230 GET IN$:IF IN$=""THEN 230                  :rem 249
235 PRINT TAB(11) IN$                          :rem 108
240 IF IN$=" "THEN C$=" ":GOTO 265             :rem 136
242 IF IN$="+"THEN400                          :rem 81
245 A=ASC(IN$)                                 :rem 252
250 A=A-38                                     :rem 238
255 FOR X=1 TO A:READ C$:NEXT                  :rem 99
260 SP$=" "                                    :rem 224
265 T$=T$+C$+SP$                               :rem 78
270 RESTORE                                    :rem 189
280 GOTO 230                                   :rem 104
300 REM: DECODING                              :rem 238
305 PRINT"{CLR}":PRINT"ENTER CODED NUMBERS","PRESS
     {RVS}RETURN{OFF} AFTER"                   :rem 243
310 PRINT:PRINT"EACH COMPLETE NUMBER"          :rem 95
315 PRINT:PRINT"TYPE 0 FOR DOUBLE","SPACES"
                                               :rem 116
320 PRINT:PRINT"TYPE -1 FOR DECODE"            :rem 88
325 INPUT A:IF A= -1 THEN 400                  :rem 211
330 IF A=0 THEN DE$={2 SPACES}" ":GOTO 350     :rem 59
335 IF A>26 THEN 325                           :rem 218
340 FOR X=1 TO A                               :rem 40
345 READ DE$:NEXT                              :rem 222
350 T$=T$+DE$                                  :rem 157
355 RESTORE                                    :rem 193
360 GOTO 325                                   :rem 108
400 REM: CODING!                               :rem 135
410 FOR T=1TO 75                               :rem 77
420 PRINT "QUKWUI";                            :rem 136
430 FOR X=128 TO 255 STEP 25                   :rem 144
440 POKE 36878,5: POKE{2 SPACES}36876,X:NEXT:NEXT
                                               :rem 37
450 POKE 36878,0                               :rem 52
460 PRINT "{CLR}":PRINT "YOUR MESSAGE IS:":PRINT
                                               :rem 250
470 PRINT:PRINT T$                             :rem 103
480 T$=" "                                     :rem 149
```

```
490 PRINT"{HOME}"SPC(255)SPC(209)"PRESS RETURN"
                                            :rem 215
495 INPUT R$:GOTO 110                       :rem 173
500 REM:GALACTIC CODE                         :rem 6
510 FOR T=1 TO 9                             :rem 27
520 X=INT(RND(1)*36)+91                     :rem 246
530 POKE 36877,X+37:POKE 36878,15:POKE 36876,X+100
                                            :rem 177
540 G$=CHR$(X)                              :rem 235
550 T$=T$+G$:NEXT                           :rem 214
560 FOR R=1 TO 48                            :rem 81
570 PRINT T$;:NEXT                           :rem 85
580 T$=" ":POKE 36878,0:RETURN              :rem 133
1000 DATA A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,
     U,V,W,X,Y,Z                              :rem 6
1010 DATA 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,1
     7,18,19,20,21,22,23,24,25,26           :rem 191
```

SpeedSki

Dub Scroggin

"SpeedSki" for the unexpanded VIC takes VIC BASIC to its limits. Like most good action games, SpeedSki is easy to learn and hard to master. What's equally impressive, the program runs extremely fast and creates an excellent, realistic physical challenge. It sounds and feels like skiing—complete with jumps, trees, fences, and an ever-changing pathway.

"SpeedSki" is a fast, action game that fits in standard memory and makes full use of the VIC's color and sound capabilities. It is controlled from the keyboard and provides up to five rounds of play for one to four players, allowing each to select from any of five skill levels.

The game was designed around one central concept: speed. Every effort, short of machine language, has been used to make the game run as fast as possible without sacrificing too much realism. The result is an exciting game requiring concentration and practice. It's easy to learn the basics at skill level one, then step gradually up to level five, but mastery will take a lot of practice.

Avoid the Hazards

The object is to steer a skier through ten gates, while avoiding the hazards posed by trees and fences. The optional jumps will improve your time. The best possible time, about 29 seconds, can be achieved at skill level five by avoiding every hazard, hitting every gate, and taking every jump. But getting this best time is not easy, even for an expert. I've played the game several hundred times and have made a perfect score only a handful of times.

You should take the jumps whenever you can—they not only move you ahead, but also take you over trees you might otherwise hit, and increase your speed. Every time you hit a tree, you move up one line on the screen (to a limit of ten), and you have more time to react to the slope coming up from the bottom. You are also a little farther from the finish line. Whenever you hit a jump, you move down a line (to a limit of three below the center), so you are closer to the finish line, but you must also react faster.

Defining Characters

Line 10 prints the title, and line 20 sets the memory limits that are necessary in a program employing user-defined characters. Moving the end of memory indicators hides a section of memory from BASIC, so this section can be used for storing the user-defined character values.

Try this: PRINT FRE(X) and hit RETURN. Then type POKE 56,28: POKE 55,250: POKE 52,28: POKE 51,250 and hit RETURN. Now type PRINT FRE(X) and hit RETURN, again. You'll see the difference. BASIC has been fooled into thinking the end of its memory is closer than it really is, and you appear to have lost about 260 bytes of memory. Line 20 also sets the screen and border colors to white, like snow.

Line 30 READs X, a memory location in the protected area set up by line 20. If X is 0, all DATA has been READ, and control passes to the instructions starting in line 70. Otherwise, line 40 READs the values to be placed in X and the seven following bytes, and POKEs these values in. For instance, line 30 READs "7672". Line 40 then READs "16" and POKEs 7672 to 16. Then it READs "56" and POKEs 7673 to 56, then 7674 to 56, and so on.

Control then goes back to line 30 where the next value of X is READ in and tested. The final data step contains a 0 for the value of X following the eight values of Y. So when all the data has been read in, line 30 ends this part of the program.

Players and Skill Levels

Lines 70–90 print the directions. Note that the symbol {T} in line 70 means to press the Commodore flag key, and then hit the T to underline the title. Line 100 is used for entering the number of players and also for rejecting bad input. A value outside the allowed range passes control back to line 70; the screen is cleared, the instructions are reprinted, and you are asked for the number of players again. Line 110 accepts the number of rounds desired and rejects bad input in the same manner as line 100.

Line 120 initializes the values of R (the number of the present round) and P (the number of the present player). Lines 130–140 prompt the player skill levels, and line 150 accepts the player choice as a string variable, A$. Lines 160–200 assign values to S$ based on the skill level input, and line 210 converts A$ to the numeric variable SK. It then uses SK to establish a value for RN, which will control the number of trees printed.

The number of trees is tied to the skill level, so that the higher the skill level, the more trees there will be. If you'd like more trees, change the 1 to a larger number, but no more than 5. If SK is not an integer, or is outside the range of 1 to 5, line 210 rejects it. Moving the cursor up ten spaces and passing control back to line 130 makes it appear that the program does nothing but sit there until a correct input is given.

Speed Versus Obstacles

Line 220 establishes a new value for SK to control the speed of the program—faster for higher skill levels. Line 230 POKEs 36869 to 255 and causes the user-defined character set to be used instead of the normal set. This may cause some problems with debugging.

If an error is present after this step, the program will stop, but all you'll see on the screen will be garbage with an occasional skier or tree thrown in. If this happens, hit the CTRL and RVS keys, then type POKE 36869, 240 and RETURN. All that garbage will suddenly make sense. Line 230 also clears the screen, sets the volume, and establishes S as the noise generator.

Line 240 prints the trees on the screen for the initial setup. Each time through this loop, a random value L between 1 and 19 is calculated. Then a fence section is printed on the left, a tree is printed at TAB(L), and a fence section is printed on the right.

The initial value of B is set to 7910 in line 250. This is the location of the skier in screen memory. C is the difference between the screen map position and the color code map position. F is the POKE value for the skier figure; the POKE value will be 55 when he's going to the left and 53 when he's going to the right. The last three statements of line 250 insure that the player is not faced with the no-escape situation of having trees directly in front of him at the start of a run.

Line 260 POKEs the flags of the first gate onto the screen, and line 270 prints the level that was determined in lines 160–200. Line 280 puts the line between the flags for the first gate, and line 290 sounds the warning tones to let you know it's time to start. Just after the last tone, line 300 sets the timer. Line 310 then waits for you to press a key. If you don't hit a key for a while, that's okay, but the timer is running. You should use the time that the warning tones give you to plan your course through the first gate, and then take off as soon as the last tone sounds.

Line 320 starts the main program loop. If SK is not zero, the computer counts to SK before proceeding. This time delay, remember, is tied to the skill level to start with, but it may be reduced by hitting the jumps.

Skier Movement

If F is 55 in line 330, the skier is going left, and a track is POKEd in behind him using a POKE value of 58. If not, he's going right and the track's POKE value is 59. The track is handled in line 340.

Lines 350 and 360 are the keyboard control steps. If PEEK (197)—which is the memory location that contains the current key pressed—is 29, the key for going left has been pressed. D will later be used to produce movement to the left; F is set to the figure for going left; and S, which is the noise generator, is set to 245. If any other key is pressed, or even if no key is pressed, the skier will be going to the right, and the values needed for D, F, and S are set by line 360. You'll notice this slight change in sound when you change directions; it should sound like wind.

Gates and the Finish Line

G is incremented in line 370. If it's less than 28, control passes to line 410, because no gate or finish line is required. Otherwise, G is reset to 0 in line 380, and E, which counts the gates, is incremented. If E is 10, a finish line is printed and control passes to 460. Line 390, which causes the program to end, is executed only if the skier is past the finish line. If E is less than 10, then a random value between 2 and 11, inclusive, is calculated. A gate is then printed starting at TAB(X), X being the random number just calculated. Control then passes to line 460.

If no gate or finish line needs printing, control passes from line 370 to line 410, skipping all the above to reduce the time required for a pass through the loop. If G is 10, then line 410 prints a jump at TAB(X), X now being a random number between 4 and 13, inclusive. Fence sections are also printed at the left and right sides of the screen.

Line 420 decides whether a tree will be printed using the value of RN that was established in line 210. For skill level five, RN will have a value of .6; if a random number is more than this, no tree is printed. This means a tree will be printed roughly 60 percent of the time. For the lower skill levels, this probability is reduced so that the lower the skill level, the fewer trees there will be. If no tree is to be printed, line 440 prints only the fence sections. Otherwise, line 430 prints a tree at TAB(L), L being a random value between 1 and 19, inclusive.

If PEEK (B) in line 450 is not 32 (a blank), the skier has run into something, and control passes to line 500 to find out what the skier has run into and what to do about it.

The Illusion of Motion

Line 460 POKEs the skier's location blank, then calculates a new position by adding the value of D (determined in lines 350 and 260) to B, the skier's location. It then POKEs the appropriate figure into that location. Essentially, the skier is placed on a horizontal line on the screen and is allowed to move only back and forth on that line. However, the screen is scrolling upward beneath him, so the illusion of forward motion is created.

The movement taken care of, control passes back to line 320 for another pass through the main loop. This loop, lines 320–470, has been kept as small as possible in order to minimize the time required for each pass through it. I have tried to be very stingy with time in this section, figuring that even one instruction repeated a few hundred times adds a lot of potentially unnecessary time.

Flags and Fences

Line 500 is reached when line 450 detects that something has been struck. This entire section was originally a part of the main loop, but removing it from the loop and replacing it with the single statement in line 450 produced a significant increase in speed. Line 500 checks to see if a gate was hit. If so, it sounds a high tone to let you know you got credit for the gate, then increments H, the number of gates hit, and passes control back into the main loop.

Line 510 checks to see if a finish line was struck. If so, H is changed to the number of gates missed, the elapsed time is placed in TM, and control passes to line 640 to end the run.

If a flag was hit, line 520 sounds a low tone to let you know you were close but get no credit for the gate. Control then passes to line 570.

If a jump wasn't hit, line 530 transfers control to 570. Lines 540–560 handle the jumps. The skier is moved two spaces horizontally in the direction (D) that he was going; the value of G is stepped up to bring the next gate closer; the screen is skipped up ten spaces; and the value of SK is reduced, which results in a slight increase in speed. The skier is moved down one line on the screen unless he is already three lines below the center. Moving him further down makes seeing what is coming very difficult, but if you'd like to try it, one way is to put a larger negative value here in place of the −3. If, for instance, you put a −10, the skier will move down every time you hit a jump. Another way would be to start the skier at a lower position on the screen. This would require simply changing the initial value of B in line 250.

Line 570 checks to see if a fence section was hit. If so, it changes your direction and passes control to 610 for the sound effect. Getting out of the fence may take a couple of tries. If a tree was struck, line 580 changes the figure to a cross and passes control to line 600. Line 590 POKEs S-3 to 0 in case it was set by hitting a flag in line 520, then passes you back to the main loop.

Shaking the Screen

Line 600 causes the screen to shake a bit when you hit a tree. The inner loop here counts from 3 to 7, then from 4 to 6, and stops at 5. POKEing these values into location 36864, which controls horizontal centering, shifts the screen rapidly back and forth around the normal value of 5. Line 610 increments OS, the number of objects that have been struck, and also controls both the sound effect and the changes in color of the cross in line 580. If a tree was struck, line 620 moves the skier up a line, adjusts the value of U, and checks to see if U has reached its limit of 10. If so, the run is aborted and you are given another chance. If not, line 630 passes control back to the main loop.

Line 640, the finish line sound effect, is reached only if the finish line was detected in line 510. Lines 650–660 print out the statistics on the run just completed and finish off the sound effect. Line 660 also POKEs 36869 back to its normal state so that the scores can be printed.

Line 670 computes the player's cumulative score, adding the score for the run just completed to the total from previous rounds, and also prints the round number. Line 680 then prints the cumulative scores for all the players, and line 690 reinitializes for the next run.

Line 700 increments the player number; if the last player hasn't gone yet, control passes back to line 130 to start another run. If the last player has just gone, line 710 increments the round number and checks to see if the game is over. If not, the player number is changed to 1 and a new round is begun. Otherwise, line 720 lets you know the game is over and asks if you wish to play again. To play again, type Y and the title screen will appear.

Variable Listing

NP	Number of players
NR	Number of rounds
R	Present round number
P	Present player number
S$	Slope title
SK	Time delay factor in main loop
RN	Controls probability of a tree being printed
S	Noise generator (36877)
L	Random variable used to position trees
B	Skier's location
C	Difference between screen map and color code map
F	Skier figure
TI$	System clock
D	Direction (1 or −1) to be added to skier's location
G	Counts spaces between gates and jumps
E	Counts gates
X	Random variable used to position gates and jumps
H	Counts gates hit
TM	Elapsed time for run
U	Controls vertical movement of skier on screen
OS	Counts number of trees and fence sections struck
SC	Player's score for a run
Z(P)	Player's cumulative score where P is the player number

SpeedSki

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

```
10 PRINT"{CLR}{9 DOWN}{6 SPACES}SPEEDSKI":PRINT"
   {9 DOWN}"                                        :rem 90
20 POKE56,28:POKE55,250:POKE52,28:POKE51,250:POKE3
   6879,25                                          :rem 45
30 READX:IFX=0THEN70                                :rem 251
40 FORI=XTOX+7:READY:POKEI,Y:NEXTI:GOTO30            :rem 4
50 DATA7672,16,56,56,124,124,254,254,16             :rem 189
51 DATA7664,0,0,15,32,64,128,0,0                     :rem 71
52 DATA7656,0,0,240,4,2,1,0,0                       :rem 166
53 DATA7648,40,40,40,40,104,56,44,40                 :rem 23
54 DATA7640,32,16,136,68,34,17,8,4                  :rem 195
55 DATA7632,4,8,17,34,68,136,16,32                  :rem 197
56 DATA7624,16,28,30,28,16,16,16,56                 :rem 246
57 DATA7616,0,0,0,0,255,85,170,255                  :rem 183
58 DATA7608,16,24,126,24,26,44,72,16                 :rem 40
59 DATA7424,0,0,0,0,0,0,0,0                          :rem 57
60 DATA7592,8,24,126,24,88,52,18,8                  :rem 204
61 DATA7584,0,0,0,0,0,0,255,0                       :rem 165
62 DATA7576,8,8,28,8,62,8,127,8                      :rem 63
63 DATA7568,8,8,62,8,8,8,0,0,0                      :rem 249
70 PRINT"{CLR}{BLK}{6 SPACES}SPEEDSKI":PRINT"
   {22 T}"                                        :rem 192
80 PRINT"{UP}{BLU}YOUR SCORE IS ELAPSED TIME + 5 F
   OR EACH GATEMISSED.{2 SPACES}LOWEST SCORE WINS.
   "                                                :rem 178
90 PRINT"{DOWN}PRESS {RVS}<{OFF} TO GO LEFT
   {4 SPACES}AND {RVS}>{OFF}TO GO RIGHT." :rem 182
100 INPUT"{DOWN}NO. PLAYERS (1-4)";NP:IFNP<1ORNP>4
    THEN70                                           :rem 56
110 INPUT"{DOWN}NO. ROUNDS{2 SPACES}(1-5)";NR:IFNR
    <1ORNR>5THEN70                                  :rem 252
120 R=1:P=1                                          :rem 75
130 PRINT"{DOWN}{RVS}{CYN}SKIER #";P:PRINT"{DOWN}
    {BLU}SLOPE DESIRED":PRINT"1=BEGINNER":PRINT"2=
    INTERMEDIATE"                                   :rem 201
140 PRINT"3=ADVANCED":PRINT"4=OLYMPIC":PRINT"5=PRO
    FESSIONAL"                                      :rem 183
150 A$="":GETA$:IFA$=""THEN150                      :rem 111
160 IFA$="1"THENS$="{2 SPACES}BEGINNER"             :rem 174
170 IFA$="2"THENS$="INTERMEDIATE"                   :rem 225
180 IFA$="3"THENS$="{2 SPACES}ADVANCED"             :rem 158
190 IFA$="4"THENS$="{2 SPACES}OLYMPIC"              :rem 135
200 IFA$="5"THENS$="PROFESSIONAL"                   :rem 248
210 SK=VAL(A$):RN=(SK+1)/10:IFSK<1ORSK>5ORSK<>INT(
    SK)THENPRINT"{10 UP}":GOTO130                   :rem 195
```

```
220 SK=35-5*SK                                    :rem 1
230 POKE36869,255:PRINT"{CLR}":POKE36878,15:S=3687
    7                                             :rem 30
240 FORI=1TO22:L=INT(RND(1)*19)+1:PRINT"{RED}<";TA
    B(L);"{GRN}?";TAB(20)"{RED}<":NEXTI      :rem 161
250 B=7910:C=30720:F=55:POKEB,F:POKEB+C,3:POKEB+22
    ,32:POKEB+21,32:POKEB+23,32              :rem 186
260 POKE8125,57:POKE8131,57:POKE8125+C,4:POKE8131+
    C,4                                           :rem 8
270 PRINT"{HOME}{8 DOWN}{4 SPACES}{RVS}";S$;"
    {13 DOWN}"                                  :rem 37
280 FORI=8126TO8130:POKEI,52:POKEI+C,4:NEXTI
                                             :rem 206
290 FORI=1TO5:POKES-1,220+5*I:FORT=1TO100:NEXTT:PO
    KES-1,0:NEXTI                            :rem 218
300 TI$="000000"                             :rem 245
310 GETA$:IFA$=""THEN310                      :rem 75
320 IFSKTHENFORT=1TOSK:NEXTT                 :rem 168
330 IFF=55THENPOKEB-21,58:GOTO350            :rem 230
340 POKEB-23,59                               :rem 52
350 IFPEEK(197)=29THEND=-1:F=55:POKES,245:GOTO370
                                             :rem 171
360 D=1:F=53:POKES,246                       :rem 244
370 G=G+1:IFG<28THEN410                       :rem 59
380 G=0:E=E+1:IFE=10THENPRINT"{PUR}988888888888888
    888889":GOTO460                          :rem 155
390 IFE>10THENPOKEB,56:GOTO510                :rem 79
400 X=INT(RND(1)*10)+2:PRINTTAB(X)"{PUR}9444449":G
    OTO460                                    :rem 93
410 IFG=10THENX=INT(RND(1)*10)+4:PRINT"{UP}{RED}<"
    ;TAB(X)"{CYN}>=";TAB(20);"{RED}<"          :rem 2
420 IFRND(1)>RNTHEN440                        :rem 48
430 L=INT(RND(1)*19)+1:PRINT"{RED}<";TAB(L)"{GRN}?
    ";TAB(20)"{RED}<":GOTO450                :rem 210
440 PRINT"{RED}<";TAB(20);"<"                 :rem 65
450 IFPEEK(B)<>32THEN500                     :rem 131
460 POKEB,32:B=B+D:POKEB,F:POKEB+C,3         :rem 155
470 GOTO320                                  :rem 105
480 END                                      :rem 115
500 IFPEEK(B)=52THENH=H+1:POKES-1,240:FORT=1TO30:N
    EXTT:POKES-1,0:GOTO460                   :rem 237
510 IFPEEK(B)=56THENH=10-H:TM=INT(TI/60):POKES-1,0
    :POKEB+D,F:GOTO640                       :rem 164
520 IFPEEK(B)=57THENPOKES-3,220:GOTO570      :rem 103
530 IFPEEK(B)<>62ANDPEEK(B)<>61THEN570       :rem 248
540 POKES,253:D=D*2:G=G+10:FORI=1TO10:PRINT"{RED}<
    ";TAB(20)"{RED}<":NEXTI:IFSK>0THENSK=SK-2
                                              :rem 11
550 IFU>-3THENB=B+22:U=U-1                    :rem 26
```

```
560 GOTO460                                          :rem 110
570 IFPEEK(B)=60THENPOKEB,60:D=D*-2:GOTO600:rem 73
580 IFPEEK(B)=63THENPOKEB-22,50:POKEB,51:GOTO600
                                                     :rem 146
590 POKES-3,0:GOTO460                                :rem 233
600 FORJ=2TO0STEP-1:FORI=5-JTO5+J:POKE36864,I:NEXT
    I,J                                              :rem 100
610 OS=OS+1:FORT=0TO127:POKES,255-T:POKEB-22+C,INT
    (T/22)+2:NEXTT:POKES-1,0                          :rem 49
620 IFPEEK(B)=51THEND=-22:U=U+1:IFU=10THENPRINT"
    {RVS}{CLR}TRY AGAIN":POKE36869,240:GOTO690
                                                      :rem 77
630 GOTO460                                          :rem 108
640 POKES,0:FORT=128TO255:POKES-3,T:NEXTT:POKES-3,
    0                                                :rem 229
650 U=0:PRINT"{CLR}{RVS}OBJECTS HIT=";OS:PRINT"
    {RVS}GATES MISSED=";H:PRINT"{RVS}TIME="TM:SC=T
    M+5*H                                            :rem 214
660 PRINT"{RVS}SCORE="SC:POKES-2,220:FORT=1TO100:N
    EXTT:POKES-2,0:POKE36869,240                      :rem 63
670 Z(P)=Z(P)+SC:PRINT"{2 DOWN}{7 SPACES}{RVS}ROUN
    D";R:PRINT" ":FORI=1TONP                           :rem 9
680 PRINT"{3 SPACES}SKIER #";I;Z(I):NEXTI :rem 133
690 SC=0:G=0:E=0:OS=0:H=0:IFU=10THENU=0:POKES,0:GO
    TO130                                            :rem 128
700 P=P+1:IFP<NP+1THEN130                            :rem 226
710 R=R+1:IFR<NR+1THENP=1:GOTO130                     :rem 28
720 PRINT"{2 DOWN}{6 SPACES}{RVS}GAME OVER":PRINT"
    {2 DOWN}{BLK}PLAY AGAIN? (Y/N)"                  :rem 173
730 GETA$:IF A$="Y"THEN RUN                           :rem 11
740 IFA$<>"N"THEN 730                                :rem 100
750 END                                              :rem 115
```

Pudding Mountain Miner

Charles Brannon

Possibly the most widely distributed game ever, "Pudding Mountain Miner" is a simple, yet entertaining game that you'll find yourself playing over and over. It's fast, it's challenging, and it's easy to play. For the unexpanded VIC.

The Pudding Mountain has smothered the stockpile of gold you put aside for that proverbial rainy day. You can't dig for it; the pudding would simply shift beneath your feet and probably drag you under. The only thing you can do is use your jet aircraft to drop pudding bombs, in hopes of uncovering the treasure. It's like sinking a mine shaft, or like drilling for oil, but you don't have a steady platform. Only your jet, its pudding bombs, and your own quick reflexes.

Jet Miner

Once you've typed in "Pudding Mountain Miner" and saved it to tape or disk, enter RUN and press RETURN. You'll see a title screen and a few instructions. Once you've read them, press any key to start the game.

Your jet immediately begins flying across the screen, always appearing on the left and moving toward the right. You don't have much control; the pudding bomber moves at a constant speed, and even gains or loses altitude each time it appears. But you don't have to worry about crashing into Pudding Mountain: Your jet's autopilot makes sure you clear the pudding. The only control you have in your jet is the pudding bomb lever, activated by pressing the space bar on your VIC. Each time you press the space bar, a pudding bomb falls and digs a small hole in Pudding Mountain. You have only 20 pudding bombs on board, so use them carefully to expose the gold.

The Pudding Mountain is randomly drawn each game, sometimes with deep canyons, other times with high peaks. The gold treasure, shown by the dollar sign ($) at the bottom of the screen, is also randomly placed each game. As your jet flies over the Mountain, press the space bar to release a pudding bomb. It falls in a straight line (you don't have to worry about calculating trajectory, point of release, or anything like that) and clears a section of pudding. You need to dig a shaft to the gold by uncovering the pudding directly above the treasure. Clearing away pudding from other sections of the Mountain may be fun, but it won't help you win the game.

As you drop each bomb, the display at the top of the screen will tell you how many bombs you've used so far. Remember that you have only 20. Use them all before you uncover the gold, and the game is over. You'll also lose the game if you dig a mineshaft through the entire Mountain, including the yellow base at the bottom of the screen, and don't uncover the gold.

Once you've used all your pudding bombs, blasted a shaft through the Mountain, or uncovered the gold, you'll hear a short sound effect and an ending display will appear on the screen. If you've uncovered the gold, you'll see how many bombs you used. Whether you won or lost the game, you can play again by simply pressing RETURN.

Digging In

Pudding Mountain Miner is a game of quick reflexes. You need to press the space bar at just the right moment to accurately release the bomb and clear pudding away from above the gold. This can be especially difficult if the gold is buried near the left side of the screen, for the jet moves fast, and it seems that as soon as it appears, it's too late to drop a pudding bomb. Try pressing the space bar as soon as the jet disappears off the right side of the screen; you'll quickly get an idea of when to release a bomb to correctly dig a mineshaft.

Every time you press the space bar, a bomb is released. If you quickly press the space bar three times, the jet will drop three bombs, moving just one space to the right before releasing the next bomb. You can use this tactic to uncover the gold. If you release a bomb only one space to the left of the gold, be sure to press the space bar again, before the jet moves. That way it will immediately drop another pudding bomb, but only after the jet's moved one space to the right. You'll score a direct hit! Don't overuse this technique, however, for you have only 20 bombs. You'll quickly run out if you start dropping them two or three spaces before the gold, trying to lay a carpet of bombs.

As with most games, you'll get better with practice. And since Pudding Mountain Miner is such a quick game, you can play it again and again in a short time.

Front Page News

Pudding Mountain Miner is probably the most widely distributed game for the VIC. More copies of its program listing have been printed than any other videogame. And up to now, it hasn't appeared in any COMPUTE! publication.

The game did appear in a number of newspaper advertisements for *COMPUTE!'s Gazette* subscriptions. *USA Today, The New York Times, The Chicago Tribune,* and *The San Francisco Chronicle* all ran the ad, which included a version of Pudding Mountain Miner. The game version you see below is expanded, however, with sound, title display, and ending routine added.

Pudding Mountain Miner

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

```
100 C=38400:W=22:T=7680:S=36879:A=1:A1=100:O=200:L
    =10:M=50:B1=20:E=20                        :rem 47
110 POKES,126:PRINT"{CLR}{7 DOWN}{BLU}PUDDING MOUN
    TAIN MINER"                                :rem 254
120 PRINT "{2 DOWN}{BLK}{4 SPACES}TO DROP A BOMB"
                                               :rem 79
130 PRINT"{DOWN}{2 SPACES}PRESS THE SPACE BAR"
                                               :rem 37
140 PRINT"{3 DOWN}{RED}{3 SPACES}YOU HAVE 20 BOMBS
    "                                          :rem 171
150 PRINT"{4 DOWN}{BLU}PRESS ANY KEY TO BEGIN"
                                               :rem 48
160 GET Q$:IF Q$=""THEN 160                    :rem 113
170 C$=CHR$(147):PRINTC$:POKES,27:FORI=0TOW-1:Q=22
    *W+I:POKET+Q,160:POKEC+Q,7                 :rem 128
180 NEXT:S$=CHR$(32)+CHR$(158)+CHR$(18)+CHR$(188)+
    CHR$(146)+CHR$(156)+CHR$(185)              :rem 232
190 S$=S$+CHR$(31)+CHR$(175):Q=RND(1)*(W-7)+3+22*W
    :POKET+Q,164:POKEC+Q,0                     :rem 22
200 FORI=0TOW-1:FORJ=0TO7*RND(1)+3:Q=(21-J)*W+I:PO
    KET+Q,160:POKEC+Q,2:NEXT:NEXT              :rem 131
210 PRINT"{HOME}BOMBS USED"B                   :rem 93
220 PRINTCHR$(142);:Y%=4*RND(1)+1:FORI=1TOY%:PRINT
    :NEXT:X=0                                  :rem 146
230 IFB=20THEN290                              :rem 207
```

```
240 L$=CHR$(157):PRINTS$;L$;L$;L$;:X=X+1:GETA$:IFA
    $=""ANDX<W-4THEN240                           :rem 65
250 IFX=W-4THENPRINTTAB(X);CHR$(32);CHR$(32);CHR$(
    32);:GOTO210                                  :rem 198
260 FORI=Y%+3TO22:Q=I*W+X+1:P=PEEK(T+Q):POKET+Q-W,
    32:POKET+Q,90:POKEC+Q,8*RND(1)                :rem 158
270 IFP=32THENNEXT                                :rem 136
280 B=B+1:POKET+Q,32:IFI<22THENGOSUB370:GOTO230
                                                  :rem 132
290 IFP<>164THENFORI=0TO255:POKEC+Q,I:POKET+Q,I:NE
    XT::PRINTC$;"YOU LOST! ";B;"BOMBS"            :rem 20
300 IFP<>164THENGOSUB420                          :rem 204
310 IFB<>20ANDP<>164THEN340                       :rem 63
320 IFB=20ANDP<>164THEN PRINT"20 BOMBS IS MAXIMUM"
    :GOTO340                                      :rem 214
330 GOSUB390:FORI=1TO50:POKET+Q,32+132*F:F=1-F:NEX
    T:PRINTC$;"YOU WON! ";B;"BOMBS"               :rem 149
340 PRINT:PRINT"PRESS ";CHR$(18);"RETURN";CHR$(146
    );" TO PLAY{2 SPACES}AGAIN"                   :rem 241
350 GETA$:IFA$<>CHR$(13)THEN350                   :rem 2
360 RUN                                           :rem 142
370 POKES-2,220:FORR=15TO0STEP-2:POKES-1,R:FORV=AT
    OA1:NEXTV:NEXTR                               :rem 251
380 POKES-2,G:POKES-1,G:RETURN                    :rem 150
390 POKES-1,L+5:FORT=ATOA+4:FORZ=O-20TOO+35STEP4:P
    OKES-3,Z                                      :rem 183
400 FORM=ATOA+9:NEXTM:NEXTZ:POKES-3,G:FORU=ATOA1-5
    0:NEXTU:NEXTT:POKES-1,E                       :rem 119
410 POKES-1,G:RETURN                              :rem 2
420 POKES-A,L-5                                   :rem 96
430 FORX=O+41TOL+35STEP-A:POKES-3,X:FORT=ATOL:NEXT
    T:NEXTX                                       :rem 62
440 POKES-A,0:RETURN                              :rem 254
```

Junk Blaster

Randy Churchill

"Junk Blaster" is an exciting single player game for the unexpanded VIC. In order to get the program to fit into the unexpanded VIC, the author has chained two programs together; the first program will LOAD and RUN the second from tape or disk.

The object of "Junk Blaster" is to destroy all eight pieces of space junk while trying not to crash into them. Movement of your spaceship is achieved using a joystick. To shoot, simply press the fire button. As you play the game, you will notice that your ship, once it starts moving, keeps on drifting and is difficult to stop. Joystick control takes time to master, but the more you play, the easier it becomes. Also, a variety of sound effects adds to the excitement of this game. Once you have destroyed all eight pieces of space junk, another eight are randomly placed on the screen. The game ends when all shields are depleted.

Chaining Programs

Program 1, BLASTER1, loads in the custom characters that the game will be using. It then displays the game instructions. When you have finished reading the three screens of instructions, characters are POKEd into the keyboard buffer. When Program 1 ends, the computer acts as though you had typed in the NEW and LOAD commands. This application is known as dynamic keyboard. The NEW and LOAD commands are then executed. When the computer finds BLASTER2, it loads and runs it.

You should SAVE a copy of Program 1 before running it. Otherwise, when the program is completed, it is automatically NEWed and you will have to retype it.

Type in and SAVE Program 1 on disk or tape with the filename BLASTER1. Then type NEW and press RETURN. Type in Program 2, BLASTER2, and SAVE it using the filename BLASTER2. Tape users should be sure to SAVE BLASTER2 immediately following BLASTER1 on the same tape. Disk users should SAVE BLASTER1 and BLASTER2 on the same disk.

There is only a slight difference between the tape and disk versions of Program 1. On line 275 the device number for tape is 1 and for disk, 8. Everything else is the same. Below are the two versions of line 275; be sure to use the correct one. Remember, mistake-proof program entry can be easily achieved if you use "The Automatic Proofreader" found in Appendix J.

For tape:

```
275 PRINT"{3 DOWN}LOAD"+CHR$(34)+"BLASTER2"+CHR$(3
    4)+",1:{HOME}"                              :rem 81
```

For Disk:

```
275 PRINT"{3 DOWN}LOAD"+CHR$(34)+"BLASTER2"+CHR$(3
    4)+",8:{HOME}"                              :rem 88
```

LOADing Junk Blaster

Once you have both parts of Junk Blaster saved, simply LOAD BLASTER1 and RUN the program. Tape users should leave the play button down until BLASTER2 has finished loading.

Blaster2 is the actual game. Below is a breakdown of the different sections of the program.

Lines	
002–070	initialization
200–219	movement of junk
350–390	bullet movement
400–490	reading joystick and control section
500–555	rotation of ship
600–650	ship thrust and movement
800–830	hyperspace
900–905	subroutines
906–932	game over and high scores

Program 1. Blaster1

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

```
10 PRINT"{CLR}":POKE52,28:POKE56,28:CLR:FORI=7168T
   O7679                                       :rem 173
20 POKEI,PEEK(I+25600):NEXT:FORI=7168TO7303:READJ:
   POKEI,J:NEXT                                :rem 148
25 DATA4,77,93,255,255,93,77,4,167,67,181,56
                                               :rem 201
30 DATA24,164,192,224,24,60,231,60,24,36,66,129,12
   9,66                                        :rem 149
```

```
35 DATA60,36,36,60,66,129,,112,36,174,255,174
                                                :rem 221
40 DATA36,112,136,72,60,39,228,60,18,17,126,24,255
                                                :rem 206
45 DATA153,60,126,60,24,255,24,24,126,66   :rem 224
50 DATA66,0,0,,8,8,28,8,8,28,62,,128,80,32,86,12,2
   4                                            :rem 242
55 DATA16,,,2,38,254,38,2,,,16,24,12,86,32,80
                                                :rem 191
60 DATA128,,62,28,8,8,28,8,8,,8,24,48,106,4,10,1,,
   ,64                                           :rem 76
70 DATA100,127,100,64,,,1,10,4,106,48,24,8,,,36,24
   ,24,36,,                                      :rem 28
80 PRINT"{RED} EARTH HAS SENT YOU{3 SPACES}AND THE
    NEW FEDERATIONSTARSHIP ORION TO             :rem 209
85 PRINT"CLEAR OUT THE JUNK{4 SPACES}BELT AROUND V
   EGA.                                         :rem 213
90 PRINT" THE ORION HAS FOUR{3 SPACES}SHIELDS.EACH
    TIME YOU GET HIT YOU LOSE ONE               :rem 230
100 PRINT"SHIELD AND START BACK IN THE CENTER OF T
    HE{2 SPACES}PLAY FIELD.                     :rem 226
110 PRINT" EACH PIECE OF SPACE{2 SPACES}JUNK YOU D
    ESTROY WILL GIVE YOU 10 POINTS.             :rem 228
120 PRINT" EARTH WILL SUPPLY YOUWITH A BONUS SHIEL
    D{3 SPACES}EVERY 150 POINTS.                :rem 193
130 PRINT:PRINT" PRESS ANY KEY                  :rem 104
140 GETA$:IFA$=""THEN140                         :rem 77
150 PRINT"{CLR}{BLU} WHEN THE GAME STARTS,YOURS IS
     THE SHIP IN{2 SPACES}THE CENTER OF THE PLAYFI
    ELD.                                        :rem 251
160 PRINT" TO MOVE,PUSH JOYSTICKFORWARD FOR A SECO
    ND.{2 SPACES}TO TURN,PUSH STICK{3 SPACES}SIDEW
    AYS.                                        :rem 144
171 PRINT" TO TURN AND THRUST,{2 SPACES}PUSH THE J
    OYSTICK DIA-GONALLY.                        :rem 237
175 PRINT" TO HYPERSPACE,PULL{3 SPACES}THE JOYSTIC
    K BACK.                                     :rem 220
180 PRINT" TO STOP MOVING,TURN{2 SPACES}IN OPPOSIT
    E DIRECTION AND THRUST A MOMENT.             :rem 29
190 PRINT" TO FIRE,POINT SHIP ATTHE PIECE OF JUNK
    {SPACE}AND PRESS FIRE BUTTON.                :rem 53
200 PRINT:PRINT" PRESS ANY KEY                  :rem 102
210 GETA$:IFA$=""THEN210                         :rem 73
220 PRINT"{CLR}{PUR} IF THE ORION GOES{4 SPACES}HY
    PERSPACE OVER FIVE{2 SPACES}TIMES IT WILL LOSE
     A{2 SPACES}SHIELD.                         :rem 163
230 PRINT" THEN IT WILL RESTART IN THE{2 SPACES}CE
    NTER WITH{3 SPACES}HYPERSPACES RESTORED
    {2 SPACES}TO FIVE.                          :rem 140
```

```
240 PRINT" THE{2 SPACES}NUMBERS ON{2 SPACES}THE TO
    P{2 SPACES}ARE{2 SPACES}YOUR SCORE,     :rem 130
245 PRINT"SHIELDS{2 SPACES}LEFT , HYPER-SPACES LEF
    T , AND THE HIGHEST OVERALL SCORE.     :rem 247
250 PRINT:PRINT:PRINT" PRESS ANY KEY        :rem 50
260 GETA$:IFA$=""THEN260                    :rem 83
270 PRINT"{CLR}NEW"                        :rem 231
275 PRINT"{3 DOWN}LOAD"+CHR$(34)+"BLASTER2"+CHR$(3
    4)+",1:{HOME}":REM DISK USERS CHANGE ,1 TO ,8
                                            :rem 54
280 POKE631,13:POKE632,131:POKE198,2        :rem 33
```

Program 2. Blaster2

```
2 DATA-22,8,-21,15,1,14,23,13,22,12,21,11,-1,10,-2
  3,9,1,484,23,484,22,483,21,484            :rem 80
5 FORT=0TO7:READB%(T):READC%(T):F%(T)=T:NEXT:FORT=
  0TO3:READD%(T):READE%(T):NEXT            :rem 202
30 P=36875:DEFFNA(X)=INT(RND(TI)*X):POKEP+4,8:POKE
   P+3,15                                  :rem 118
33 POKEP-6,255:L=7702:M=8185:POKE37154,127:rem 112
35 O=7955:Q=484:PRINT"{CLR}":SH=4:SC=0:D=O:POKED,8
   :I=0:H=8:E=-22:HY=0:J=0:K=0:GOSUB904     :rem 40
40 N=0:FORT=0TO7                           :rem 219
60 A(T)=FNA(Q)+L:IFA(T)=DTHEN60             :rem 60
70 NEXT                                    :rem 166
200 IFSH<1THENPOKEP+2,0:POKEP,0:PRINT"{CLR}":GOTO9
    10                                      :rem 80
201 IFN=8THEN40                            :rem 120
202 FORT=0TO3:IFA(T)=0THENNEXT:GOTO210     :rem 162
204 POKEA(T),32:A(T)=A(T)-D%(T):IFA(T)<LTHENA(T)=A
    (T)+E%(T)                               :rem 99
205 IFPEEK(A(T))=32THEN209                 :rem 238
206 IFA(T)=DTHENGOSUB900:NEXT:GOTO210       :rem 85
207 IFA(T)=FTHENGOSUB902:NEXT:GOTO210       :rem 90
209 POKEA(T),F%(T):NEXT                    :rem 101
210 FORT=4TO7:IFA(T)=0THENNEXT:GOTO350     :rem 174
212 POKEA(T),32:A(T)=A(T)+D%(T-4):IFA(T)>MTHENA(T)
    =A(T)-E%(T-4)                           :rem 39
213 IFPEEK(A(T))=32THEN219                 :rem 238
214 IFA(T)=DTHENGOSUB900:NEXT:GOTO350       :rem 89
215 IFA(T)=FTHENGOSUB902:NEXT:GOTO350       :rem 94
219 POKEA(T),F%(T):NEXT                    :rem 102
350 IFC=0THEN400                           :rem 154
355 POKEF,32                               :rem 163
356 F=F+G:IFF<LTHENF=F+Q                   :rem 228
360 IFF>MTHENF=F-Q                         :rem 111
365 IFPEEK(F)=32THEN377                     :rem 91
370 FORT=0TO7:IFF=A(T)THENGOSUB902:GOTO400:rem 159
```

```
375 NEXT                                          :rem 222
377 IFR=1THENR=Ø:GOTO355                          :rem 238
38Ø C=C-1:IFC<1THENC=Ø:G=Ø:POKEF,32:F=Ø:GOTO4ØØ
                                                   :rem 54
39Ø POKEF,16                                      :rem 164
4ØØ IFC=ØTHENB=PEEK(37137)AND32:IFB=ØTHEN4Ø3
                                                  :rem 114
4Ø2 GOTO41Ø                                       :rem 1ØØ
4Ø3 IFJ=ØTHENC=8:F=D:G=E:GOTO356                  :rem 217
4Ø4 C=8:F=D+E:G=E:GOTO9Ø6                         :rem 214
41Ø POKEP,Ø:A=(PEEK(37137)AND28)OR(PEEK(37152)AND1
    28)                                             :rem 9
42Ø A=ABS((A-1ØØ)/4)-7                             :rem 82
43Ø IFA/2><INT(A/2)THENGOSUB641                   :rem 1Ø6
45Ø ONAGOTO475,485,46Ø,,465,47Ø,475,,,,48Ø,49Ø,475
                                                   :rem 78
46Ø I=I-1:POKEP+2,253:GOSUB5ØØ:GOTO2ØØ            :rem 247
465 POKEP+2,Ø:GOTO8ØØ                             :rem 226
47Ø POKEP+2,255:GOSUB6ØØ:GOTO2ØØ                  :rem 148
475 POKEP+2,Ø:GOTO2ØØ                             :rem 221
48Ø I=I+1:POKEP+2,253:GOSUB5ØØ:GOTO2ØØ            :rem 247
485 I=I-1:POKEP+2,235:GOSUB5ØØ:GOSUB6ØØ:GOTO2ØØ
                                                   :rem 78
49Ø I=I+1:POKEP+2,235:GOSUB5ØØ:GOSUB6ØØ:GOTO2ØØ
                                                   :rem 72
5ØØ IFI<ØTHENI=7                                  :rem 197
5Ø5 IFI>7THENI=Ø                                  :rem 2Ø4
55Ø IFJ=ØTHENK=I                                  :rem 224
552 E=B%(I):H=C%(I):IFRTHENRETURN                  :rem 1Ø
555 POKED,H:RETURN                                :rem 16Ø
6ØØ IFE=-JTHENJ=Ø:GOTO641                          :rem 18
6Ø2 IFK-I=2ORK-I=3THENK=K-1:GOTO64Ø               :rem 15Ø
6Ø4 IFK-I=-2ORK-I=-3THENK=K+1:GOTO64Ø             :rem 24Ø
61Ø IFI=ØANDK=6ORI=6ANDK=ØORI=5ANDK=ØORI=1ANDK=6TH
    ENK=7:GOTO64Ø                                  :rem 32
614 IFI=7ANDK=1ORI=1ANDK=7ORI=6ANDK=1ORI=2ANDK=7TH
    ENK=Ø:GOTO64Ø                                  :rem 37
618 IFI=ØANDK=5THENK=6:GOTO64Ø                    :rem 11Ø
628 IFI=7ANDK=2THENK=1:GOTO64Ø                    :rem 11Ø
638 K=I                                           :rem 114
64Ø J=B%(K)                                        :rem 36
641 POKED,32:D=D+J:IFD<LTHEND=D+Q                  :rem 24
642 IFD>MTHEND=D-Q                                :rem 1Ø8
644 IFPEEK(D)=32THEN65Ø                            :rem 83
646 FORT=ØTO7:IFD=A(T)THENGOSUB9ØØ:GOTO65Ø:rem 168
648 NEXT                                          :rem 225
65Ø POKED,H:RETURN                                :rem 156
8ØØ POKED,32:D=FNA(Q)+L:I=FNA(8):R=1:J=Ø:GOSUB55Ø:
    R=Ø:W=FNA(8Ø)+128:IFPEEK(D)=32THEN822 :rem 173
```

```
811 FORT=ØTO7:IFD=A(T)THENGOSUB9ØØ:GOTO822:rem 163
812 NEXT                                   :rem 218
822 HY=HY+1:IFHY>4THENSH=SH-1:GOSUB9Ø4:HY=Ø
                                           :rem 242
83Ø GOSUB9Ø4:POKEP,W:POKED,H:GOTO2ØØ        :rem 26
9ØØ POKED,32:D=O:SH=SH-1:HY=Ø:GOSUB9Ø4:I=Ø:H=8:E=-
    22:POKED,H:J=Ø:POKEA(T),32:A(T)=Ø        :rem 4
9Ø1 K=Ø:N=N+1:RETURN                       :rem 219
9Ø2 POKEF,32:SC=SC+1Ø:Z=1:GOSUB9Ø4:G=Ø:C=Ø:F=Ø:POK
    EA(T),32:A(T)=Ø:N=N+1:RETURN           :rem 228
9Ø4 IFZ=1THENIFINT(SC/15Ø)=SC/15ØTHENSH=SH+1
                                           :rem 223
9Ø5 PRINT"{HOME}"SC;SH;HY;S1:Z=Ø:RETURN    :rem 164
9Ø6 IFF<LTHENF=F+Q                         :rem 112
9Ø7 R=1:GOTO36Ø                            :rem 1Ø8
91Ø POKEP-6,24Ø:IFSC=ØTHEN93Ø              :rem 213
911 IFSC<S1THEN914                          :rem 77
912 D$=C$:C$=B$:S3=S2:S2=S1:S1=SC:INPUT"WHAT IS YO
    UR NAME";A$                            :rem 1Ø9
913 B$=LEFT$(A$,6)+" HAS":GOTO93Ø            :rem 1
914 IFSC<S2THEN917                          :rem 84
915 D$=C$:S3=S2:INPUT"WHAT IS YOUR NAME";A$:S2=SC
                                           :rem 173
916 C$=LEFT$(A$,6)+" HAS":GOTO93Ø            :rem 5
917 IFSC<S3THEN93Ø                          :rem 83
918 S3=SC:INPUT"WHAT IS YOUR NAME";A$:D$=A$+" HAS"
                                           :rem 12Ø
919 D$=LEFT$(A$,6)+" HAS"                  :rem 25Ø
929 PRINT"{CLR}"TAB(139)"HI SCORES"        :rem 113
93Ø PRINT:PRINTB$,S1,C$,S2,D$,S3:PRINT"HIT ANY KEY
     FOR A NEW GAME"                       :rem 125
931 GETA$:IFA$=""THEN931                    :rem 93
932 PRINT"{CLR}":GOTO3Ø                    :rem 216
```

Chapter 3

Applications

Budget Planner

Charles B. Silbergleith

This home budget program allows you to keep track of various household expenses and calculate totals quickly and easily (at least 8K memory expansion required).

In the dark days prior to automation, I would plan my budget by writing all my month's expenses on a sheet of paper, adding items, and adjusting amounts as I received a bill.

This process worked very well except for the number of revisions necessary for revolving credit accounts such as credit cards. Every time one of the item amounts changed, the grand total changed and needed to be recalculated. That was messy.

I decided to write a program which allowed me to make a list of my monthly expenses, to change amounts, and which provided a grand total of all items. I also wanted the program to save this list to tape or disk and recall it.

What was produced was a program that allowed me to maintain a list of expense items, add new items, change amounts, and delete items. And it would quickly sort and sum all the amounts. This was useful in seeing whether new expenses could be incurred (could I really afford that new disk drive or not?), or whether bill consolidation would help.

Program Operation

First, here are some basic characteristics of the program before I discuss how to use it. The list allows for entries of ten characters (maximum) per item and amounts of up to 9999.99. The list will be sorted, a total calculated over all item amounts, and the options menu displayed at the end of an add, update, or delete modification to the list. The sort is done by item name. You will be repeatedly prompted for the next add, update, or delete to the list until you type *END to one of the prompts for input. In fact, any function will terminate whenever you respond with an *END to a prompt.

The program uses the special function keys f1 through f8. Described below are the functions:

f1 Display Expense List. This function displays the list and a total of all item amounts at the bottom of the screen. Pressing f1 will display the next 20 items, and the cursor up and down keys scroll the list vertically. All function keys are available.

f2 **Add New Expense to the List.** This allows you to add a new item to the list. The program will not check for duplicates. However, it's simple enough to change or delete an item if you mistakenly duplicate one. Names are up to ten characters, and amounts should not be larger than + or − 9999.99. These restrictions prevent the screen display from overlapping, wrapping around, or otherwise messing up on the 22-column VIC. Type *END to return to the menu screen.

f3 **Expense List Update.** The screen lists a number next to each item. This number is the item's index. Use this number for the ITEM # prompt. The item will be displayed and a new name or amount may be entered, replacing the old data. Pressing the RETURN key without data when prompted for an ITEM NAME or AMT will leave the current data intact. Again, type *END to return to the menu.

f4 **Save the List on Tape or Disk.** The program asks for a FILENAME. This should be any name that follows normal Commodore filenaming conventions. This is the filename SAVEd on tape or disk. Remember it.

f5 **Delete Items from the List.** The START AT and END AT prompts allow a block of items to be deleted by putting the starting and ending index numbers in the appropriate places. Leaving out the ending index will delete only the starting index number's item. Type *END when prompted for the starting index number to return to the main menu.

f6 **Display the Option Menu.** Function keys and their associated functions are displayed. See program lines 6030–6100 for details.

f7 **Load or Merge a List.** A previously saved list can be loaded into memory or a list on tape can be merged with a list in memory. For the merge, an item on tape is compared to the items in memory, and if the item names match, their amounts are averaged together and replace the previous amount. If the item doesn't match, the item is added to the list.

f8 **End of Program.** This function allows you to first save the list before actually ending the program—handy if you've forgotten to save the list before.

Technical Notes

The program is written using the modular concept of structured programming. This means that the program is written in order to isolate its various tasks. Common routines are separate from the routines that use them and are accessed by GOSUB statements.

The main routine (lines 200–299) calls various subfunctions at the user's request. A request to display the list (f1) calls a subroutine at lines 1000–1999; update (f3) calls lines 3000–3999, etc. Notice that each function key corresponds to a range of 1000 line numbers—f1 is lines 1000–1999; f2 is lines 2000–2999; f3 is lines 3000–3999, etc. This makes it easier to remember where things are in the program.

In addition, two utilities are included as separate modules for use by any function. These are the bubble sort, lines 500–599, and an accumulator, lines 300–399.

GOTO statements are kept to a minimum and are used only for branching within subroutines. While certain advocates of structured programming insist on GOTO-less code, I find it sometimes more cumbersome to eliminate all of them than to use a few. Again, the word to remember is *few*.

One last note. The variable SZ (line 20) controls the number of items that can be listed. Naturally, the more items on the list, the more memory is required. Since the computer will consume more memory as needed when the program runs, it is possible to make this variable too large and run out of room while working with the program. As an exercise, I suggest you add a function which will display the amount of memory left. Use the ? key to invoke it. I think you'll find it fairly easy to do, given the way the program is organized.

Disk Or Tape?

"Budget Planner" was originally written to be used with tape. Program 1 is the version that tape users should enter. Disk users should also enter Program 1, but replace the lines in Program 1 with those in Program 2.

As with all programs in this book, it is best to use "The Automatic Proofreader" (Appendix J) for perfect program entry.

Program 1. Budget Planner

```
10 REM DEF VARIABLES                              :rem 173
20 SZ=10Ø:I=-19                                   :rem 52
3Ø R$=CHR$(13):TA=Ø                               :rem 8
4Ø DIM A$(SZ),AE(SZ)                              :rem 81
5Ø DEFFNRN(X)=INT(X*1ØØ+.5)/1ØØ                   :rem 235
2ØØ REM MAINROUTINE                               :rem 193
21Ø GOSUB6ØØØ                                     :rem 217
22Ø Z$="":GETZ$:IFZ$=""THENGOTO22Ø                :rem 239
23Ø IFZ$=CHR$(133)THENI=I+2Ø:GOSUB1ØØØ            :rem 206
235 IFZ$=CHR$(134)THENGOSUB3ØØØ                   :rem 64
24Ø IFZ$=CHR$(135)THENGOSUB5ØØØ                   :rem 63
245 IFZ$=CHR$(136)THENGOSUB7ØØØ                   :rem 71
25Ø IFZ$=CHR$(137)THENGOSUB2ØØØ                   :rem 63
255 IFZ$=CHR$(138)THENGOSUB4ØØØ                   :rem 71
26Ø IFZ$=CHR$(139)THENGOSUB6ØØØ                   :rem 7Ø
265 IFZ$=CHR$(14Ø)THENGOSUB8ØØØ                   :rem 69
27Ø IFZ$=CHR$(17)THENI=I-1:GOSUB1ØØØ              :rem 116
275 IFZ$=CHR$(145)THENI=I+1:GOSUB1ØØØ             :rem 169
299 GOTO22Ø                                       :rem 113
3ØØ REM ACCUM TOTALS                              :rem 183
31Ø TA=Ø                                          :rem 15Ø
32Ø FOR J=1TOMX                                   :rem 124
33Ø TA=TA+AE(J)                                   :rem 73
34Ø NEXTJ                                         :rem 32
399 RETURN                                        :rem 133
4ØØ REM LOAD FILES                                :rem 11
41Ø INPUT"FILE NAME";F$                           :rem 79
42Ø IFF$="*END"THENGOSUB6ØØØ:RETURN               :rem 16Ø
45Ø OPEN1,1,Ø,F$                                  :rem 75
455 PRINT"{RVS}{GRN}FOUND{OFF}{BLK}";F$           :rem 226
46Ø INPUT#1,MX                                    :rem 79
47Ø FORJ=1TOMX                                    :rem 13Ø
48Ø INPUT#1,Y,A$(J),AE(J)                         :rem 126
49Ø NEXTJ                                         :rem 38
495 CLOSE1                                        :rem 73
499 RETURN                                        :rem 134
5ØØ REM SORT BY NAME                              :rem 125
5Ø5 IFMX=1THENGOTO599                             :rem 75
51Ø PRINT"{2 DOWN}{5 RIGHT}{RVS}SORTING{OFF}"
                                                  :rem 228
52Ø FORJ=1TOMX-1                                  :rem 22Ø
53Ø FORK=J+1TOMX                                  :rem 245
54Ø IFA$(K)>A$(J)THENGOTO59Ø                      :rem 1Ø9
55Ø SM$=A$(K):SM=AE(K)                            :rem 213
56Ø A$(K)=A$(J):AE(K)=AE(J)                       :rem 147
57Ø A$(J)=SM$:AE(J)=SM                            :rem 213
59Ø NEXTK                                         :rem 4Ø
595 NEXTJ                                         :rem 44
```

```
599 RETURN                                        :rem 135
1000 REM DISPLAY                                  :rem 187
1010 IF(I<1)OR(I>MX)THENI=1                       :rem 92
1020 PRINT"{CLR} #"TAB(5)"{CYN}EXPENSES{BLK}"TAB(1
     6)"{BLU}AMT{BLK}"                            :rem 233
1030 FORJ=ITOI+19                                 :rem 252
1040 IFJ>MXTHENPRINT" ":GOTO1080                  :rem 189
1050 PR$=STR$(AE(J)+.001):PR$=MID$(PR$,2,(LEN(PR$)
     -2))                                         :rem 196
1060 IFAE(J)=0THENPR$="0.00"                      :rem 24
1065 J$=MID$(STR$(J),2)                           :rem 220
1070 PRINTTAB(3-LEN(J$))J$;TAB(4)A$(J)TAB(21-LEN(P
     R$))PR$                                      :rem 244
1080 NEXTJ                                        :rem 82
1090 TA$=STR$(TA+.001)                            :rem 173
1100 TA$=LEFT$(TA$,LEN(TA$)-1)                    :rem 132
1110 IFTA=0THENTA$="0.00"                         :rem 123
1120 PRINT"{CYN}TOTAL {BLK}"TA$                   :rem 1
1999 RETURN                                       :rem 188
2000 REM ADD NEW                                  :rem 89
2010 R=MX+1:N$="":E1$=""                          :rem 213
2020 PRINT"{CLR}{3 RIGHT}ADD NEW EXPENSES":rem 157
2030 PRINT"{DOWN}{12 RIGHT}ITEM #";R              :rem 226
2040 INPUT"{DOWN}ITEM NAME ";N$                   :rem 168
2050 IFN$="*END"THENGOTO2999                      :rem 143
2055 IFLEN(N$)>10THENN$=LEFT$(N$,10)              :rem 25
2060 A$(R)=N$                                     :rem 127
2070 INPUT"{DOWN}ITEM AMT{2 SPACES}";E1$          :rem 148
2080 IFE1$="*END"THENGOTO2999                     :rem 186
2085 IFVAL(E1$)=0THENAE(R)=0:GOTO2100             :rem 148
2090 AE(R)=FNRN(VAL(E1$))                         :rem 132
2095 IFAE(R)>9999.99THENAE(R)=9999.99             :rem 99
2100 MX=MX+1                                      :rem 166
2110 GOTO2010                                     :rem 192
2200 MX=MX+1                                      :rem 167
2999 GOSUB500:GOSUB300:GOSUB6000:RETURN           :rem 217
3000 REM UPDATE                                   :rem 106
3010 PRINT "{CLR}{BLU}EXPENSE ";"{RVS}UPDATE{OFF}
     {BLK}"                                       :rem 213
3020 INPUT"{DOWN}ITEM# ";P1$                      :rem 220
3025 IFP1$="*END"THENGOTO3999                     :rem 198
3026 IF(VAL(P1$)=0)OR(VAL(P1$)<1)THENPRINT"
     {2 DOWN}{4 RIGHT}{RED}{RVS}INPUT ERROR{OFF}
     {BLK}":GOTO3020                              :rem 97
3027 P=INT(VAL(P1$))                              :rem 110
3030 N$="":E1$=""                                 :rem 14
3040 IFP>SZTHENPRINT"MAX EXCEEDED":P=SZ:MX=P
                                                  :rem 142
3050 IFP>MXTHENMX=P                               :rem 235
```

```
3Ø6Ø PR$=STR$(AE(P)+.ØØ1):PR$=MID$(PR$,2,(LEN(PR$)
     -2))                                   :rem 2Ø5
3Ø65 IFAE(P)=ØTHENPR$="Ø.ØØ"                 :rem 37
3Ø7Ø PRINTP;TAB(4)A$(P)TAB(21-LEN(PR$))PR$:rem 184
3Ø8Ø INPUT"{DOWN}ITEM NAME";N$              :rem 173
3Ø9Ø IFN$="*END"THENGOTO3999                :rem 149
31ØØ IFN$<>""THENA$(P)=N$                   :rem 1Ø3
31Ø5 IFLEN(A$(P))>1ØTHENA$(P)=LEFT$(A$(P),1Ø)
                                            :rem 21Ø
311Ø INPUT"AMT ";E1$                         :rem 8Ø
312Ø IFE1$="*END"THENGOTO3999               :rem 183
3125 IFE1$=""GOTO3Ø1Ø                       :rem 114
313Ø IF(VAL(E1$)=Ø)AND(E1$<>"Ø")THENPRINT"{2 DOWN}
     {3 RIGHT}{RVS}{RED}INPUT ERROR{OFF}{BLK}":GOT
     O311Ø                                  :rem 169
3135 IFVAL(E1$)=ØTHENAE(P)=Ø:GOTO38ØØ       :rem 151
314Ø AE(P)=FNRN(VAL(E1$))                   :rem 127
315Ø IFAE(P)>9999.99THENAE(P)=9999.99        :rem 88
38ØØ GOTO3Ø1Ø                               :rem 2ØØ
3999 GOSUB5ØØ:GOSUB3ØØ:GOSUB6ØØØ:RETURN     :rem 218
4ØØØ REM SAVE FILE                          :rem 247
4Ø1Ø PRINT"{CLR}{3 RIGHT}SAVE EXPENSE LIST" :rem 3
4Ø2Ø INPUT"{2 DOWN}FILE NAME";F$            :rem 162
4Ø3Ø IFF$="*END"THENGOSUB6ØØØ:RETURN        :rem 2Ø9
4Ø5Ø OPEN1,1,1,F$                           :rem 124
4Ø6Ø PRINT#1,MX                             :rem 124
4Ø7Ø FORJ=1TOMX                             :rem 178
4Ø8Ø PRINT#1,J;R$;A$(J)R$;AE(J);R$          :rem 146
4Ø9Ø NEXTJ                                   :rem 86
41ØØ CLOSE1                                 :rem 1Ø8
4999 GOSUB6ØØØ:RETURN                        :rem 63
5ØØØ REM DELETE                              :rem 92
5ØØ5 DT=Ø:TM=Ø                               :rem 23
5Ø1Ø PRINT"{CLR}{8 RIGHT}DELETE"            :rem 197
5Ø2Ø S1$=""                                 :rem 24Ø
5Ø3Ø INPUT"{2 DOWN}START AT";S1$            :rem 196
5Ø4Ø IFS1$="*END"THENGOTO59ØØ               :rem 184
5Ø5Ø DS=INT(VAL(S1$))                       :rem 182
5Ø6Ø S1$=""                                 :rem 244
5Ø7Ø IFDS=ØTHENPRINT"{DOWN}{6 RIGHT}{RVS}{RED}INPU
     T ERROR{OFF}{BLK}":GOTO5Ø2Ø            :rem 194
5Ø8Ø S1$=""                                 :rem 246
5Ø9Ø INPUT"{2 DOWN}END AT";S1$               :rem 19
51ØØ IFS1$="*END"THENGOTO59ØØ               :rem 181
511Ø IFS1$=""ORS1$="Ø"THENDE=Ø:GOTO52ØØ     :rem 216
512Ø DE=INT(VAL(S1$))                       :rem 166
5125 IFDE>MXTHENDE=MX                        :rem 98
513Ø IFDE=>DSTHENGOTO52ØØ                    :rem 34
5135 PRINT"{2 DOWN}{2 RIGHT}{RVS}{PUR}Ø OR NUMBER
     {SPACE}GREATER"                         :rem 77
```

```
5140 PRINT"{2 DOWN}{2 RIGHT}THAN{OFF}{RED}";DE;"
     {RVS}{PUR}REQUIRED"                          :rem 34
5150 GOTO5Ø8Ø                                    :rem 2Ø9
52ØØ IFDE=ØTHENDE=DS                             :rem 216
52Ø5 TM=DE-DS+1                                   :rem 83
52Ø7 DT=DT+TM                                      :rem 7
521Ø FORJ=DSTODE                                 :rem 249
522Ø A$(J)="[9 B]":AE(J)=Ø                       :rem 2Ø1
523Ø NEXTJ                                        :rem 83
524Ø GOTO5Ø1Ø                                    :rem 2Ø2
59ØØ GOSUB5ØØ                                    :rem 227
591Ø MX=MX-DT                                     :rem 27
5999 GOSUB3ØØ:GOSUB6ØØØ:RETURN                   :rem 141
6ØØØ REM OPTIONS MENU                             :rem 11
6Ø1Ø PRINT"{CLR}{7 RIGHT}{PUR}OPTIONS:{BLK}"
                                                 :rem 136
6Ø2Ø PRINT"{7 RIGHT}{YEL}========{BLK}"          :rem 122
6Ø3Ø PRINT"{DOWN}{RVS}{RED}F1{OFF}{BLK}-DISPLAY EX
     PENSES"                                      :rem 32
6Ø4Ø PRINT"{DOWN}{RVS}{RED}F2{OFF}{BLK}-ADD NEW EX
     PENSES"                                     :rem 191
6Ø5Ø PRINT"{DOWN}{RVS}{RED}F3{OFF}{BLK}-UPDATE EXP
     ENSE LST"                                   :rem 113
6Ø6Ø PRINT"{DOWN}{RVS}{RED}F4{OFF}{BLK}-SAVE EXPEN
     SE LIST"                                     :rem 4Ø
6Ø7Ø PRINT"{DOWN}{RVS}{RED}F5{OFF}{BLK}-DELETE FRO
     M LIST"                                     :rem 2Ø2
6Ø8Ø PRINT"{DOWN}{RVS}{RED}F6{OFF}{BLK}-OPTIONS SC
     REEN"                                       :rem 149
6Ø9Ø PRINT"{DOWN}{RVS}{RED}F7{OFF}{BLK}-LOAD/MERGE
      FILES"                                     :rem 221
61ØØ PRINT"{DOWN}{RVS}{RED}F8{OFF}{BLK}-END"
                                                 :rem 123
6999 RETURN                                      :rem 193
7ØØØ REM LOAD/MERGE                              :rem 1Ø6
7Ø1Ø PRINT"{CLR}{6 RIGHT}LOAD/MERGE"             :rem 153
7Ø2Ø PRINT"{DOWN}{5 RIGHT}EXPENSE FILES"         :rem 199
7Ø3Ø INPUT"LOAD OR MERGE (L/M)";AN$              :rem 214
7Ø4Ø IFAN$="L"THENMX=Ø:GOSUB4ØØ:GOTO7999         :rem 19Ø
7Ø5Ø IFAN$="*END"THENGOSUB6ØØØ:RETURN             :rem 31
7Ø6Ø IFAN$<>"M"GOTO7Ø3Ø                           :rem 29
7Ø7Ø PRINT"{DOWN}{4 RIGHT}MERGE"                 :rem 148
7Ø8Ø INPUT"{DOWN}FILE NAME";F$                   :rem 154
7Ø9Ø IFF$="*END"THENGOSUB6ØØØ:RETURN             :rem 218
712Ø OPEN1,1,Ø,F$                                :rem 124
713Ø INPUT#1,T1                                   :rem 96
714Ø FORT2=1TOT1                                 :rem 2Ø7
715Ø INPUT#1,Y,T3$,T4                            :rem 193
716Ø FORJ=1TOMX                                  :rem 181
```

```
7170 IFA$(J)=T3$THENAE(J)=INT(((AE(J)+T4)/2)*100)/
     100:T3$=""                                :rem 199
7180 NEXTJ                                      :rem 89
7190 IFT3$<>""THENMX=MX+1:A$(MX)=T3$:AE(MX)=T4
                                               :rem 211
7200 NEXT                                        :rem 8
7210 CLOSE1                                    :rem 113
7999 GOSUB500:GOSUB300:GOSUB6000:RETURN        :rem 222
8000 REM END OF JOB                            :rem 243
8010 PRINT"{CLR}{4 RIGHT}END OF PROGRAM{2 DOWN}"
                                                :rem 71
8020 PRINT"WOULD YOU LIKE TO SAVE (Y/N)":INPUT AN$
                                               :rem 190
8030 IFAN$="*END"THENGOSUB6000:RETURN           :rem 30
8040 IFAN$="N"THENGOTO8060                      :rem 19
8050 GOSUB4000                                  :rem 17
8060 PRINT"{CLR}THANK YOU"                     :rem 165
8070 PRINT"{13 RIGHT}END"                      :rem 240
8080 END                                       :rem 167
```

Program 2. Make these changes if you are using a disk drive.

```
450 OPEN15,8,0,F$                              :rem 135
460 INPUT#15,MX                                :rem 132
480 INPUT#15,Y,A$(J),AE(J)                     :rem 179
495 CLOSE 15                                   :rem 126
4050 OPEN15,8,1,F$                             :rem 184
4060 PRINT#15,MX                               :rem 177
4080 PRINT#15,J;R$;A$(J)R$;AE(J);R$            :rem 199
7120 OPEN15,8,0,F$                             :rem 184
7130 INPUT#15,T1                               :rem 149
7150 INPUT#15,Y,T3$,T4                         :rem 246
7210 CLOSE15                                   :rem 166
```

Perpetual Calendar

Robert Lewis

Have you ever wanted to see a calendar for next year? Last year? Or the year you were born? "Perpetual Calendar" is able to produce a correct calendar for any year after 1582, the year our present calendar began, on an unexpanded VIC.

This program is able to produce a *correct* calendar for any year after 1900. There are other starting years in program lines 10–60. These years are 1950, 1800, and 1582, the year today's calendar was started.

To change the starting year, delete the REM statement from the year wanted *and* the next following line. Be sure to add REM statements to the old lines, or you may get an error when the program is run.

To use the program, simply INPUT the full year (example: 1984) when asked YEAR WANTED? After entering the date, the program uses a loop to find what day of the week January 1 falls on in that year. A large difference between the entered and starting years may cause a delay of several seconds.

Leap Years

Leap years present a very special problem for a program like this. A leap year is any year that is divisible by 4—unless the year is also divisible by 100. There is one more complication: Years divisible by 100 but also divisible by 400 *are* leap years. In other words, 1700, 1800, 1900, 2100, 2200 are *not* leap years; 1600, 1984, 1992, 2000, 2400 *are* leap years.

When the program leaves the loop that finds the day of the week for January 1, it checks to see if the year is a leap year and adjusts the count. Line 280 checks to see if the year can be divided by 4 evenly. Line 285 checks to see if the year can be divided evenly by 100 but not by 400.

If you don't understand all this, it doesn't matter. You can use the program without knowing which years are leap years.

Displaying the Months

The last half of the program is devoted to displaying each month on the screen. The VIC's 22 character screen is an asset. The display consists of (starting from the top) the year, the number of days passed and left, the month, the days of the week and, finally, the dates. February and the days left and passed are automatically compensated for in leap years. All calendars start with January, and each preceding month is displayed by pressing any key but f1. When f1 is pressed, the program asks for another YEAR WANTED? with a statement about the starting year. This also occurs when you enter a date before the starting year. At December, when you press a key, the program goes to January of the next year.

Perpetual Calendar

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

```
10 REM REMOVE REM FROM YEAR DESIRED AND ADJACENT L
   INE                                        :rem 145
15 REM AND ADD TO THE OLD LINE                :rem 113
20 REM YY=18ØØ                                :rem 254
21 REM CC=2                                    :rem 6Ø
3Ø YY=19ØØ                                     :rem 28
31 CC=1                                        :rem 88
4Ø REMYY=195Ø                                   :rem 6
41 REMCC=Ø                                     :rem 6Ø
5Ø REM YY=1582                                  :rem 8
6Ø REM CC=2                                    :rem 63
65 REM INSTRUCTIONS                             :rem 4
7Ø POKE36879,8                                 :rem 11
8Ø B1$="{2 SPACES}VIC'S{6 SPACES}":{2 SPACES}B2$="
   {2 SPACES}PERPETUAL{2 SPACES}":{2 SPACES}B3$="
   {2 SPACES}CALENDAR{3 SPACES}"              :rem 111
9Ø PRINT"{CLR}":PRINT:PRINT"{RVS}{GRN}{5 SPACES}
   E*3"                                       :rem 174
1ØØ PRINT"{RVS}{6 SPACES}E*3"                  :rem 83
11Ø PRINT"{RVS}{7 SPACES}E*3"                  :rem 84
12Ø FORI=1TO13:PRINTTAB(I-1)"E*3{RVS} "MID$(B3$,
    I,1);" ";MID$(B2$,I,1);" ";MID$(B1$,I,1);"
    E*3{OFF}                                  :rem 112
125 NEXTI                                      :rem 32
16Ø PRINT"{HOME}":PRINT:PRINT:PRINT"{RVS}{YEL}"SPC
    (15);YY;"{LEFT} {OFF}":PRINTSPC(17)"EM3EG3
    "                                         :rem 242
```

```
165 PRINTTAB(15);"{RVS} ???? {OFF}"                    :rem 214
170 PRINT:PRINTTAB(15)"{CYN}INST?"                     :rem 218
180 GETA$:IFA$=""THEN180                                :rem 85
190 IFA$="N"THEN330                                     :rem 34
200 PRINT"{GRN}{CLR} THIS PROGRAM MAKES{3 SPACES}U
    P A CALENDER FOR ANY"                              :rem 200
210 PRINT"YEAR {RED}{RVS}AFTER";YY;"{OFF}.":rem 97
220 PRINT:PRINT"{CYN} OTHER STARTING YEARS ARE IN
    {SPACE}LINES 10-60."                                :rem 73
230 PRINT:PRINT"{BLU} TO ENTER ANOTHER CAL-ENDAR Y
    EAR PRESS {RVS}F1{OFF}."                           :rem 236
240 PRINT:PRINT"{YEL} THE YEAR ALWAYS BE-
    {2 SPACES}GINS IN JANUARY.{2 SPACES}TO "
                                                        :rem 22
250 PRINT"SEE THE PROCEEDING{4 SPACES}MONTHS PRESS
     ANY OTHER KEY.";                                   :rem 40
260 PRINT" {PUR}(IN DECEMBER THE CALENDAR GOES TO
    {SPACE}THE NEXT YEAR)":PRINT                        :rem 40
270 INPUT"{WHT}YEAR WANTED";Y                          :rem 250
280 DA=365:IFY/4=INT(Y/4)THENDA=366                     :rem 68
281 IF DA=365 THEN 290                                  :rem 84
285 IF Y/100=INT(Y/100)AND Y/400<>INT(Y/400)THEN D
    A=365                                               :rem 41
290 H=0:DB=DA                                          :rem 210
300 IFY>YYTHEN360                                       :rem 51
310 RESTORE                                            :rem 184
320 K=0                                                 :rem 77
330 PRINT"{CLR}{WHT}I CAN'T START BEFORE ";YY+1
                                                        :rem 31
340 POKE36879,8                                         :rem 59
350 GOTO270                                            :rem 106
355 REM YEAR LOOP                                      :rem 236
360 Z=Y-YY:C=CC                                        :rem 168
370 FORI=1TOZ                                           :rem 53
380 C=C+1:YX=YY+I                                        :rem 8
390 IFYX/4=INT(YX/4)THENC=C+1                           :rem 26
400 IFC>=7THENC=C-7                                    :rem 110
410 NEXTI                                               :rem 29
420 IFY/4=INT(Y/4)THENC=C-1:IFC<0THENC=7+C :rem 50
425 REM READS MONTH                                    :rem 116
430 READM$,M:S=LEN(M$)                                   :rem 8
435 REMADJUST FOR LEAP YEAR                            :rem 133
440 IFM=3AND(Y/4=INT(Y/4))THENM=2                      :rem 228
450 IFC>7ORC=7THENC=C-7                                :rem 142
460 B=C                                                 :rem 92
490 REM PRINTS MONTH                                   :rem 231
500 PRINT"{BLK}":K=K+1:IFK=6THENK=0                    :rem 205
510 POKE36879,40+(K*16)                                :rem 190
520 PRINT"{CLR}":PRINT"{RVS}"TAB(8);Y;"{LEFT}
    {OFF}"                                             :rem 186
```

```
530 PRINTTAB(1);H;TAB(16);DB                          :rem 140
540 PRINTTAB(10-(S/2))"{A}";:FORI=1TOS:PRINT"*"
    ;:NEXTI:PRINT"{S}"                                 :rem 95
550 PRINTTAB(10-(S/2))"-{RVS}";M$;"{OFF}-":rem 175
560 PRINTTAB(10-(S/2))"{Z}";:FORI=1TOS:PRINT"*"
    ;:NEXTI:PRINT"{X}"                                :rem 109
570 PRINT:PRINT" S{2 SPACES}M{2 SPACES}T{2 SPACES}
    W{2 SPACES}T{2 SPACES}F{2 SPACES}S ":PRINT
                                                       :rem 51
580 PRINTSPC(3*B);                                     :rem 59
590 FORI=1TO31-M                                      :rem 189
600 PRINTI;:IFI>9THENPRINT"{LEFT}";                     :rem 8
610 IFPOS(0)>19THENPRINT:PRINT                        :rem 196
620 NEXTI:PRINT                                       :rem 231
700 GETX$:IFX$=""THEN700                              :rem 127
705 REM NEW DATE WANTED                                :rem 75
710 IFX$="{F1}"THEN310                                :rem 108
715 REM SETS UP NEXT                                  :rem 164
716 REM MONTH OR YEAR                                 :rem 218
720 C=B+3-M                                            :rem 51
730 IFM$<>"DECEMBER"THEN790                            :rem 94
740 RESTORE                                           :rem 191
750 Y=Y+1                                             :rem 231
760 H=O:DB=365:IFY/4=INT(Y/4)THENDB=366                :rem 87
770 DA=DB                                             :rem 230
780 GOTO430                                           :rem 111
790 DB=DB-(31-M):H=DA-DB                               :rem 60
800 GOTO430                                           :rem 104
810 END                                               :rem 112
1000 DATAJANUARY,0,FEBUARY,3,MARCH,0,APRIL,1,MAY,0
     ,JUNE,1,JULY,0,AUGUST,0,SEPTEMBER,1              :rem 101
1010 DATAOCTOBER,0,NOVEMBER,1,DECEMBER,0              :rem 236
```

Mailing List

Joseph J. Shaughnessy

With a few simple changes you can make yourself, this program will work on the VIC-20 with either disk or tape. Expansion memory is not required, but will greatly increase the capacity of the mailing list.

The following program is a modified and expanded version of a utility program from the Toronto PET Users Group, called "Addresses," and 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 accompanying program listing is for the VIC-20 and 1540/1541 disk drive, but it can easily be modified to operate with the cassette recorder by changing line 55 to:

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

Each address field is set up to receive eight items of information, as shown in lines 17 and 18. 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 28). 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.

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 mail list items.

This program will fit into any memory size VIC-20, but memory expansion is necessary to store very many addresses. (For instance, I have 65 names stored, and it takes about 12K of memory.) If you need space for more names (and have the memory), just add more dummy 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.

Mailing List

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

```
16 POKE36879,93:READR$,R:FORI=1TOR:READO$(I):NEXT:
   DATA"VIC MAILING LIST                      :rem 199
17 DATA8,"NAME(LAST NAME FIRST)","STREET ADDRESS",
   "CITY","STATE","ZIPCODE"                   :rem 222
18 DATA"HOME PHONE NO.","COMPANY NAME","WORK PHONE
    NO."                                      :rem 213
19 PRINT"{CLR}{BLK}{A}*******************{S}":PRIN
   T"-{2 SPACES}"R$" -":PRINT"{Z}*****************
   **{X}                                       :rem 14
20 PRINT"{DOWN} 1. ADD NAME":PRINT"{DOWN} 2. REMOV
   AL{4 SPACES}":PRINT"{DOWN} 3. SEARCH":PRINT"
   {DOWN} 4. EXAMINE                          :rem 181
21 PRINT"{DOWN}{SHIFT-SPACE}5. CHANGE":PRINT"
   {DOWN} 6. SAVE UPDATE":PRINT"{DOWN} 7. PRINT OP
   TION":PRINT"{DOWN} 8. END"                 :rem 160
22 RESTORE:PRINT"{2 DOWN}WHICH DO YOU WANT:rem 208
23 GETA$:IFA$=""THEN23                        :rem 237
24 IFA$<"1"ORA$>"8"THEN23                      :rem 95
25 READB$:IFB$<>"{-}"THEN25                    :rem 72
26 A=VAL(A$):ONAGOTO29,34,37,47,54,55,60,56:rem 79
28 POKE198,10:FORI=0TO9:POKE631+I,13:NEXT:END
                                               :rem 12
29 READA$:IFA$<>"{+}"THEN29                  :rem 24
30 READA:PRINT"{CLR}INPUT 0 FOR UNKNOWNS{DOWN}"
                                              :rem 153
31 PRINT"ITEM : "A"{DOWN}":FORI=1TOR:PRINTO$(I):IN
   PUTW$(I):PRINT:IFW$(1)=""THEN19             :rem 91
```

```
32 NEXT:W$(Ø)="XX"+CHR$(34)+","+STR$(A):Z=A*1Ø+5ØØ
   :K=Ø:PRINT"{CLR}{2 DOWN}"                     :rem 115
33 FORI=ZTOZ+R:PRINTI;"DATA"CHR$(34)W$(K):K=K+1:NE
   XT:PRINT"RUN{HOME}":GOTO28                    :rem 242
34 B$="":PRINT"{CLR}WHICH ITEM TO REMOVE ":INPUTB$
   :IFVAL(B$)=ØTHEN19                            :rem 152
35 PRINT"{CLR}{2 DOWN}":Z=VAL(B$)*1Ø+5ØØ:PRINTZ"DA
   TA"CHR$(34)"{+}"CHR$(34)","VAL(B$)    :rem 137
36 FORI=Z+1TOZ+R:PRINTI:NEXT:PRINT"RUN{HOME}":GOTO
   28                                            :rem 127
37 INPUT"{CLR}SEARCH FOR ";B$:IFB$=""THEN19:rem 88
38 H=Ø:READA$:IFA$="END"THEN19                   :rem 25Ø
39 IFA$="{+}"THENREADA:GOTO38                    :rem 241
4Ø READA:FORI=1TOR:READA$(I):IFLEFT$(A$(I),LEN(B$)
   )=B$THENH=1                                   :rem 149
41 NEXT:IFH=ØTHEN38                              :rem 188
42 PRINT"{CLR}ITEM :"A"{2 DOWN}":FORI=1TOR:PRINT"
   {2 SPACES}"A$(I):NEXT:IFW=1THENRETURN  :rem 199
43 PRINT"{2 DOWN}HIT ANY KEY TO PROCEED"  :rem 181
44 GETA$:IFA$=""THEN44                           :rem 243
45 IFQ=1THENRETURN                               :rem 198
46 GOTO38                                        :rem 14
47 A$="":INPUT"{CLR}WHICH ITEM";A$:IFA$="."THEN19
                                                 :rem 17Ø
48 A=VAL(A$):IFA=ØTHEN19                         :rem 147
49 READA$:IFA$="END"THEN19                       :rem 13
5Ø IFA<>VAL(A$)THEN49                            :rem 228
51 READA$(1):IFA$(1)="{+}"THEN19                 :rem 217
52 FORI=2TOR:READA$(I):NEXT:Q=1:GOSUB42:Q=Ø:IFW=1T
   HENRETURN                                     :rem 119
53 GOTO19                                        :rem 11
54 W=1:GOSUB47:W=Ø:PRINT"{HOME}{2 DOWN}":FORI=1TOR
   :INPUTW$(I):GOTO32                            :rem 21
55 PRINT"{CLR}":SAVE"@Ø:"+R$,8:END               :rem 59
56 END                                           :rem 66
6Ø PRINT"{CLR}{2 DOWN}{RVS}{3 SPACES}PRINTER OPTIO
   NS{3 SPACES}"                                 :rem 78
62 PRINT"{DOWN}{RVS}1{OFF} ENTIRE LIST"      :rem 34
64 PRINT"{DOWN}{RVS}2{OFF} MAILING LABELS":rem 214
66 PRINT"{DOWN}{RVS}3{OFF} INDIVIDUAL DATA":rem 4Ø
68 PRINT"{DOWN}{RVS}4{OFF} SINGLE MAILING LABEL";
                                                 :rem 134
7Ø PRINT"{DOWN}{RVS}5{OFF} RETURN TO PROGRAM"
                                                 :rem 189
72 GETZ$:IFZ$=""THEN72                           :rem 39
73 Z=VAL(Z$)                                     :rem 179
74 IFZ<1ORZ>5THEN6Ø                              :rem 196
76 OPEN1,4:RESTORE                               :rem 142
78 ONZGOTO82,98,1Ø6,1Ø6                          :rem 44
8Ø CLOSE1:GOTO19                                 :rem 236
```

```
82 READB$:IFB$<>"⁅-⁆"THEN82                              :rem 78
84 READB$:IFB$="⁅+⁆"THENCLOSE1:GOTO19                   :rem 247
86 IFB$="XX"THENREADA:PRINT#1,CHR$(10)CHR$(10)"ITE
   M";A:GOSUB92:GOTO84                                  :rem 192
88 PRINT#1,B$                                           :rem 227
90 GOTO84                                                :rem 14
92 READB$:FORI=1TO50                                    :rem 205
93 IFMID$(B$,I,1)=" "THENX=I:I=50                        :rem 53
94 NEXTI                                                :rem 245
95 N2$=LEFT$(B$,X):N1$=RIGHT$(B$,LEN(B$)-X):PRINT#
   1                                                    :rem 174
96 PRINT#1,N1$;" ";N2$:RETURN                           :rem 151
98 READB$:IFB$<>"⁅-⁆"THEN98                              :rem 92
99 READB$:IFB$="⁅+⁆"THENCLOSE1:GOTO19                   :rem 253
100 IFB$<>"XX"THEN99                                    :rem 149
101 READA:PRINT#1,CHR$(10):GOSUB92:GOSUB102:GOTO99
                                                         :rem 64
102 FORI=1TO4:READA$(I):NEXT                            :rem 214
103 PRINT#1,A$(1):PRINT#1,A$(2);", ";A$(3);"
    {3 SPACES}";A$(4)                                    :rem 42
104 RETURN                                              :rem 117
106 INPUT"{CLR}{DOWN}WHICH ITEM";Q:RESTORE:rem 155
107 READB$:IFB$<>"⁅-⁆"THEN107                           :rem 170
108 READB$:IFB$="⁅+⁆"THENPRINT"NO SUCH ITEM ON F
    ILE":FORX=0TO1500:NEXTX:CLOSE1:GOTO19  :rem 11
110 IFB$="XX"THENREADA:IFA=QTHEN114                     :rem 161
112 GOTO108                                             :rem 102
114 PRINT#1,CHR$(10)CHR$(10)"ITEM";A:GOSUB92
                                                         :rem 29
116 IFZ=4THEN120                                        :rem 180
118 FORX=1TOR-1:READB$:PRINT#1,B$:NEXT:CLOSE1:GOTO
    60                                                   :rem 54
120 GOSUB102:CLOSE1:GOTO60                               :rem 96
500 DATA"⁅-⁆"                                       :rem 207
510 DATA"XX", 1                                           :rem 1
511 DATA"SHAUGHNESSY J                                  :rem 111
512 DATA"12345 MAIN STREET                              :rem 207
513 DATA"ANYTOWN                                          :rem 5
514 DATA"ANYSTATE                                        :rem 63
515 DATA"00000                                          :rem 199
516 DATA"555-222-3333                                    :rem 51
517 DATA"ACME CORP.                                      :rem 81
518 DATA"555-222-4444                                    :rem 57
520 DATA"⁅+⁆", 2                                        :rem 249
530 DATA"⁅+⁆", 3                                        :rem 251
540 DATA"⁅+⁆", 4                                        :rem 253
550 DATA"⁅+⁆",5                                         :rem 255
560 DATA"⁅+⁆",6                                           :rem 1
570 DATA"⁅+⁆",7                                           :rem 3
```

```
580 DATA"[+]",8                                    :rem 5
590 DATA"[+]",9                                    :rem 7
600 DATA"[+]",10                                   :rem 39
610 DATA"END"                                      :rem 204
```

VICCAL: Super Calculator

Tommy Michael Tillman

Programming your VIC to be a calculator can be time-consuming and difficult. "VICCAL" is easy to use and does all the work for you. Requires at least an 8K memory expansion.

The VIC is a great home computer, but there are times when it would be nice to be able to use it as a calculator for computations. The VIC can do all the complex calculations that most people need, but the programming needed and the time involved are often not justified. Even in direct mode you have to contend with parentheses and which operator has priority over the other. In simple terms, it is easy to make mistakes.

Therefore, I set out to write a program that would make the VIC emulate a popular type of calculator. The system I imitated was the RPN (Reverse Polish Notation) System. This system looks difficult at first, but with a few minutes practice, it becomes natural and very easy to use.

RPN

What is the RPN System? How does it work? There are four easy steps: Type in the first number, press the RETURN key, type in the second number, and press the appropriate function key. Also, if the first number is already in the machine, you have to do only the last two steps. It's that easy.

The program needs an 8K expander ("VICCAL" will also work in higher memory configurations).

When you RUN the program, you'll be greeted by the cover page, which will stay on the screen for a few seconds. Next, a short instruction page will be presented. Basically, it states that you will have two different pages to use. The first of the two pages will present all the function keys that are available. The second page will show the *stack* and all the intermediate calculations. To move from one page to the other, you need only press f8 (SHIFTed f7).

The Stack

RPN-type calculators use a *stack.* The stack is made up of four memory levels. The first level is called the display or X level. The next higher level is called the *Y* level. The next is called the Z level. The highest level is the *T* level because it is the top of the stack.

Whenever a number is typed into the VIC, it is automatically stored in the X level. If you wish to push it higher into the stack, you must press the RETURN key. Whatever is in X is duplicated into Y. Likewise, whatever was in Y is duplicated into the Z level. And whatever was in Z is duplicated into the T level. Whatever was in T is now lost (erased).

If you are inputting a value into X and you previously pressed a function key (a one- or two-number calculation or function), the stack is automatically raised. (This eliminates a lot of unnecessary keystrokes).

After a calculation is performed (two numbers), the stack is automatically dropped. This means the value in T is duplicated into the Z level (the value in T remains the same as before). The previous value of Z is duplicated into the Y level. The previous value of Y and X are combined, depending upon the function you selected, and the result is placed into X.

After a one-number function is performed, the stack does not drop. The value in X changes only to its new value.

Using VICCAL

Let's try a few simple problems. First let's do some one-number calculations.

1. Find the sine (30 degrees):
   Type the number 30;
   press the Sin key (A). The result 0.5 returns in X level (if you are in degree mode).
2. Find the square root of 9:
   Type 9;
   press the Square Root key (J). The result 3 returns.
   Now let's try some two-number calculations.
3. 2 + 3:
   Type the number 2;
   press the RETURN key to enter the number into the machine;
   type the number 3;
   press the + key to add. The answer 5 returns in the X level.

4. 9 − 5:
   Type the number 9:
   press the RETURN key;
   type the number 5;
   press the − key. Answer 4 appears in X level.
5. 6 * 4:
   Type the number 6;
   press the RETURN key;
   type the number 4;
   press the * key (to multiply). The result 24 returns.
6. 24/8:
   Type the number 24;
   press the RETURN key;
   type the number 8;
   press the / key (to divide). The result 3 returns in the X level.

Now let's look at number input.

The allowable number characters are *1 2 3 4 5 6 7 8 9 0* ← . *E*

The symbol ← is used to input a minus sign before a number and before a power of 10 (exponential notation).The symbol . is the decimal point. The symbol *E* is for exponent (power of 10).

Positive integer—23456: Type 23456
Negative integer—−4568: Type ←4568
Positive decimal number—123.333: Type 123.333
Negative decimal number—−23.987: Type ←23.987
Positive exponential number with positive exponent—
1.234E12 (that is, 1.234*10↑12): Type 1.234E12
Positive exponential number with a negative exponent—
12.45E−12: Type 12.45E←12
Negative exponential number with positive exponent—
−44.98E14: Type ←44.98E14
Negative exponential number with negative exponent—
−23.45E−22: Type ←23.45E←22

Examples of More Complicated Problems

1. (2+3)*(4+5):
   Type 2; press RETURN;
   type 3 (3 replaces the old value in the X level);
   press + (the X and Y levels are added together and the result is in X);

type 4 (the result goes to the Y level and 4 goes in the X level);
press RETURN (the result goes to the Z level, 4 goes to the Y and X levels);
type 5 (5 goes in the X level);
press + (X and Y are added, first result drops back to Y level);
press * (the intermediate answers are multiplied).

2. (3*4)−(4*7):
   Type 3;
   press RETURN (input the value 3 into the X and Y levels);
   type 4 (replaces the old value in X with the new value of 4);
   press * (multiplies X and Y, result in X);
   type 4 (result goes to Y level and the value 4 goes in X);
   press RETURN (result goes to Z level, 4 goes to Y level, the 4 stays in X level);
   type 7 (7 goes to the X level);
   press * (X and Y are multiplied, first result drops to the Y level);
   press − (subtract the two intermediate results).
3. ((5+3)*(6−4))/(8−4):
   Type 5;
   press RETURN;
   type 3;
   press +;
   type 6;
   press RETURN
   type 4;
   press −;
   press *;
   type 8;
   press RETURN;
   type 4;
   press −;
   press /;

The Function Keys

f1 Clears X—stack remains the same except for X, which is now 0. Used to clear bad input.
f2 Clears everything except STO X—stack clears. Useful when

doing many complex calculations (such as linear regression).
f3 Exchanges the X value for the Y value—Z and T remain unchanged.
f4 Moves stack up one level—T goes to X, X goes to Y, Y goes to Z, Z goes to T.
f5 Moves stack down one level—X goes to T, T goes to Z, Z goes to Y, Y goes to X.
f6 Recalls last input value of X to the X level. Useful for correcting errors.
f7 Used to change trig mode from degrees to radians or radians to degrees. Press *before* any trig functions are performed or *after* you are through with one mode or the other.
f8 Used to shift from screen 1 to screen 2 or vice versa.

A Used to find the sine of an angle. The angle can be in degrees or radians. The stack does not drop.
B Used to find the cosine of an angle. The angle can be in degrees or radians. The stack does not drop.
C Used to find the tangent of an angle. The angle can be in degrees or radians. The stack does not drop.
D Used to find the arc sine, arc cosine, or arc tangent of a value (answer can be in degrees or radians). To pick correct function, when you press D it asks for your choice of the three functions or if you wish to exit. Pick the choice you wish. Stack does not drop.
E No function here. Remember, this key is used to represent the power of 10 when inputting exponential numbers.
F Used to find the value of e↑X (where e is the natural logarithm 2.7182818...).
G Used to find the natural logarithm of a number. Warning: X must be greater than 0. Example ln(2)=0.693147181
H Used to find the base 10 logarithm of a number. Warning: X must be greater than 0. Example log(6)=0.778151251
I Used to find 10↑X (the antilogarithm of X to base 10).
J Used to find the square root of X. Warning: X must not be negative. Example SQR(8)=2.82842713
K Used to find the square of X. Example: (13) 2=169
L Used to find the value of X raised to the power of Y. Y value must be in Y, and X value must be in X. Stack will drop after this calculation. Warning: Error if X=0 and Y<0, or X<0 and Y is noninteger. Example: 2 ↑ 3= 8
M M is used in conjunction with keys N through V.

This key will perform special internal calculations needed for the keys O–V.

To use this key, you will input one or two numbers as sets of *data pairs*. First, clear the stack using key f2. Next, input your first data pair (Y first, then X). Press M. The data pair disappears. X is replaced by 1. The next data pair is keyed in, and M is pressed again. Now 2 is in X. Therefore, X tells how many data pairs are input. This procedure is repeated until all data pairs are input.

N This key is used to correct an error in the last data pair that you input. If you can see that you input a bad number in the data pair, just input the same bad pair again and press N. This will reverse all the internal calculations and reduce the X value by 1. Now you can pick up where you left off in your data input procedure.

O Used to change polar coordinates to rectangular coordinates for vectors. To use this key, input the angle first (measured counterclockwise from positive X-axis) in either degree or radian mode, then the magnitude of the vector. Press O. The Y value of the rectangular coordinates will be in Y, and X value will be in X.

Vectors can be added or subtracted easily, using keys M, N, O, and P. First, clear the memory stack. Next, enter your first vector (in polar coordinates). Press O to change the vector to rectangular coordinates. Press M to sum the X-coordinates in X and the Y coordinates in Y. Next, enter the next vector. Press O to convert to rectangular coordinates. Press M to sum the coordinates.

Continue this procedure until all vectors have been converted and summed. Now press P to convert the summed X and Y coordinates to polar coordinates. The magnitude of the resultant vector is in X and the angle (in either trig or radian mode) is in Y.

To subtract vectors, input the vector in polar coordinates, press O to convert to rectangular coordinates, then press N to subtract. Continue, pressing either M to add or N to subtract.

P Used to change rectangular coordinates to polar coordinates for vectors. To use this key, input the Y value of the rectangular coordinate first, then input the X value. Press P. The angle of the polar coordinate will be in Y (in degree or radian mode) and magnitude of the vector will be in X.

Q Used to find the average value of X and Y. First input the data pairs using M, then press Q to find the average value of X in X and the average value of Y in Y.

Example:

data pair 1 (10,8)
data pair 2 (8,6)
data pair 3 (6,4)

Type in data pair 1 first. Type 8 first, RETURN, type 10, then M.
Type data pair 2 next. Type 6, RETURN, type 8, M.
Type the third data pair. Type 4, RETURN, type 6, M. All data pairs in. Now press Q to get the average of X and the average of Y. Averages are X=8 and Y=6.
R Used to find the standard deviation of X and Y of your data pairs. First input the data pairs using key M. Next press R. The standard deviation of X is in X and standard deviation of Y is in Y.
Example:
Type in the data pairs for Example Q. Press R. The standard deviation for X (1.999999999) is in X and standard deviation for Y (1.999999999) is in Y.
S Used to find the coordinates of the best-fit line for data pairs. First input data pairs using key M. Next press S. The slope for the line will be in X, the Y-intercept will be in Y, and the correlation coefficient will be in Z.
Example:

data pair 1(25,19)
data pair 2(18,16)
data pair 3(15,15)
data pair 4(8,6)
data pair 5(30,28)

Key in the data pairs. Type 19, RETURN, type 25, M. Type 16, RETURN, type 18, M. Type 15, RETURN, type 15, M. Type 6, RETURN, type 8, M. Type 28, RETURN, type 30, M. Now press S.
Slope of line in X is 0.892808684
Y-intercept of line in Y is −.341926689
Coefficient of correlation in Z is 0.967962159

The correlation coefficient is a measure of how good the line fits the data. The range of the coefficient is from 1 to −1. The closer the value to 1, the better the data fits a line with positive slope. The closer the value to −1, the better the data fits a line with a negative slope. The closer the value is to 0, the worse the fit.

T Used to find an estimated value for X, given a value of Y when a best-fit curve has been determined (look at S above). First, find the best-fit line for your data pairs using S above. *Do not clear.* Type a value of Y in (leave it in the X level). Press T. The estimated value of X will now be displayed in the X level.
Example: Input data pairs from example S above. Press S to get line coefficients. Given these Y values, determine X.
Y=8: First type 8, press T. Value of X=9.34346505
Y=12: Type 12, Press T. Value X=13.8237082
U Used to find an estimated value for Y, given a value of X when a best-fit curve has been determined (look at S above). First find the best-fit line for your data pairs, using S above. *Do not clear.* Type a value of X in (leave it in the X level). Press U. The estimated value of Y will be displayed in X.
Example: Input data pairs from S above. Press S to determine line coefficients.
X=17: Type 17, press U. The value of Y (14.8358209) will be displayed in X.
V Used to output the sum of the X values of the data pairs and the sum of the Y values. The data pairs are input using key M, and after all data pairs are input, key V is pressed. This automatically calculates the sum of X and the sum of Y and displays the sum of X in X and the sum of Y in Y.
Example: Input the data pairs from example S above.
Press V; the sum of X is in X. X=96. The sum of Y is in Y. Y=84
W Used to determine the reciprocal of numbers. Input the value in X and press W. The reciprocal will be displayed in X.
X This key inputs the value of PI.
Y This key allows the storage of a constant into memory (but not in the stack). To use, input the constant into X, and press the Y key. The constant is now in permanent memory.
Z This key is used to recall the constant from permanent memory. To use, press Z. The constant will be input into X (the stack will be raised).

As with any calculator, practice makes perfect. Use the VICCAL; with a few minutes practice, it will be easy to use.

VICCAL

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

```
100 POKE36879,27:TM=0                              :rem 174
110 PRINT"{CLR}{4 DOWN}{7 SPACES}{RVS}{PUR}
    {8 SPACES}{OFF}":PRINT"{7 SPACES}{RVS}{PUR} VI
    CCAL {OFF}":PRINT"{7 SPACES}{RVS}{8 SPACES}
    {OFF}"                                          :rem 38
120 PRINT"{3 DOWN} {RVS}{RED}{20 SPACES}{OFF}
    {2 SPACES}{RVS}VIC SUPER CALCULATOR{OFF}
    {2 SPACES}{RVS}{20 SPACES}{OFF}                :rem 216
130 FORN=1TO2000:NEXTN                              :rem 99
140 PRINT"{CLR}{6 SPACES}DIRECTIONS"               :rem 237
145 PRINT"{2 DOWN}THIS PROGRAM ALLOWS{3 SPACES}THE
     VIC TO EMULATE AN RPN STYLE OF"               :rem 199
150 PRINT"CALCULATOR. THE LIST{2 SPACES}OF DESIGNA
    TED KEYS ANDTHEIR FUNCTIONS";                  :rem 136
160 PRINT" ARE ONPAGE 1 AND THE STACK{2 SPACES}DIS
    PLAY IS ON PAGE 2."                             :rem 33
170 PRINT"{2 DOWN}{RED}PRESS F8 TO GO TO LIST"
                                                   :rem 202
180 GETA$:IFA$=""THEN180                            :rem 85
190 IFA$=CHR$(140)THEN2050                         :rem 168
200 GOTO180                                        :rem 100
210 IFA$="X"THEN230                                 :rem 36
220 GOTO250                                        :rem 100
230 IFL$=CHR$(13)ORL$=CHR$(133)THENA$="3.141592654
    ":LX=X:X=↑:GOSUB2580:GOTO2040                   :rem 12
240 A$="3.141592654":LX=X:T=Z:Z=Y:Y=X:X=↑:GOSUB258
    0:GOTO2040                                      :rem 44
250 IFA$="1"THEN410                                  :rem 1
260 IFA$="2"THEN410                                  :rem 3
270 IFA$="3"THEN410                                  :rem 5
280 IFA$="4"THEN410                                  :rem 7
290 IFA$="5"THEN410                                  :rem 9
300 IFA$="6"THEN410                                  :rem 2
310 IFA$="7"THEN410                                  :rem 4
320 IFA$="8"THEN410                                  :rem 6
330 IFA$="9"THEN410                                  :rem 8
340 IFA$="0"THEN410                                  :rem 0
350 IFA$="."THEN410                                :rem 255
360 IFA$="←"THEN410                                 :rem 49
370 IFA$="E"THEN410                                 :rem 24
380 IFA$=CHR$(20)ANDLEN(X$)>0THENX$=LEFT$(X$,(LEN(
    X$)-1)):A$="":GOTO400                           :rem 31
390 GOTO720                                        :rem 110
400 IFL$="Y"THENX$="":GOTO470                      :rem 167
410 IFL$=CHR$(13)THENX$="":GOTO470                 :rem 193
```

```
420 IFL$="+"ORL$="-"ORL$="*"ORL$="/"ORL$=CHR$(134)
    ORL$=CHR$(138)THEN427                          :rem 189
425 GOTO430                                        :rem 107
427 T=Z:Z=Y:Y=X:X$="":GOTO 470                     :rem 31
430 IFL$=CHR$(135)ORL$=CHR$(139)ORL$="A"ORL$="B"OR
    L$="C"ORL$="D"THEN437                          :rem 26
435 GOTO440                                        :rem 109
437 T=Z:Z=Y:Y=X:X$="":GOTO 470                     :rem 32
440 IFL$="T"ORL$="F"ORL$="G"ORL$="H"ORL$="I"ORL$="
    J"ORL$="K"THEN447                              :rem 89
445 GOTO450                                        :rem 111
447 T=Z:Z=Y:Y=X:X$="":GOTO 470                     :rem 33
450 IFL$="L"ORL$="M"ORL$="N"ORL$="O"ORL$="P"ORL$="
    Q"ORL$="R"THEN457                              :rem 125
455 GOTO460                                        :rem 113
457 T=Z:Z=Y:Y=X:X$="":GOTO 470                     :rem 34
460 IFL$="S"ORL$="T"ORL$="U"ORL$="V"ORL$="W"ORL$="
    X"ORL$="Z"THEN467                              :rem 177
465 GOTO470                                        :rem 115
467 T=Z:Z=Y:Y=X:X$=""                              :rem 21
470 IFLEN(X$)=0ANDA$="←"THENX$="-"+X$:A$="":GOTO4
    90                                             :rem 139
480 IFRIGHT$(X$,1)="E"ANDA$="←"THENX$=X$+"-":A$="
    "                                              :rem 245
490 X$=X$+A$:IFLEN(X$)<16THENPRINT"{OFF}{BLK}
    {HOME}{17 SPACES}{17 LEFT}X="X$:GOTO510:rem 54
500 PRINT"{HOME}MAX LENGTH IS 15 CHAR":PRINT"PRESS
     F1 TO CLEAR":GOTO510                          :rem 99
510 FORN=1TOLEN(X$)                                :rem 136
520 IFMID$(X$,N,1)="."THEN540                      :rem 63
530 NEXTN:GOTO620                                  :rem 48
540 XL$=LEFT$(X$,N-1):XR$=RIGHT$(X$,LEN(X$)-N):NF=
    1                                              :rem 213
550 IFLEFT$(XL$,1)="-"THENNF=-1:XL$=RIGHT$(XL$,LEN
    (XL$)-1)                                       :rem 118
560 XL=VAL(XL$):FORN=1TOLEN(XR$):IFMID$(XR$,N,1)="
    E"THEN580                                      :rem 69
570 NEXTN:XR=VAL(XR$)/(10↑(LEN(XR$))):X=NF*(XL+XR)
    :GOTO2040                                      :rem 106
580 XM$=LEFT$(XR$,N-1):XE$=RIGHT$(XR$,LEN(XR$)-N):
    XM=VAL(XM$)/(10↑(LEN(XM$)))                    :rem 41
585 X=NF*(XL+XM)                                   :rem 186
590 EF=1:IFLEFT$(XE$,1)="-"THENEF=-1:XE$=RIGHT$(XE
    $,LEN(XE$)-1)                                  :rem 136
600 XE=VAL(XE$):IFEF=1THENX=X*(10↑(XE)):GOTO2040
                                                   :rem 164
610 X=X/(10↑(XE)):GOTO2040                         :rem 234
620 FORN=1TOLEN(X$):IFMID$(X$,N,1)="E"THEN660
                                                   :rem 134
```

```
630 NEXTN:NF=1:IFLEFT$(X$,1)="-"THENNF=-1:X$=RIGHT
    $(X$,LEN(X$)-1)                          :rem 72
640 X=NF*VAL(X$):IFNF=-1THENX$="-"+X$:GOTO2040
                                             :rem 206
650 GOTO2040                                 :rem 154
660 XL$=LEFT$(X$,N-1):XE$=RIGHT$(X$,(LEN(X$)-N)):N
    F=1                                      :rem 28
670 IFLEFT$(XL$,1)="-"THENNF=-1:XL$=RIGHT$(XL$,N-2
    )                                        :rem 208
680 XL=VAL(XL$):EF=1                         :rem 174
690 IFLEFT$(XE$,1)="-"THENEF=-1:XE$=RIGHT$(XE$,(LE
    N(XE$)-1))                               :rem 167
700 IFEF=1THENX=NF*XL*(10↑(VAL(XE$))):GOTO2040
                                             :rem 254
710 X=NF*XL/(10↑(VAL(XE$))):GOTO2040          :rem 77
720 IFA$=CHR$(13)THEN740                     :rem 74
730 GOTO750                                  :rem 111
740 LX=X:T=Z:Z=Y:Y=X:X$="":GOSUB2580:GOTO2040
                                             :rem 68
750 IFA$="+"THENLX=X:X=X+Y:Y=Z:Z=T:X$="":GOSUB2580
    :GOTO2040                                :rem 151
760 IFA$="-"THENLX=X:X=Y-X:Y=Z:Z=T:X$="":GOSUB2580
    :GOTO2040                                :rem 156
770 IFA$="*"THENLX=X:X=X*Y:Y=Z:Z=T:X$="":GOSUB2580
    :GOTO2040                                :rem 151
780 IFA$="/"THENLX=X:X=Y/X:Y=Z:Z=T:X$="":GOSUB2580
    :GOTO2040                                :rem 162
790 IFA$=CHR$(133)THENX=0:X$="":GOSUB2580:GOTO2040
                                             :rem 167
800 IFA$=CHR$(137)THEN807                    :rem 132
805 GOTO 810                                 :rem 111
807 LX=0:X=0:Y=0:Z=0:T=0:M1=0:M2=0:M3=0:M4=0:M5=0:
    M6=0:X$="":GOTO820M6=0:X$="":GOTO820      :rem 16
810 GOTO830                                  :rem 109
820 GOSUB2580:GOTO2040                       :rem 34
830 IFA$=CHR$(134)THENTT=Y:Y=X:X=TT:TT=0:X$="":GOS
    UB2580:GOTO2040                          :rem 10
840 IFA$=CHR$(138)THENTT=T:T=Z:Z=Y:Y=X:X=TT:X$="":
    GOSUB2580:GOTO2040                       :rem 10
850 IFA$=CHR$(135)THENTT=X:X=Y:Y=Z:Z=T:T=TT:X$="":
    GOSUB2580:GOTO2040                        :rem 8
860 IFA$=CHR$(139)THENT=Z:Z=Y:Y=X:X=LX:X$="":GOSUB
    2580:GOTO2040                            :rem 150
870 IFA$=CHR$(136)THEN890                    :rem 140
880 GOTO910                                  :rem 115
890 IFTM=1THENTM=0:GOTO2040                  :rem 181
900 TM=1:GOTO2040                            :rem 225
910 IFA$="A"THEN930                          :rem 27
920 GOTO950                                  :rem 114
```

```
930 IFTM=ØTHENLX=X:X=X*(2*↑)/36Ø:X=SIN(X):X$="":GO
    SUB258Ø:GOTO2Ø4Ø                            :rem 193
94Ø LX=X:X=SIN(X):X$="":GOSUB258Ø:GOTO2Ø4Ø :rem 49
95Ø IFA$="B"THEN97Ø                             :rem 36
96Ø GOTO99Ø                                     :rem 122
97Ø IFTM=ØTHENLX=X:X=X*(2*↑)/36Ø:X=COS(X):X$="":GO
    SUB258Ø:GOTO2Ø4Ø                            :rem 192
98Ø LX=X:X=COS(X):X$="":GOSUB258Ø:GOTO2Ø4Ø :rem 48
99Ø IFA$="C"THEN1Ø1Ø                            :rem 75
1ØØØ GOTO1Ø3Ø                                   :rem 19Ø
1Ø1Ø IFTM=ØTHENLX=X:X=X*(2*↑)/36Ø:X=TAN(X):X$="":G
     OSUB258Ø:GOTO2Ø4Ø                          :rem 224
1Ø2Ø LX=X:X=TAN(X):X$="":GOSUB258Ø:GOTO2Ø4Ø:rem 8Ø
1Ø3Ø IFA$="D"THENPRINT"{HOME}{OFF}{PUR}PICK FUNCTI
     ON A-SIN{4 SPACES}B-COS C-TAN D-ESC":GOTO1Ø5Ø
                                                :rem 2Ø7
1Ø4Ø GOTO117Ø                                   :rem 199
1Ø5Ø GETB$:IFB$=""THEN1Ø5Ø                      :rem 177
1Ø6Ø IFB$="A"THEN111Ø                           :rem 112
1Ø7Ø IFB$="B"THEN113Ø                           :rem 116
1Ø8Ø IFB$="C"THEN115Ø                           :rem 12Ø
1Ø9Ø IFB$="D"THENGOSUB258Ø:GOTO2Ø4Ø             :rem 59
11ØØ GOTO1Ø4Ø                                   :rem 192
111Ø LX=X:X=ATN(X/SQR(-X*X+1)):IFTM=1THENX$="":GOS
     UB258Ø:GOTO2Ø4Ø                            :rem 246
112Ø X=X*36Ø/(2*↑):X$="":GOSUB258Ø:GOTO2Ø4Ø:rem 72
113Ø LX=X:X=-ATN(X/SQR(-X*X+1))+↑/2:IFTM=1THENX$="
     ":GOSUB258Ø:GOTO2Ø4Ø                       :rem 176
114Ø X=X*36Ø/(2*↑):X$="":GOSUB258Ø:GOTO2Ø4Ø:rem 74
115Ø LX=X:X=ATN(X):IFTM=1THENX$="":GOSUB258Ø:GOTO2
     Ø4Ø                                        :rem 33
116Ø X=X*36Ø/(2*↑):X$="":GOSUB258Ø:GOTO2Ø4Ø:rem 76
117Ø IFA$="F"THEN119Ø                           :rem 126
118Ø GOTO12ØØ                                   :rem 198
119Ø LX=X:X=EXP(X):X$="":GOSUB258Ø:GOTO2Ø4Ø:rem 98
12ØØ IFA$="G"THEN122Ø                           :rem 115
121Ø GOTO123Ø                                   :rem 195
122Ø LX=X:X=LOG(X):X$="":GOSUB258Ø:GOTO2Ø4Ø:rem 81
123Ø IFA$="H"THEN125Ø                           :rem 122
124Ø GOTO126Ø                                   :rem 2Ø1
125Ø LX=X:X=Ø.4342944819*LOG(X):X$="":GOSUB258Ø:GO
     TO2Ø4Ø                                     :rem 236
126Ø IFA$="I"THENGOTO128Ø                       :rem 186
127Ø GOTO129Ø                                   :rem 2Ø7
128Ø LX=X:X=1Ø↑X:X$="":GOSUB258Ø:GOTO2Ø4Ø :rem 227
129Ø IFA$="J"THEN131Ø                           :rem 127
13ØØ GOTO132Ø                                   :rem 195
131Ø LX=X:X=SQR(X):X$="":GOSUB258Ø:GOTO2Ø4Ø
                                                :rem 1Ø1
```

```
1320 IFA$="K"THEN1340                              :rem 125
1330 GOTO1350                                      :rem 201
1340 LX=X:X=X↑2:X$="":GOSUB2580:GOTO2040           :rem 177
1350 IFA$="L"THEN1370                              :rem 132
1360 GOTO1380                                      :rem 207
1370 LX=X:X=X↑Y:Y=Z:Z=T:X$="":GOSUB2580:GOTO2040
                                                    :rem 42
1380 IFA$="M"THEN1400                              :rem 130
1390 GOTO1420                                      :rem 205
1400 M1=M1+1:M2=M2+X:M3=M3+X↑2:M4=M4+Y:M5=M5+Y↑2:M
     6=M6+(X*Y):X=M1:Y=0:Z=0:T=0:X$=""             :rem 188
1410 GOSUB2580:GOTO2040                             :rem 78
1420 IFA$="N"THEN1440                              :rem 130
1430 GOTO1460                                      :rem 204
1440 M1=M1-1:M2=M2-X:M3=M3-X↑2:M4=M4-Y:M5=M5-Y↑2:M
     6=M6-(X*Y):X=M1:Y=0:Z=0:T=0                   :rem 149
1445 GOSUB2580                                      :rem 29
1450 X$="":GOTO2040                                  :rem 0
1460 IFA$="Q"THEN1480                              :rem 141
1470 GOTO1490                                      :rem 211
1480 X=M2/M1:Y=M4/M1:Z=0:T=0:X$="":GOSUB2580:GOTO2
     040                                           :rem 129
1490 IFA$="R"THEN1510                              :rem 139
1500 GOTO1530                                      :rem 200
1510 X=SQR(((M1*M3)-(M2↑2))/(M1*(M1-1))):Y=SQR(((M
     1*M5)-(M4↑2))/(M1*(M1-1))):Z=0:T=0             :rem 18
1520 X$="":GOSUB2580:GOTO2040                      :rem 135
1530 IFA$="S"THEN1550                              :rem 139
1540 GOTO1580                                      :rem 209
1550 X=(((M1*M6)-(M2*M4))/((M1*M3)-M2↑2)):Y=(((M4*
     M3)-(M2*M6))/((M1*M3)-M2↑2)):T=0               :rem 35
1560 Z=((M1*M6)-(M2*M4))/SQR(((M1*M3)-M2↑2)*((M1*M
     5)-M4↑2))                                     :rem 136
1570 X$="":GOSUB2580:GOTO2040                      :rem 140
1580 IFA$="T"THEN1600                              :rem 141
1590 GOTO1620                                      :rem 209
1600 X=((((M1*M6)-(M2*M4))*M2)+((M1*M3)-M2↑2)*((M1
     *X)-M4))/(M1*((M1*M6)-(M2*M4)))               :rem 100
1610 Y=0:Z=0:X$="":GOSUB2580:GOTO2040              :rem 136
1620 IFA$="U"THEN1640                              :rem 141
1630 GOTO1660                                      :rem 208
1640 X=((((M1*M3)-M2↑2)*M4)+(((M1*M6)-(M2*M4))*((M
     1*X)-M2)))/(M1*((M1*M3)-M2↑2))                 :rem 74
1650 Y=0:Z=0:T=0:X$="":GOSUB2580:GOTO2040          :rem 135
1660 IFA$="O"THEN1680                              :rem 143
1670 GOTO1700                                      :rem 207
1680 IFTM=0THENY=Y*2*π/360                         :rem 215
1690 R=X:ANGLE=Y:X=R*COS(AN):Y=R*SIN(AN):Z=0:T=0:R
     =0:AN=0:X$="":GOSUB2580:GOTO2040               :rem 56
```

```
1700 IFA$="P"THEN172Ø                                :rem 134
171Ø GOTO191Ø                                        :rem 2Ø5
172Ø IFL$="M"ORL$="N"ORL$="V"THEN182Ø                 :rem 89
173Ø X1=X:Y1=Y:X=SQR(X1↑2+Y1↑2):IFY1>ØANDX1=ØTHENY
     =↑/2:GOTO18ØØ                                   :rem 162
174Ø IFY1<ØANDX1=ØTHENY=3*↑/2:GOTO18ØØ                :rem 216
175Ø IFX1<ØANDY1=ØTHENY=↑:GOTO18ØØ                    :rem 27
176Ø Y=ATN(Y1/X1)                                    :rem 218
177Ø IFX1<ØANDY1<ØTHENY=Y+↑:GOTO18ØØ                  :rem 16Ø
178Ø IFX1>ØANDY1<ØTHENY=2*↑+Y:GOTO18ØØ                :rem 255
179Ø IFX1<ØANDY1>ØTHENY=↑+Y                           :rem 1Ø4
18ØØ IFTM=ØTHENY=Y*36Ø/(2*↑)                          :rem 34
181Ø X$="":GOSUB258Ø:GOTO2Ø4Ø                         :rem 137
182Ø X=SQR(M2↑2+M4↑2):IFM4>ØANDM2=ØTHENY=↑/2:GOTO1
     89Ø                                             :rem 211
183Ø IFM4<ØANDM2=ØTHENY=3*↑/2:GOTO189Ø                :rem 2Ø6
184Ø IFM2<ØANDM4=ØTHENY=↑:GOTO189Ø                    :rem 17
185Ø Y=ATN(M4/M2)                                    :rem 199
186Ø IFM2<ØANDM4<ØTHENY=Y+↑:GOTO189Ø                  :rem 15Ø
187Ø IFM2>ØANDM4<ØTHENY=2*↑+Y:GOTO189Ø                :rem 245
188Ø IFM2<ØANDM4>ØTHENY=↑+Y                           :rem 85
189Ø IFTM=ØTHENY=Y*36Ø/(2*↑)                          :rem 43
19ØØ X$="":GOSUB258Ø:GOTO2Ø4Ø                         :rem 137
191Ø IFA$="W"THEN193Ø                                 :rem 147
192Ø GOTO195Ø                                        :rem 212
193Ø IFX<>ØTHENX=1/X:X$="":GOSUB258Ø:GOTO2Ø4Ø
                                                     :rem 211
194Ø PRINT"{HOME}{BLU}{OFF}X="X"{2 SPACES}{RVS}CAN
     NOT DIVIDE{2 SPACES}RESULT IS UNDEFINED{OFF}"
     :GOTO2Ø4Ø                                       :rem 246
195Ø IFA$="V"THEN197Ø                                 :rem 154
196Ø GOTO198Ø                                        :rem 219
197Ø X=M2:Y=M4:Z=Ø:T=Ø:X$="":GOSUB258Ø:GOTO2Ø4Ø
                                                      :rem 43
198Ø IFA$="Y"THEN2ØØØ                                 :rem 145
199Ø GOTO2Ø1Ø                                        :rem 2Ø7
2ØØØ X2=X:X$="":X$="":GOSUB258Ø:GOTO2Ø4Ø              :rem 17
2Ø1Ø IFA$="Z"THEN2Ø3Ø                                 :rem 134
2Ø2Ø GOTO2Ø4Ø                                        :rem 195
2Ø3Ø T=Z:Z=Y:Y=X:X=X2:X$="":GOSUB258Ø                 :rem 27
2Ø4Ø L$=A$:RETURN                                    :rem 242
2Ø5Ø PRINT"{CLR}{OFF}{BLK}X="X:PRINT"{DOWN}{RVS}
     {CYN}{16 SPACES}";                               :rem 66
2Ø6Ø IFTM=ØTHENPRINT"{BLK}TM={RVS}DEG{CYN}
     {22 SPACES}";:GOTO2Ø8Ø                          :rem 2Ø4
2Ø7Ø PRINT"{RVS}TM={GRN}RAD{CYN}{22 SPACES}";
                                                      :rem 89
2Ø8Ø PRINT"{RVS}{RED}F1=CLR X{5 SPACES}{OFF}J=SQR(
     X)                                               :rem 76
```

```
2090 PRINT"{RVS}{RED}F2=CLR ALL{3 SPACES}{OFF}K=X↑
     2                                        :rem 25
2100 PRINT"{RED}{RVS}F3=X FOR Y{3 SPACES}{OFF}L=X↑
     Y                                        :rem 24
2110 PRINT"{RED}{RVS}F4=STACK UP{2 SPACES}{BLK}M=S
     UM +{2 SPACES}{OFF}";                    :rem 156
2120 PRINT"{RVS}{RED}F5=STACK DOWN{BLK}N=SUM -
     {2 SPACES}{OFF}";                        :rem 52
2130 PRINT"{RVS}{RED}F6=LAST X{4 SPACES}{BLK}O=P T
     O R {OFF}";                              :rem 56
2140 PRINT"{RVS}{RED}F7=DEG - RAD {BLK}P=R TO P
     {OFF}";                                  :rem 131
2150 PRINT"{OFF}{BLU} A=SIN(X){4 SPACES}{RVS}{BLK}
     Q=AVR X{2 SPACES}{OFF}";                 :rem 153
2160 PRINT"{OFF}{BLU} B=COS(X){4 SPACES}{RVS}{BLK}
     R= S{5 SPACES}{OFF}";                    :rem 169
2170 PRINT"{OFF}{BLU} C=TAN(X){4 SPACES}{RVS}{BLK}
     S=LIN R{2 SPACES}{OFF}";                 :rem 140
2180 PRINT"{OFF}{BLU} D=ARC{7 SPACES}{RVS}{BLK}T=X
      EST{2 SPACES}{OFF}";                    :rem 232
2190 PRINT"{BLK} {BLK}F=E↑X{7 SPACES}{RVS}{BLK}U=Y
      EST{2 SPACES}{OFF}";                    :rem 129
2200 PRINT"{BLK} G=LN(X){5 SPACES}{RVS}{BLK}V=RCL
     {SPACE}SUM{OFF}";                        :rem 196
2210 PRINT"{BLK} H=LOG(X){4 SPACES}{RVS}{PUR}W= 1/
     X{3 SPACES}{OFF}";                       :rem 253
2220 PRINT"{BLK} I=10↑X{6 SPACES}{RVS}{PUR}X= PI
     {4 SPACES}{OFF}";                        :rem 109
2230 PRINT"{BLK}{13 SPACES}{RVS}{BLU}Y=STO X
     {2 SPACES}{OFF}";                        :rem 10
2240 PRINT"{BLK}{13 SPACES}{RVS}{BLU}Z=RCL X
     {2 SPACES}{OFF}";                        :rem 247
2250 PRINT"{RVS}{CYN}{22 SPACES}";            :rem 134
2260 PRINT"{RVS}{BLK}PRESS F8 TO SEE STAC {LEFT}
     {INST}K";                                :rem 170
2270 GETA$:IFA$=""THEN2270                    :rem 185
2280 IFA$=CHR$(140)THEN2320                   :rem 218
2290 GOSUB210:IFTM=0THENPRINT"{HOME}{2 DOWN}
     {19 RIGHT}{RVS}{BLK}DEG{OFF}";:GOTO2270
                                              :rem 144
2300 PRINT"{HOME}{2 DOWN}{19 RIGHT}{RVS}{GRN}RAD
     {OFF}";                                  :rem 198
2310 GOTO2270                                 :rem 202
2320 PRINT"{CLR}{BLU}{OFF}X="X:PRINT"{DOWN}{OFF}
     {7 SPACES}{RVS}{RED}STACK{OFF}";:IFTM=0THENPR
     INT"{4 SPACES}{BLU}TM={RVS}{BLK}DEG{BLU}{OFF}
     ";                                       :rem 26
2325 GOTO2340                                 :rem 206
2330 PRINT"{4 SPACES}{BLU}TM={RVS}{GRN}RAD{BLU}
     {OFF}";                                  :rem 137
```

```
2340 PRINT" {BLU}{A}***************{S}              :rem 53
2350 PRINT"T-{15 SPACES}-                          :rem 135
2360 PRINT" {Q}**************{W}                    :rem 24
2370 PRINT"Z-{15 SPACES}-"                         :rem 177
2380 PRINT" {Q}**************{W}"                   :rem 60
2390 PRINT"Y-{15 SPACES}-                          :rem 144
2400 PRINT" {Q}**************{W}                    :rem 19
2410 PRINT"X-{15 SPACES}-                          :rem 136
2420 PRINT" {Z}**************{X}                    :rem 33
2430 PRINT"{DOWN}{PUR}{7 SPACES}{RVS}LAST X{OFF}
                                                    :rem 85
2440 PRINT" {A}**************{S}                    :rem 23
2450 PRINT" -{15 SPACES}-"                          :rem 86
2460 PRINT" {Z}**************{X}                    :rem 37
2470 PRINT"{5 DOWN}{BLK}PRESS F8 TO GO TO LIT
     {LEFT}{INST}S";                                :rem 18
2480 PRINT"{HOME}{OFF}{4 DOWN}{2 RIGHT}{15 SPACES}
     ":PRINT"{HOME}{BLU}{OFF}{4 DOWN}{2 RIGHT}"T
                                                    :rem 99
2490 PRINT"{HOME}{6 DOWN}{2 RIGHT}{15 SPACES}":PRI
     NT"{HOME}{BLU}{6 DOWN}{2 RIGHT}"Z        :rem 138
2500 PRINT"{HOME}{8 DOWN}{2 RIGHT}{15 SPACES}":PRI
     NT"{HOME}{BLU}{8 DOWN}{2 RIGHT}"Y        :rem 197
2510 PRINT"{HOME}{10 DOWN}{2 RIGHT}{15 SPACES}":PR
     INT"{HOME}{BLU}{10 DOWN}{2 RIGHT}"X        :rem 9
2520 PRINT"{HOME}{15 DOWN}{2 RIGHT}{15 SPACES}":PR
     INT"{HOME}{RVS}{BLK}{15 DOWN}{2 RIGHT}"LX"
     {RVS}"                                        :rem 217
2530 GETA$:IFA$=""THEN2530                         :rem 183
2540 IFA$=CHR$(140)THEN2050                        :rem 217
2550 GOSUB210:IFTM=0THENPRINT"{HOME}{2 DOWN}
     {19 RIGHT}{RVS}{BLK}DEG{OFF}";:GOTO2480
                                                   :rem 146
2560 PRINT"{HOME}{2 DOWN}{19 RIGHT}{RVS}{GRN}RAD
     {OFF}";                                       :rem 206
2570 GOTO2480                                      :rem 213
2580 PRINT"{HOME}{OFF}{BLU}{41 SPACES}":PRINT"
     {HOME}X="X:RETURN                             :rem 137
```

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 VIC-20 with 8K or greater memory expansion. SpeedScript will considerably amplify the utility of your computer.

A current advertising campaign extols the virtues of a ballpoint 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 word processing 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 VIC 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 VIC, we think 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. The program looks long, but it is 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 the MLX article (Appendix K) before you begin typing in SpeedScript.

Type in and SAVE the MLX program. (Both MLX and SpeedScript require an 8K expander.) 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 44,37:POKE 9472,0: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: 4609. The ending address is the last number plus five: 9348. After you enter this, follow the instructions in the MLX article to enter the listing. We've entered it here, and 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.

If you type SpeedScript in several sessions, be sure to enter the line above each time before you load MLX.

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 white with black 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 are 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. 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 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 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 (a RETURN mark by itself, which you can use to make blank lines, counts as a paragraph).

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. In a very few cases, 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 5K of text memory on a VIC with an 8K expander.

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 juunmped over
```

DEL is struck twice, deleting the erroneous characters:

```
The quick brown fox juumped over
The quick brown fox jumped 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.

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. As an analogy, CTRL-Delete is like the DEL key, and CTRL-Erase is like the CTRL-back-arrow.

What if you pressed one key too many in the Erase command? What if you change your mind? What if you accidentally erased 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 *fail-safe buffer,* a 1K reserved area of memory.

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. Additionally, 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 (1K), 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, saving your work as you go along, then coming back to it at another time. Saving and loading alone elevate 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. 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).

If you wish to check the available memory, press CTRL-= and SpeedScript will tell you how much memory you have left.

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). It may be obvious by now that CTRL-↑ is much like the DOS wedge.

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 VIC, 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.

Well, 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 hardcopy (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 color 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, etc. 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:

```
l10r60s1t10b50←
```

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.

[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 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 [#] anyplace you want a page number to appear. Examples:

A center page title 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 of the current page.

[s] Line spacing. This command allows you to change from double to single line spacing.

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 are 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, italic is turned on with ESC4. 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, 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 anyplace you want to print a back-arrow, just embed a [1] in your text. 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 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
13444432 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 (NEC8023), 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.

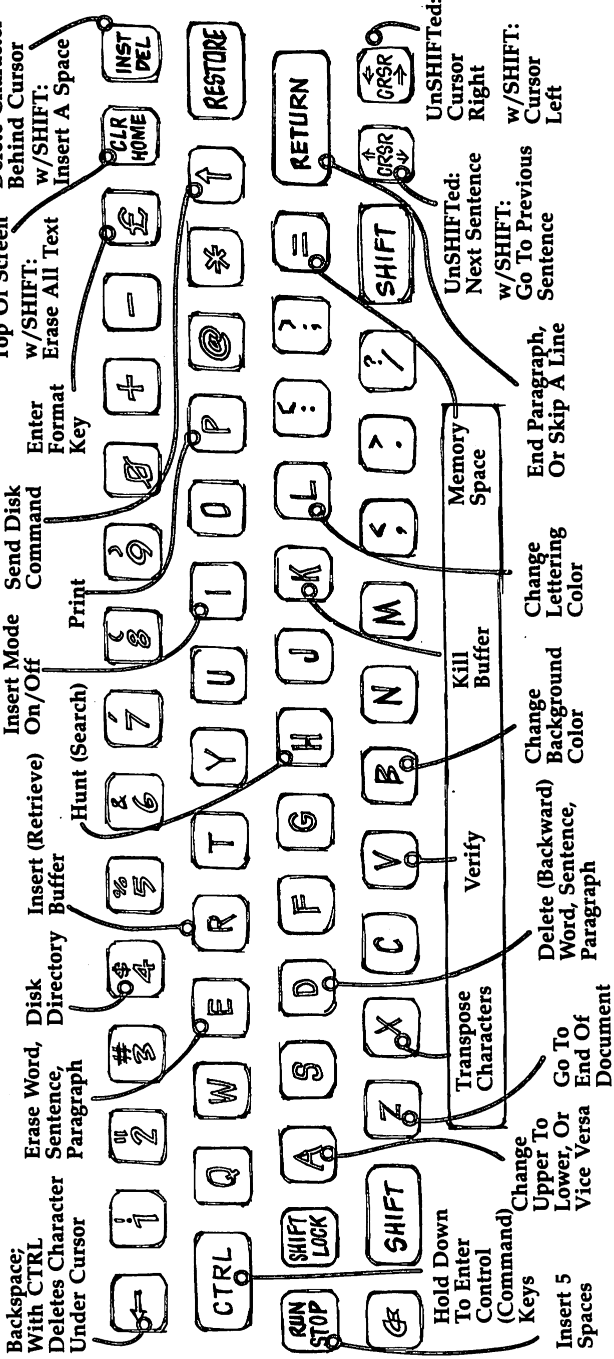
You also need to use the [a] if you want to bypass the emulation offered by the interface. You may do this to be able to activate your printer's special function codes which are often intercepted and interpreted by the interface. You will also have to use a different secondary address. I'll have to bow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

Figure 1. Keyboard Map

Lettered Commands Entered With CTRL



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.

As you can tell, SpeedScript is a truly comprehensive word processor. Although SpeedScript is ultimately easy to use, it may take you awhile to master all the features and variations. To make it easy to learn to use SpeedScript, we have included summaries of the commands and functions on pages 126, 145, 147, and 149. I hope your adventure will prove to be fascinating and fruitful.

SpeedScript

```
4609 :Ø11,Ø18,Ø10,ØØØ,158,Ø52,25Ø
4615 :Ø54,Ø5Ø,Ø43,ØØØ,ØØØ,ØØØ,16Ø
4621 :Ø32,114,Ø19,Ø76,247,Ø19,ØØ8
4627 :ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,Ø19
4633 :ØØØ,ØØØ,165,251,141,Ø59,129
4639 :Ø18,165,252,141,Ø6Ø,Ø18,173
4645 :165,253,141,Ø62,Ø18,165,Ø73
4651 :254,141,Ø63,Ø18,166,181,Ø98
4657 :24Ø,Ø32,169,ØØØ,141,132,251
4663 :Ø36,16Ø,ØØØ,185,ØØØ,ØØØ,18Ø
4669 :153,ØØØ,ØØØ,2ØØ,2Ø4,132,238
4675 :Ø36,2Ø8,244,238,Ø6Ø,Ø18,1Ø3
4681 :238,Ø63,Ø18,224,ØØØ,24Ø,Ø88
4687 :ØØ7,2Ø2,2Ø8,224,165,18Ø,Ø41
4693 :2Ø8,222,Ø96,165,181,17Ø,1Ø3
4699 :ØØ5,18Ø,2Ø8,ØØ1,Ø96,Ø24,Ø93
47Ø5 :138,1Ø1,252,141,131,Ø18,11Ø
4711 :165,251,141,13Ø,Ø18,Ø24,Ø64
4717 :138,1Ø1,254,141,134,Ø18,127
4723 :165,253,141,133,Ø18,232,Ø33
4729 :164,18Ø,2Ø8,ØØ4,24Ø,Ø13,162
4735 :16Ø,255,185,ØØØ,ØØØ,153,112
4741 :ØØØ,ØØØ,136,192,255,2Ø8,156
4747 :245,2Ø6,131,Ø18,2Ø6,134,Ø55
4753 :Ø18,2Ø2,2Ø8,234,Ø96,169,Ø48
4759 :Ø22,133,195,133,Ø2Ø,169,Ø55
4765 :Ø16,133,196,169,148,133,184
4771 :Ø21,173,128,Ø36,133,155,Ø41
4777 :173,129,Ø36,133,156,173,2Ø1
```

```
4783 :131,036,032,223,019,162,010
4789 :001,160,000,173,140,036,179
4795 :145,020,177,155,153,142,211
4801 :036,200,041,127,201,031,061
4807 :240,019,192,022,208,235,091
4813 :136,177,155,041,127,201,018
4819 :032,240,005,136,208,245,053
4825 :160,021,200,132,167,136,009
4831 :185,142,036,145,195,136,038
4837 :016,248,164,167,024,152,232
4843 :101,155,133,155,165,156,076
4849 :105,000,133,156,152,157,176
4855 :060,003,192,022,240,008,004
4861 :169,032,145,195,200,076,046
4867 :249,018,024,165,195,105,247
4873 :022,133,195,133,020,144,144
4879 :004,230,196,230,021,232,160
4885 :224,023,240,003,076,182,001
4891 :018,165,155,141,138,036,168
4897 :165,156,141,139,036,096,254
4903 :173,019,018,133,155,141,166
4909 :128,036,141,134,036,133,141
4915 :038,173,020,018,133,156,077
4921 :141,129,036,141,135,036,163
4927 :133,039,056,173,022,018,248
4933 :237,020,018,170,169,032,203
4939 :160,255,198,156,145,155,120
4945 :200,230,156,145,155,200,143
4951 :208,251,230,156,202,208,062
4957 :246,145,155,096,133,167,011
4963 :132,168,160,000,177,167,135
4969 :240,006,032,210,255,200,024
4975 :208,246,096,169,001,141,204
4981 :141,036,032,174,022,169,179
4987 :000,141,019,018,141,021,207
4993 :018,141,023,018,141,025,239
4999 :018,024,173,130,002,105,075
5005 :020,141,020,018,056,173,057
5011 :132,002,233,001,141,026,170
5017 :018,056,233,004,141,024,117
5023 :018,056,233,001,141,022,118
5029 :018,169,000,141,140,036,157
5035 :032,039,019,169,000,141,059
5041 :131,036,169,255,141,138,023
5047 :002,032,121,023,032,203,084
5053 :019,169,076,160,035,032,168
5059 :097,019,169,000,141,130,239
5065 :036,096,162,021,169,160,077
5071 :157,000,016,202,016,250,080
```

```
5077 :169,019,032,210,255,169,043
5083 :018,076,210,255,141,134,029
5089 :002,162,021,157,000,148,203
5095 :202,016,250,096,072,041,140
5101 :128,074,133,167,104,041,116
5107 :063,005,167,096,160,000,222
5113 :177,038,133,002,160,000,247
5119 :177,038,073,128,145,038,086
5125 :032,150,018,173,141,002,009
5131 :041,004,240,009,165,197,155
5137 :201,064,240,003,076,161,250
5143 :020,032,228,255,208,013,011
5149 :165,162,041,016,240,229,114
5155 :169,000,133,162,076,253,060
5161 :019,170,160,000,165,002,045
5167 :145,038,224,095,208,012,001
5173 :032,007,022,169,032,160,219
5179 :000,145,038,076,247,019,072
5185 :173,130,036,240,007,138,021
5191 :072,032,187,019,104,170,143
5197 :138,201,013,208,002,162,033
5203 :095,138,041,127,201,032,205
5209 :144,092,224,160,208,002,151
5215 :162,032,138,072,173,131,035
5221 :036,240,003,032,007,025,188
5227 :104,032,235,019,160,000,145
5233 :145,038,032,150,018,056,040
5239 :165,038,237,134,036,133,094
5245 :167,165,039,237,135,036,136
5251 :005,167,144,014,165,038,152
5257 :105,000,141,134,036,165,206
5263 :039,105,000,141,135,036,087
5269 :230,038,208,002,230,039,128
5275 :032,067,021,076,247,019,105
5281 :160,000,165,002,145,038,159
5287 :024,165,197,105,064,170,124
5293 :132,162,165,162,201,006,233
5299 :208,250,132,198,138,174,255
5305 :217,020,221,217,020,240,096
5311 :006,202,208,248,076,247,154
5317 :019,202,138,010,170,169,137
5323 :019,072,169,246,072,189,202
5329 :254,020,072,189,253,020,249
5335 :072,096,035,029,157,137,229
5341 :133,099,085,138,134,020,062
5347 :148,082,019,076,147,135,066
5353 :139,113,136,140,091,145,229
5359 :017,121,074,090,097,077,203
5365 :070,118,072,081,108,107,033
```

```
5371 :110,003,252,021,006,022,153
5377 :018,022,076,022,162,022,067
5383 :193,022,208,022,055,023,018
5389 :094,024,006,025,133,024,063
5395 :203,024,068,025,092,025,200
5401 :122,025,149,025,241,025,100
5407 :255,027,242,026,083,028,180
5413 :208,022,055,023,127,028,244
5419 :120,029,013,030,134,022,135
5425 :098,030,219,027,108,033,052
5431 :121,024,029,030,120,023,146
5437 :211,033,049,035,245,024,146
5443 :032,165,021,056,165,038,032
5449 :237,128,036,133,167,165,171
5455 :039,237,129,036,005,167,180
5461 :176,032,056,173,128,036,174
5467 :237,019,018,133,167,173,070
5473 :129,036,237,020,018,005,030
5479 :167,240,013,165,038,141,099
5485 :128,036,165,039,141,129,235
5491 :036,032,150,018,056,173,068
5497 :138,036,229,038,133,155,082
5503 :173,139,036,229,039,133,108
5509 :156,005,155,240,002,176,099
5515 :024,024,173,128,036,109,121
5521 :061,003,141,128,036,173,175
5527 :129,036,105,000,141,129,179
5533 :036,032,150,018,076,119,076
5539 :021,096,056,173,134,036,167
5545 :237,021,018,133,167,173,150
5551 :135,036,237,022,018,005,116
5557 :167,144,012,173,021,018,204
5563 :141,134,036,173,022,018,199
5569 :141,135,036,056,165,038,252
5575 :237,019,018,133,167,165,170
5581 :039,237,020,018,005,167,179
5587 :176,011,173,019,018,133,229
5593 :038,173,020,018,133,039,126
5599 :096,056,165,038,237,134,181
5605 :036,133,167,165,039,237,238
5611 :135,036,005,167,176,001,243
5617 :096,173,134,036,133,038,083
5623 :173,135,036,133,039,096,091
5629 :230,038,208,002,230,039,232
5635 :032,067,021,096,165,038,166
5641 :208,002,198,039,198,038,180
5647 :032,067,021,096,165,038,178
5653 :133,155,165,039,133,156,034
5659 :198,156,160,255,177,155,104
```

```
5665 :201,032,240,004,201,031,230
5671 :208,003,136,208,243,177,246
5677 :155,201,032,240,008,201,114
5683 :031,240,004,136,208,243,145
5689 :096,132,167,056,165,155,060
5695 :101,167,133,038,165,156,055
5701 :105,000,133,039,032,067,189
5707 :021,096,160,000,177,038,055
5713 :201,032,240,008,201,031,026
5719 :240,004,200,208,243,096,054
5725 :200,240,026,177,038,201,207
5731 :032,240,247,201,031,240,066
5737 :243,024,152,101,038,133,028
5743 :038,165,039,105,000,133,079
5749 :039,032,067,021,096,173,033
5755 :134,036,133,038,173,135,004
5761 :036,133,039,076,118,022,041
5767 :169,000,141,128,036,173,014
5773 :135,036,056,233,004,205,042
5779 :020,018,176,003,173,020,045
5785 :018,141,129,036,032,150,147
5791 :018,076,122,022,238,141,008
5797 :036,173,141,036,041,015,095
5803 :141,141,036,010,010,010,007
5809 :010,133,167,173,141,036,069
5815 :041,007,024,105,008,101,213
5821 :167,141,015,144,096,238,222
5827 :140,036,173,140,036,041,249
5833 :007,141,140,036,032,150,195
5839 :018,096,165,038,133,155,044
5845 :165,039,133,156,198,156,036
5851 :160,255,177,155,201,046,189
5857 :240,012,201,033,240,008,191
5863 :201,063,240,004,201,031,203
5869 :208,004,136,208,235,096,100
5875 :177,155,201,046,240,027,065
5881 :201,033,240,023,201,063,242
5887 :240,019,201,031,240,015,233
5893 :136,208,235,198,156,165,079
5899 :156,205,019,018,176,226,043
5905 :076,042,023,132,167,198,143
5911 :167,200,240,010,177,155,204
5917 :201,032,240,247,136,076,193
5923 :058,022,164,167,076,243,253
5929 :022,173,019,018,133,038,188
5935 :173,020,018,133,039,032,206
5941 :067,021,096,160,000,177,062
5947 :038,201,046,240,029,201,046
5953 :033,240,025,201,063,240,099
```

```
5959 :021,201,031,240,017,200,013
5965 :208,235,230,039,165,039,225
5971 :205,135,036,240,226,144,045
5977 :224,076,122,022,200,240,205
5983 :250,177,038,201,032,240,009
5989 :247,201,046,240,243,201,255
5995 :033,240,239,201,063,240,099
6001 :235,201,031,240,231,076,103
6007 :106,022,173,023,018,141,090
6013 :076,037,173,024,018,141,082
6019 :077,037,032,203,019,169,156
6025 :091,160,035,032,097,019,059
6031 :169,001,141,130,036,096,204
6037 :056,165,038,237,019,018,170
6043 :133,167,165,039,237,020,148
6049 :018,005,167,208,003,104,154
6055 :104,096,165,038,133,251,186
6061 :165,039,133,252,096,056,146
6067 :165,038,133,253,073,255,072
6073 :101,251,141,080,037,165,192
6079 :039,133,254,073,255,101,022
6085 :252,141,081,037,165,251,100
6091 :141,082,037,165,252,141,253
6097 :083,037,165,253,141,084,204
6103 :037,133,251,165,254,141,172
6109 :085,037,133,252,056,173,189
6115 :081,037,109,077,037,205,005
6121 :026,018,144,020,032,203,164
6127 :019,169,106,160,035,032,248
6133 :097,019,169,001,141,130,034
6139 :036,169,000,133,198,096,115
6145 :173,076,037,133,253,173,078
6151 :077,037,133,254,173,080,249
6157 :037,133,180,024,109,076,060
6163 :037,141,076,037,173,081,052
6169 :037,133,181,109,077,037,087
6175 :141,077,037,032,027,018,107
6181 :173,082,037,133,251,173,118
6187 :083,037,133,252,173,084,037
6193 :037,133,253,173,085,037,255
6199 :133,254,056,173,134,036,073
6205 :229,253,133,180,173,135,140
6211 :036,229,254,133,181,032,164
6217 :027,018,056,173,134,036,005
6223 :237,080,037,141,134,036,232
6229 :173,135,036,237,081,037,016
6235 :141,135,036,096,032,149,168
6241 :023,032,007,022,032,178,135
6247 :023,056,173,076,037,233,189
```

```
6253 :001,141,076,037,173,077,102
6259 :037,233,000,141,077,037,128
6265 :096,032,253,021,032,149,192
6271 :023,032,007,022,076,178,209
6277 :023,032,121,023,169,002,247
6283 :032,223,019,032,203,019,155
6289 :169,118,160,035,032,097,244
6295 :019,032,228,255,240,251,152
6301 :072,032,187,019,104,041,100
6307 :191,201,023,208,009,032,059
6313 :149,023,032,019,022,076,234
6319 :178,023,201,019,208,009,045
6325 :032,149,023,032,209,022,136
6331 :076,178,023,201,016,208,121
6337 :009,032,149,023,032,150,076
6343 :025,076,178,023,096,056,141
6349 :165,038,237,128,036,133,174
6355 :167,165,039,237,129,036,216
6361 :005,167,240,011,173,128,173
6367 :036,133,038,173,129,036,000
6373 :133,039,096,173,019,018,195
6379 :133,038,173,020,018,133,238
6385 :039,032,067,021,096,160,144
6391 :005,140,102,037,032,007,058
6397 :025,172,102,037,136,208,165
6403 :244,076,077,022,024,165,099
6409 :038,133,251,105,001,133,158
6415 :253,165,039,133,252,105,194
6421 :000,133,254,056,173,134,003
6427 :036,229,253,133,180,173,007
6433 :135,036,229,254,133,181,233
6439 :201,255,208,006,169,001,111
6445 :133,180,230,181,032,088,121
6451 :018,160,000,169,032,145,063
6457 :038,238,134,036,208,003,202
6463 :238,135,036,076,118,022,176
6469 :173,131,036,073,006,141,117
6475 :131,036,096,169,133,160,032
6481 :035,032,097,019,032,228,012
6487 :255,240,251,201,089,096,195
6493 :032,203,019,169,144,160,052
6499 :035,032,097,019,169,002,197
6505 :032,223,019,032,078,025,002
6511 :240,004,032,187,019,096,177
6517 :162,255,154,076,013,018,027
6523 :160,000,177,038,201,031,218
6529 :240,015,200,208,247,230,245
6535 :039,165,039,205,135,036,242
6541 :144,238,076,122,022,200,175
```

```
6547 :076,106,022,165,038,133,175
6553 :155,165,039,133,156,198,231
6559 :156,160,255,177,155,201,239
6565 :031,240,017,136,192,255,012
6571 :208,245,198,156,165,156,019
6577 :205,020,018,176,236,076,140
6583 :042,023,056,152,101,155,200
6589 :133,155,169,000,101,156,135
6595 :133,156,056,165,155,229,065
6601 :038,133,167,165,156,229,065
6607 :039,005,167,208,018,132,008
6613 :167,024,165,155,229,167,096
6619 :133,155,165,156,233,000,037
6625 :133,156,076,168,025,165,180
6631 :155,133,038,165,156,133,243
6637 :039,032,067,021,096,173,153
6643 :141,002,041,001,208,003,127
6649 :032,121,023,032,203,019,167
6655 :169,155,160,035,032,097,135
6661 :019,160,000,177,038,073,216
6667 :128,145,038,032,150,018,010
6673 :160,000,177,038,073,128,081
6679 :145,038,169,002,032,223,120
6685 :019,032,228,255,240,251,030
6691 :009,064,201,087,208,009,101
6697 :032,079,026,032,077,022,053
6703 :076,094,026,201,083,208,223
6709 :009,032,079,026,032,056,031
6715 :023,076,094,026,201,080,047
6721 :208,009,032,079,026,032,195
6727 :123,025,076,094,026,076,235
6733 :187,019,165,038,133,253,104
6739 :141,247,036,165,039,133,076
6745 :254,141,248,036,096,056,152
6751 :165,038,133,251,237,247,142
6757 :036,141,080,037,165,039,087
6763 :133,252,237,248,036,141,130
6769 :081,037,032,201,023,173,148
6775 :247,036,133,038,173,248,226
6781 :036,133,039,032,150,018,021
6787 :076,006,026,169,022,229,147
6793 :211,141,136,036,169,000,062
6799 :141,104,037,160,000,169,242
6805 :166,032,210,255,169,157,114
6811 :032,210,255,140,137,036,197
6817 :032,228,255,240,251,172,059
6823 :137,036,133,167,169,032,073
6829 :032,210,255,169,157,032,004
6835 :210,255,165,167,201,013,166
```

```
6841 :240,046,201,020,208,015,147
6847 :136,016,004,200,076,148,003
6853 :026,169,157,032,210,255,022
6859 :076,148,026,041,127,201,054
6865 :032,144,192,204,136,036,185
6871 :240,187,165,167,153,182,029
6877 :036,032,210,255,169,000,155
6883 :133,212,200,076,148,026,254
6889 :032,210,255,169,000,153,028
6895 :182,036,152,096,032,203,172
6901 :019,169,183,160,035,032,075
6907 :097,019,032,087,027,176,177
6913 :033,173,019,018,133,155,020
6919 :173,020,018,133,156,174,169
6925 :134,036,172,135,036,169,183
6931 :155,032,216,255,176,010,095
6937 :032,183,255,041,191,208,167
6943 :003,076,068,028,240,036,226
6949 :173,086,027,201,008,144,164
6955 :006,032,170,033,076,067,171
6961 :027,173,086,027,201,001,052
6967 :240,249,032,203,019,169,199
6973 :189,160,035,032,097,019,081
6979 :169,001,141,130,036,096,128
6985 :032,203,019,169,200,160,088
6991 :035,032,097,019,076,067,149
6997 :027,000,032,134,026,240,032
7003 :024,169,231,160,035,032,230
7009 :097,019,032,228,255,240,200
7015 :251,162,008,201,068,240,009
7021 :012,162,001,201,084,240,041
7027 :006,032,187,019,104,104,055
7033 :096,142,086,027,169,001,130
7039 :160,000,032,186,255,160,152
7045 :000,224,001,240,042,185,057
7051 :182,036,201,064,208,007,069
7057 :185,183,036,201,058,240,024
7063 :028,169,048,141,222,036,027
7069 :169,058,141,223,036,185,201
7075 :182,036,153,224,036,200,226
7081 :204,137,036,144,244,240,150
7087 :242,200,076,192,027,185,073
7093 :182,036,153,222,036,200,242
7099 :204,137,036,208,244,140,132
7105 :246,036,032,203,019,169,130
7111 :182,160,036,032,097,019,213
7117 :173,246,036,162,222,160,180
7123 :036,032,189,255,169,013,137
7129 :076,210,255,032,203,019,244
```

```
7135 :169,178,16Ø,Ø35,Ø32,Ø97,126
7141 :Ø19,Ø32,228,255,24Ø,251,23Ø
7147 :Ø32,235,Ø19,ØØ9,128,Ø72,218
7153 :173,131,Ø36,24Ø,ØØ3,Ø32,Ø88
7159 :ØØ7,Ø25,Ø32,187,Ø19,1Ø4,1Ø9
7165 :Ø76,111,Ø2Ø,Ø56,165,Ø38,2Ø7
7171 :237,Ø19,Ø18,133,155,165,218
7177 :Ø39,237,Ø2Ø,Ø18,ØØ5,155,227
7183 :133,155,24Ø,ØØ4,169,ØØ6,21Ø
7189 :133,155,Ø32,2Ø3,Ø19,169,22Ø
7195 :251,16Ø,Ø35,Ø32,Ø97,Ø19,1Ø9
72Ø1 :165,155,Ø32,223,Ø19,Ø32,147
72Ø7 :Ø87,Ø27,165,155,2Ø8,ØØ3,172
7213 :Ø32,Ø39,Ø19,169,ØØØ,166,214
7219 :Ø38,164,Ø39,Ø32,213,255,Ø24
7225 :144,ØØ3,Ø76,Ø35,Ø27,142,228
7231 :134,Ø36,14Ø,135,Ø36,Ø32,Ø64
7237 :231,255,Ø32,2Ø3,Ø19,169,21Ø
7243 :221,16Ø,Ø35,Ø32,Ø97,Ø19,127
7249 :Ø76,Ø67,Ø27,Ø32,2Ø3,Ø19,249
7255 :169,ØØ1,16Ø,Ø36,Ø32,Ø97,Ø7Ø
7261 :Ø19,Ø32,Ø87,Ø27,169,ØØ1,172
7267 :174,Ø19,Ø18,172,Ø2Ø,Ø18,ØØ8
7273 :Ø32,213,255,Ø32,183,255,Ø51
7279 :Ø41,191,24Ø,2Ø9,Ø32,2Ø3,ØØ3
7285 :Ø19,169,2Ø8,16Ø,Ø35,Ø32,228
7291 :Ø97,Ø19,Ø76,Ø67,Ø27,169,Ø66
7297 :147,Ø32,21Ø,255,169,Ø13,187
73Ø3 :Ø32,21Ø,255,Ø32,172,Ø28,Ø96
73Ø9 :169,Ø13,Ø32,21Ø,255,169,221
7315 :ØØ9,16Ø,Ø36,Ø32,Ø97,Ø19,244
7321 :Ø32,228,255,2Ø1,Ø13,2Ø8,Ø66
7327 :249,Ø76,187,Ø19,Ø32,2Ø4,158
7333 :255,169,ØØ1,Ø32,195,255,Ø48
7339 :Ø96,Ø32,231,255,169,ØØ1,187
7345 :162,ØØ8,16Ø,ØØØ,Ø32,186,213
7351 :255,169,ØØ2,162,Ø24,16Ø,187
7357 :Ø36,Ø32,189,255,Ø32,192,157
7363 :255,176,221,162,ØØ1,Ø32,Ø18
7369 :198,255,Ø32,2Ø7,255,Ø32,156
7375 :2Ø7,255,Ø32,2Ø7,255,Ø32,171
7381 :183,255,2Ø8,2Ø2,Ø32,2Ø7,Ø2Ø
7387 :255,24Ø,197,Ø32,2Ø4,255,122
7393 :Ø32,228,255,2Ø1,Ø32,2Ø8,157
7399 :ØØ5,Ø32,228,255,24Ø,251,218
74Ø5 :162,ØØ1,Ø32,198,255,Ø32,149
7411 :2Ø7,255,Ø72,Ø32,2Ø7,255,247
7417 :168,1Ø4,17Ø,152,Ø32,2Ø5,Ø56
7423 :221,169,Ø32,Ø32,21Ø,255,15Ø
```

```
7429 :032,207,255,240,006,032,009
7435 :210,255,076,005,029,169,243
7441 :013,032,210,255,076,209,044
7447 :028,162,000,142,249,036,128
7453 :142,250,036,142,251,036,118
7459 :056,177,155,233,048,144,080
7465 :042,201,010,176,038,014,010
7471 :249,036,046,250,036,014,166
7477 :249,036,046,250,036,014,172
7483 :249,036,046,250,036,014,178
7489 :249,036,046,250,036,013,183
7495 :249,036,141,249,036,200,214
7501 :208,212,230,156,076,035,226
7507 :029,248,173,249,036,013,063
7513 :250,036,240,023,056,173,099
7519 :249,036,233,001,141,249,236
7525 :036,173,250,036,233,000,061
7531 :141,250,036,238,251,036,035
7537 :076,085,029,173,251,036,251
7543 :216,096,056,173,076,037,005
7549 :237,023,018,141,078,037,147
7555 :173,077,037,237,024,018,185
7561 :141,079,037,013,078,037,010
7567 :208,016,032,203,019,169,022
7573 :034,160,036,032,097,019,015
7579 :169,001,141,130,036,096,216
7585 :024,165,038,133,251,109,113
7591 :078,037,133,253,165,039,104
7597 :133,252,109,079,037,133,148
7603 :254,056,173,134,036,229,037
7609 :251,133,180,173,135,036,069
7615 :229,252,133,181,024,101,087
7621 :254,205,022,018,144,016,088
7627 :032,203,019,169,026,160,044
7633 :036,032,097,019,169,001,051
7639 :141,130,036,096,032,088,226
7645 :018,024,173,078,037,133,172
7651 :180,109,134,036,141,134,193
7657 :036,173,079,037,133,181,104
7663 :109,135,036,141,135,036,063
7669 :165,038,133,253,165,039,014
7675 :133,254,173,023,018,133,217
7681 :251,173,024,018,133,252,084
7687 :032,027,018,032,067,021,204
7693 :096,160,000,177,038,170,142
7699 :200,177,038,136,145,038,241
7705 :200,138,145,038,096,160,034
7711 :000,177,038,041,063,240,078
7717 :010,201,027,176,006,177,122
```

```
7723 :038,073,064,145,038,076,221
7729 :253,021,133,167,041,064,216
7735 :010,005,167,041,191,133,090
7741 :167,041,032,073,032,010,160
7747 :005,167,096,005,075,066,225
7753 :005,058,002,001,027,014,180
7759 :015,018,000,000,000,000,112
7765 :000,032,203,019,169,078,074
7771 :160,036,076,097,019,076,043
7777 :121,031,169,004,141,105,156
7783 :037,160,007,173,141,002,111
7789 :041,001,240,054,032,203,168
7795 :019,169,053,160,036,032,072
7801 :097,019,032,228,255,240,224
7807 :251,056,233,048,201,003,151
7813 :144,217,201,008,176,213,068
7819 :141,105,037,032,203,019,164
7825 :169,062,160,036,032,097,189
7831 :019,032,228,255,240,251,152
7837 :056,233,048,048,190,201,165
7843 :010,176,186,168,169,001,105
7849 :174,105,037,032,186,255,190
7855 :169,000,032,189,255,032,084
7861 :086,030,032,192,255,162,170
7867 :001,032,201,255,144,003,055
7873 :076,121,031,173,019,018,119
7879 :133,155,173,020,018,133,063
7885 :156,162,000,142,087,037,021
7891 :142,086,037,142,106,037,249
7897 :142,107,037,142,100,037,014
7903 :189,070,030,157,092,037,030
7909 :232,224,007,208,245,169,034
7915 :255,141,101,037,160,000,161
7921 :177,155,016,003,076,071,227
7927 :032,201,031,240,034,153,170
7933 :252,036,200,238,099,037,091
7939 :173,099,037,205,093,037,135
7945 :144,230,136,140,133,036,060
7951 :177,155,201,032,240,009,061
7957 :206,099,037,136,208,244,183
7963 :172,133,036,140,133,036,165
7969 :152,056,101,155,133,155,017
7975 :165,156,105,000,133,156,242
7981 :160,000,173,101,037,201,205
7987 :255,208,003,032,234,031,046
7993 :032,036,032,173,133,036,243
7999 :141,132,036,169,252,133,158
8005 :169,169,036,133,170,032,010
8011 :108,034,032,051,032,173,249
```

```
8017 :101,037,205,096,037,144,189
8023 :003,032,146,031,056,165,008
8029 :155,237,134,036,133,167,187
8035 :165,156,237,135,036,005,065
8041 :167,240,035,144,033,169,125
8047 :000,141,086,037,141,095,099
8053 :037,032,146,031,032,225,108
8059 :255,240,251,169,013,032,059
8065 :210,255,032,204,255,169,230
8071 :001,032,195,255,032,187,069
8077 :019,096,076,239,030,056,145
8083 :173,094,037,237,101,037,058
8089 :168,136,136,240,010,048,123
8095 :008,169,013,032,210,255,078
8101 :136,208,248,173,087,037,030
8107 :240,019,141,132,036,173,144
8113 :090,037,133,169,173,091,102
8119 :037,133,170,032,036,032,111
8125 :032,108,034,169,013,032,065
8131 :210,255,032,210,255,032,165
8137 :210,255,173,098,037,208,158
8143 :026,032,204,255,032,203,191
8149 :019,169,087,160,036,032,204
8155 :097,019,032,228,255,240,066
8161 :251,032,086,030,162,001,019
8167 :032,201,255,238,100,037,070
8173 :173,086,037,240,019,141,165
8179 :132,036,173,088,037,133,074
8185 :169,173,089,037,133,170,252
8191 :032,036,032,032,108,034,017
8197 :169,013,032,210,255,172,088
8203 :095,037,140,101,037,136,045
8209 :136,240,015,048,013,169,126
8215 :013,032,210,255,032,225,022
8221 :255,240,018,136,208,243,105
8227 :096,169,032,172,092,037,121
8233 :140,099,037,032,210,255,046
8239 :136,208,250,096,172,097,238
8245 :037,024,152,109,101,037,001
8251 :141,101,037,169,013,032,040
8257 :210,255,136,208,250,096,196
8263 :141,103,037,041,127,032,040
8269 :051,030,201,049,144,007,047
8275 :201,058,176,003,076,076,161
8281 :033,174,160,032,221,160,101
8287 :032,240,012,202,208,248,013
8293 :206,099,037,173,103,037,244
8299 :076,248,030,202,138,010,043
8305 :170,140,102,037,169,032,251
```

```
8311 :072,169,132,072,189,172,157
8317 :032,072,189,171,032,072,181
8323 :096,056,173,102,037,101,184
8329 :155,133,155,165,156,105,238
8335 :000,133,156,076,239,030,009
8341 :177,155,201,031,240,001,186
8347 :136,140,102,037,096,010,164
8353 :087,065,076,082,084,066,109
8359 :083,078,072,070,190,032,180
8365 :199,032,208,032,218,032,126
8371 :228,032,238,032,248,032,221
8377 :002,033,008,033,046,033,084
8383 :169,000,141,098,037,200,068
8389 :076,149,032,200,169,001,056
8395 :141,106,037,076,149,032,232
8401 :200,032,024,029,141,092,215
8407 :037,076,149,032,200,032,229
8413 :024,029,141,093,037,076,109
8419 :149,032,200,032,024,029,181
8425 :141,095,037,076,149,032,251
8431 :200,032,024,029,141,096,249
8437 :037,076,149,032,200,032,003
8443 :024,029,141,097,037,076,143
8449 :149,032,238,102,037,076,123
8455 :146,031,056,152,101,155,136
8461 :141,088,037,165,156,105,193
8467 :000,141,089,037,032,038,100
8473 :033,056,152,237,102,037,130
8479 :141,086,037,200,076,149,208
8485 :032,200,177,155,201,031,065
8491 :208,249,136,096,056,152,172
8497 :101,155,141,090,037,165,226
8503 :156,105,000,141,091,037,073
8509 :032,038,033,056,152,237,097
8515 :102,037,141,087,037,200,159
8521 :076,149,032,200,177,155,094
8527 :201,061,240,004,136,076,029
8533 :104,032,200,032,024,029,250
8539 :072,173,103,037,041,015,020
8545 :170,202,104,157,077,030,069
8551 :032,149,032,076,132,032,044
8557 :032,231,255,169,000,032,060
8563 :189,255,169,015,162,008,145
8569 :160,015,032,186,255,032,033
8575 :192,255,144,001,096,032,079
8581 :203,019,169,062,032,210,060
8587 :255,032,134,026,240,025,083
8593 :162,015,032,201,255,176,218
8599 :012,169,182,160,036,032,230
```

```
8605 :097,019,169,013,032,210,185
8611 :255,032,231,255,076,187,175
8617 :019,032,231,255,169,000,107
8623 :032,189,255,169,015,162,229
8629 :008,160,015,032,186,255,069
8635 :032,192,255,176,228,032,078
8641 :203,019,162,015,032,198,054
8647 :255,032,134,026,032,231,141
8653 :255,169,001,141,130,036,169
8659 :096,173,141,002,201,005,061
8665 :240,005,173,104,037,208,216
8671 :040,032,203,019,169,108,026
8677 :160,036,032,097,019,032,093
8683 :134,026,208,004,032,187,058
8689 :019,096,169,001,141,104,003
8695 :037,141,130,036,173,019,015
8701 :018,133,155,173,020,018,002
8707 :133,156,076,021,034,165,076
8713 :038,133,155,165,039,133,160
8719 :156,160,001,076,023,034,209
8725 :160,000,162,000,189,182,202
8731 :036,032,235,019,209,155,201
8737 :240,002,162,255,200,208,076
8743 :011,230,156,165,156,205,194
8749 :135,036,240,002,176,035,157
8755 :232,236,137,036,208,224,100
8761 :024,152,101,155,133,038,148
8767 :165,156,105,000,133,039,149
8773 :056,165,038,237,137,036,226
8779 :133,038,165,039,233,000,171
8785 :133,039,076,067,021,032,193
8791 :203,019,169,118,160,036,024
8797 :032,097,019,169,001,141,040
8803 :130,036,169,000,141,104,167
8809 :037,096,096,160,000,204,186
8815 :132,036,240,248,177,169,089
8821 :048,038,032,051,030,032,092
8827 :018,035,032,210,255,173,078
8833 :107,037,240,010,169,008,188
8839 :032,210,255,169,095,032,160
8845 :210,255,032,225,255,208,046
8851 :005,104,104,076,121,031,076
8857 :200,076,110,034,140,102,047
8863 :037,041,127,032,051,030,221
8869 :201,049,144,017,201,058,067
8875 :176,013,041,015,170,202,020
8881 :189,077,030,032,210,255,202
8887 :076,153,034,201,067,208,154
8893 :026,056,169,080,237,132,121
```

```
8899 :036,074,056,237,092,037,215
8905 :168,169,032,032,210,255,043
8911 :136,208,250,172,102,037,088
8917 :076,153,034,201,069,208,186
8923 :017,056,173,093,037,237,064
8929 :132,036,056,237,092,037,047
8935 :168,169,032,076,204,034,146
8941 :201,085,208,008,173,107,251
8947 :037,073,001,141,107,037,127
8953 :201,035,240,003,076,153,189
8959 :034,174,100,037,169,000,001
8965 :032,205,221,160,054,132,041
8971 :001,172,102,037,076,153,040
8977 :034,174,106,037,240,026,122
8983 :133,167,041,127,201,065,245
8989 :144,018,201,091,176,014,161
8995 :170,165,167,041,128,073,011
9001 :128,074,074,133,167,138,243
9007 :005,167,096,032,203,019,057
9013 :056,173,021,018,237,134,180
9019 :036,170,173,022,018,237,203
9025 :135,036,032,205,221,169,095
9031 :001,141,130,036,096,014,233
9037 :008,144,211,080,069,069,146
9043 :068,211,067,082,073,080,152
9049 :084,000,194,085,070,070,080
9055 :069,082,032,195,076,069,106
9061 :065,082,069,068,000,194,067
9067 :085,070,070,069,082,032,003
9073 :198,085,076,076,000,196,232
9079 :069,076,069,084,069,032,006
9085 :040,211,044,215,044,208,119
9091 :041,000,058,211,085,082,096
9097 :069,063,032,217,047,206,003
9103 :000,197,210,193,211,197,127
9109 :032,212,197,216,212,000,250
9115 :197,082,065,083,069,032,171
9121 :040,211,044,215,044,208,155
9127 :041,058,060,210,197,212,177
9133 :213,210,206,062,000,203,043
9139 :069,089,058,000,211,065,159
9145 :086,069,058,000,212,065,163
9151 :080,069,032,197,210,210,221
9157 :207,210,000,211,084,079,220
9163 :080,080,069,068,000,214,202
9169 :069,082,073,070,089,032,112
9175 :197,082,082,079,082,000,225
9181 :206,079,032,069,082,082,003
9187 :079,082,083,000,147,032,138
```

```
9193 :018,212,146,065,080,069,055
9199 :032,079,082,032,018,196,166
9205 :146,073,083,075,063,000,173
9211 :204,079,065,068,058,000,213
9217 :214,069,082,073,070,089,086
9223 :058,000,208,082,069,083,251
9229 :083,032,018,210,197,212,253
9235 :213,210,206,146,000,036,062
9241 :048,206,079,032,210,079,167
9247 :079,077,000,206,079,032,248
9253 :084,069,088,084,032,073,211
9259 :078,032,066,085,070,070,188
9265 :069,082,046,000,196,069,255
9271 :086,073,067,069,032,035,161
9277 :000,211,069,067,079,078,053
9283 :068,046,032,193,068,068,030
9289 :082,046,032,035,000,208,220
9295 :082,073,078,084,073,078,035
9301 :071,000,206,069,088,084,091
9307 :032,083,072,069,069,084,244
9313 :044,032,146,210,197,212,170
9319 :213,210,206,018,000,200,182
9325 :085,078,084,032,070,079,025
9331 :082,058,000,206,079,084,112
9337 :032,198,079,085,078,068,149
9343 :000,000,000,000,000,000,127
```

Figure 2. Clip-Out Function Key Overlay

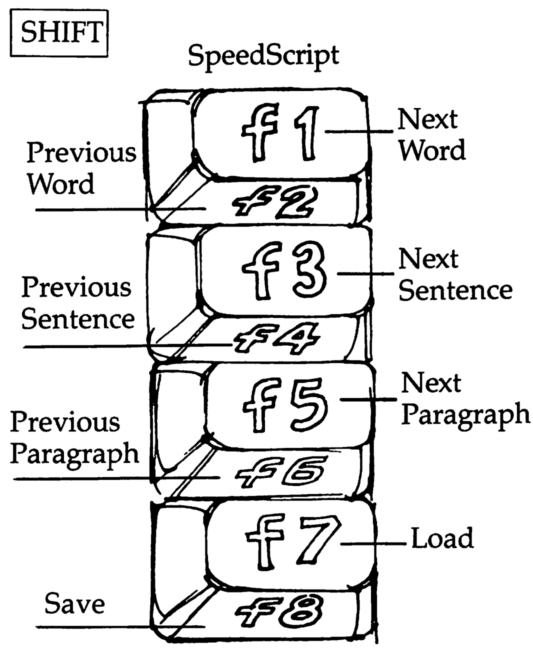


Table 1. Clip-Out Quick Reference Card—Editing Commands

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
f1: Next word
f2: Previous word
f3: Next sentence
f4: 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
CLR/HOME: Erase All
←: Backspace
CTRL-←: 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	see text	

Chapter 4

Graphics

Introduction to Custom Characters

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.

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.

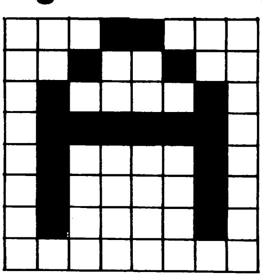
Character Sets

First, let's clarify exactly what is a *character set.* 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 the 64 special graphics characters that are pictured on the front of the keys. In all, the VIC 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.)

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 VIC 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 is 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 anyway you want (we'll cover this in detail in a moment).

Okay 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 VIC 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.

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.

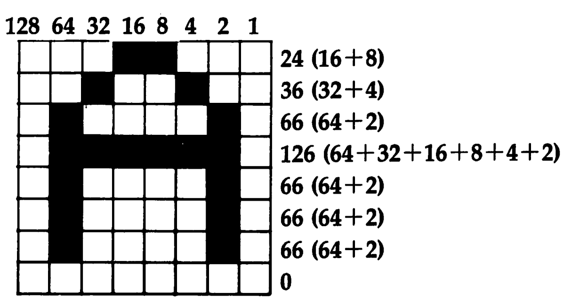
For example, exactly how are characters customized?

Recall that the character set is defined by a list of numbers which describe 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, but even if you don't understand binary, look at Figure 2.

The eight numbers running vertically along the right side of Figure 2 are the numbers which define the dot pattern for an A. 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 2. Dot Pattern A



Along the top of Figure 2, running horizontally *from right to left,* are eight more numbers: 1, 2, 4, 8, 16, 32, 64, and 128. Notice that each of the numbers in this series is twice as much as its predecessor. That's how binary works. (If you want to get technical, each number represents a *bit* in a *byte*.)

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 36, because the colored dots in the second row of the grid fall beneath the 4 and 32, which add up to 36. 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.)

Study these figures until you're sure you know how to add up the dot patterns to arrive at the eight numbers along the right. This is the key to customizing characters.

Customizing Characters

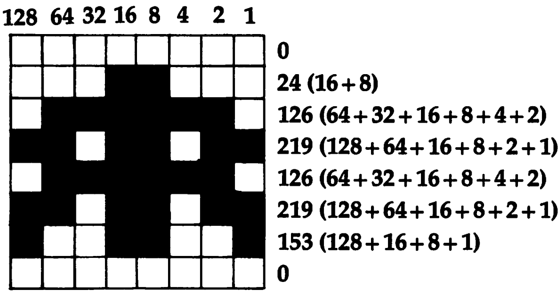
Once you understand how character patterns work, it's easy to customize them at will.

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 Figures 2 and 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 3 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.

Figure 3. Dot Pattern for a Customized Character (*Space Invaders*-Type Alien)



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 (although on the unexpanded VIC, memory limits make it practical to customize only 64 characters).

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 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.

How to Make Custom Characters

Gregg Keizer

Before reading this article, be sure to read the previous article "Introduction to Custom Characters," especially if you're unfamiliar with the concepts of redefined characters.

The standard characters provided with the VIC are certainly useful, afford plenty of variety, and can be combined to create new shapes and figures. Many games on the VIC, for instance, often use only the standard character set to display the screen and show objects or user-controlled figures.

But there will be times when you need to draw a new shape or figure that the standard character set just can't produce. You'll often find this true as you design your own games. Or perhaps you simply want to experiment, to see what you can do with the VIC.

Creating custom characters takes up more memory, which can be crucial when you're using the VIC, and it can take time to design and add them to a program. But when you're looking for just the right figure, and it shows on the screen during a game, you'll agree that it was worth the effort.

Fooling the VIC

You've already looked through the article "Introduction to Custom Characters," so you know what custom characters are and how to design them using graph paper. Now that you have the figures in mind, you can actually begin to place those custom characters in the VIC.

Remember that the character set of most computers, including the VIC, is located in Read Only Memory, or ROM, and is permanently stored there. The VIC's character memory begins at location 32768, which stores the number value of the top row of the @ character. The number value of the second row of that character is at location 32769, the third row at location 32770, and so on.

In order to change the character set and insert your own custom characters, you first need to change the place where the VIC looks to find its character set. You can do this by changing the *pointer,* which fortunately is in a Random Access Memory (RAM) location. By changing this memory location, you are in effect instructing the computer to look elsewhere for its character set.

The VIC looks to location 36869 for its pointer. Although the pointer's value is usually 240 or 242, it can be changed by POKEing a new value into that location. Entering POKE 36869,255, for example, fools the computer into looking to a new location in RAM, 7168, for character data, instead of the ROM location 32768. You can begin your custom character set in a RAM location ranging from 4096 to 7168, but the best place to start is at 7168.

Right now, however, there is nothing there for the VIC to look at. You need to copy the character set data to your new RAM location. As explained in "Introduction to Custom Characters," it isn't that hard.

What you need to do is free enough space from BASIC to fit in your recopied character set, as well as protect it from the BASIC's operations. Then you have to tell the VIC to read the numbers in the character set stored in ROM and copy those numbers starting at your new RAM location, 7168.

A short routine such as the one below does all the work for you in only three program lines.

```
10 PRINT"{CLR}":POKE 36869,255
20 POKE 52,28:POKE 56,28:CLR
30 FOR I=7168 TO 7679:POKE I, PEEK(25600+I)
```

Line 10 tells the VIC to go to location 7168 to get the data for its character set, rather than looking to ROM location 32768. Line 20 frees 512 bytes of memory from BASIC by changing the pointers to the top of available RAM memory. A value of 28 takes 512 bytes, just enough for the 64 characters you would normally copy to RAM, and also protects this area from BASIC. The last line copies the first 64 characters from their ROM locations to your new RAM location. This is done by PEEKing at the values from 32768 to 33280 (PEEK 25600+I), and then POKEing those values into the new locations running from

7168 to 7679. (We're moving only 64 characters out of a total character set of 256 in order to conserve memory.)

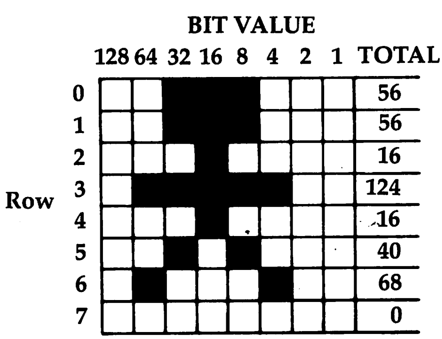
Now that you have part of the character set moved to RAM, the VIC told to look there from now on for its character data, and the area protected from BASIC, you can begin to place your own characters in this set.

Placing Your Custom Characters

Your custom characters have already been designed, using graph paper. You've added up the dot values and should have eight numbers for each new character. These represent the eight bytes of memory each character requires to be displayed on the screen.

What you now need to do is POKE these new numbers where the old values are, replacing one of the standard characters with one of your custom characters. For example, a custom character such as that in the figure would have the following values:

Custom Characters



The eight numbers to POKE into a memory location are 56, 56, 16, 124, 16, 40, 68, and 0 to create this character.

The most convenient way to replace old characters is with DATA and READ statements. By placing all the new numbers into DATA statements, and then having the VIC READ them, your programming task will be simplified. The computer always READs the DATA in the order it's listed, so be sure the numbers are in the right order, and that there are eight numbers for each character. The DATA statements can be anywhere in the program, as long as they are in the same order as the READ commands.

Glance through the Screen Code (Appendix H). You've copied the first 64 characters—from the @ to the ?—into RAM, but you'll lose some of these when you replace them with your custom characters. Decide which standard characters you won't need, and look up the screen codes for those. If you are designing a game that uses some of the letters in a display, for example, make sure those won't be lost when you develop your custom characters.

Because you changed the pointer, your character set now begins at memory location 7168. Each character takes up eight bytes of memory, so by multiplying the screen code number by 8 and adding it to 7168, you can find the location of the top row of any character. For example, the letter A, with a screen code value of 1, begins at location 7176.

A format you can use to replace a standard character with your own custom character is:

```
FOR C(your new character)=X TO X+7:READ D:POKE
C,D:NEXT
```

where X is the memory location of a character you want to replace. To replace the A character with the custom character from the figure, for instance, you would write:

```
FOR C=7176 TO 7183:READ D:POKE C,D:NEXT
```

and include somewhere in the program the DATA statement:

```
DATA 56,56,16,124,16,40,68,0
```

Add these two lines to the program used to copy characters into RAM, and you'll see the custom figure every time you press the A key. Notice, however, that you have lost the A character. There is now no way to print that on the screen. In other words, make sure that the characters you replace are ones you won't want to use.

If you have several new characters and they are replacing standard characters right after each other on the Screen Code table, you can place more than one in a READ statement, simplifying your programming. Replacing the first five standard characters, for example, would look like this:

```
FOR C=7168 TO 7207:READ D:POKE C,D:NEXT
```

You would then need five DATA statements, one for each new character created.

8K Expanded VIC

If you have an expanded VIC, with 8K or more of RAM, you'll need to enter additional commands before you run any program which copies a character set and creates custom characters.

For a VIC with an 8K or more memory expansion, enter these few additional POKEs *before* you load and run any program creating custom characters. Enter each individual POKE, then press RETURN:

```
POKE 43,1:POKE 44,32
POKE 8192,Ø:NEW
POKE 36869,24Ø:POKE 36866,15Ø
POKE 648,3Ø
```

The first line of POKEs sets the pointer to the start of the BASIC program, much like the POKE 52,28 did in the unexpanded VIC. The second POKE, the first memory location of BASIC, must be set to 0, or you won't be able to run your programs. The third line of POKEs relocates the screen, while the last POKE makes it possible for the operating system to see the screen.

As you enter the last two lines of POKE statements, the screen will change drastically. Don't worry—you haven't done anything wrong. You do, however, need to be careful as you enter these lines, for you can't really see what you've typed on the screen, due to the jumbled display.

Once these are entered, you can load and run your program to copy characters and create custom figures. Line 20 in the program, used to copy characters to RAM, must be eliminated, however, if you use the expanded VIC. If you leave it in, the pointers in BASIC will change again, and you won't see the correct screen display.

Custom Hints

You now have the ability to create your own characters. The graph paper method is an excellent way to begin designing your characters. This method is fine as long as you are designing only a few characters. Using one of the many character editors available (see "Pixelator," *COMPUTE!'s Second Book of VIC*) will make the job much easier.

Custom Characters on the Expanded VIC

Dan Carmichael

The VIC-20 Programmer's Reference Guide *has an entire section on creating custom characters on the unexpanded VIC. However, it only briefly touches upon how to set up the expanded VIC for custom characters. If you want to program custom characters on the expanded VIC (8K or more), there are some important differences to learn.*

Using custom characters on the unexpanded VIC is easy. The way memory is laid out is perfect for it. With BASIC programming memory running from 4096 to 7679, you can partition off, or *reserve,* 512 bytes from the top of BASIC (7168 to 7679), enough for up to 64 custom characters. This, plus the fact that memory is neatly laid out, makes the task easy.

However, getting the (8K or more) expanded VIC-20 set up for custom characters takes a little more work. The start of BASIC programming memory moves from 4096 to 4608, and the area where BASIC was in the unexpanded VIC (4096–4607) is now screen memory. The color memory starting address also moves from 38400 to 37888.

Making custom characters in the expanded configuration should be easy. Just reserve 512 or more bytes at the top of BASIC memory as we did in the unexpanded VIC and go, right? Unfortunately, it's not that easy. The problem is that the VIC chip, the chip which determines where the VIC-20 gets its character information, cannot *see* expansion memory. Because of this limitation, we cannot put our custom characters anywhere in the VIC's expansion RAM.

The answer is to put the custom characters underneath the user BASIC area, in an area of memory accessible to the VIC chip. This is accomplished by moving BASIC memory up and reserving a block of memory for the custom characters.

Moving BASIC

The first part of our task is moving BASIC memory up a page or two (a *page* is a block of 256 bytes in memory). This is done with a few easy POKEs. First we'll POKE memory locations 43 and 44, which signal to the operating system where the *start of BASIC* is. When you add 8K or more expansion to the VIC, the values in 43 and 44 change to 1 and 18, respectively. This signals the system that the start of BASIC is at 4608. To make room for the custom characters, we'll POKE 43,1 and POKE 44,22. This tells the operating system that we now want the start of BASIC to begin at 5632.

Next we'll POKE memory locations 45 and 46. These two bytes tell the system where the *start of variables* is. The start of variables always stays a few bytes just past the end of your BASIC program, no matter how large the program grows. We'll POKE 45,3 and POKE 46,22.

Now we have to tell the operating system where we moved things. Bytes 641 and 642 signal where the *start of memory for the operating system* is. We'll POKE 641,0 and POKE 642,22.

The last thing we have to do is POKE zeros into the beginning of BASIC to signal the operating system that it's ready. We'll POKE 5632, 5633, and 5634 with zeros. These three zeros tell the system that this is the end of the BASIC program. Because there is no BASIC program in memory, the end *is* the beginning.

These POKEs will reserve 512 bytes (from 5120 to 5631) for our custom characters. This is enough memory to hold up to 64 characters.

Using the Program

The program below POKEs a short machine language routine into memory that sets all the necessary parameters in the expanded VIC for custom characters. The program simply performs all the POKEs we just discussed.

Program 1. Memory Setup

```
1 FORA=8192TO8224:READB:POKEA,B:NEXT         :rem 6
2 PRINT"{CLR}{WHT}SYS8192:CLR{BLU}":POKE631,19:POK
  E632,13:POKE198,2                          :rem 62
5 DATA169,Ø,141,129,2,141,Ø,22,141,1,22,141,2,22,1
  69,1,133,43,169,3,133,45,169,22,133        :rem 97
6 DATA44,133,46,141,13Ø,2,96,234             :rem 81
```

When you run this program, make sure no BASIC programs are in memory. You could lose all or part of the other program. Line 2 starts the machine language routine. It does this by PRINTing SYS8192:CLR at the top of the screen. Then, by POKEing CHR$(19) (cursor home) and CHR$(13) (carriage return) into the keyboard buffer (bytes 631–640), it fools the VIC into thinking you typed these commands from the keyboard. POKEing 198,2 tells the operating system to read the characters in the keyboard buffer, starting the machine language routine. This programming technique, known as the *dynamic keyboard,* is a very useful tool.

Type in Program 1, verify it carefully, and SAVE it. Be sure to save it first, because after running, it will seem to disappear. Also check your DATA statements carefully because an error in a machine language program can lock up your VIC.

Now enter RUN. After running, you are ready to load in your BASIC program and create your custom characters.

To switch to the custom characters, POKE 36869,205. To switch back to standard character ROM, POKE 36869,192. If you wish to copy the first 64 characters from standard character ROM into your custom character area, add this line to your program:

```
10000 FOR P=5120 TO 5631: POKE P,PEEK (P+27648):NEXT
```

You can then change or delete them at will.

More Custom Characters

If 64 custom characters are not enough for you, you can enter Program 2. Program 2 works basically the same except it sets aside enough memory for 128 custom characters. They will reside from 5120 to 6143. With Program 2, the start of BASIC will move from 5632 to 6144, giving us the extra memory we need for 64 more characters.

Program 2. Extra Memory Setup

```
1 FORA=8192TO8224:READB:POKEA,B:NEXT           :rem 6
2 PRINT"{CLR}{WHT}SYS8192:CLR{BLU}":POKE631,19:POK
  E632,13:POKE198,2                            :rem 62
5 DATA169,Ø,141,129,2,141,Ø,24,141,1,24,141,2,24,1
  69,1,133,43,169,3,133,45,169,24,133         :rem 1Ø5
6 DATA44,133,46,141,13Ø,2,96,234               :rem 81
```

In Program 2, if you want to copy the first 128 characters from ROM into your custom character area, add this line to your BASIC program:

10000 FOR P=5120 TO 6143:POKE P,PEEK (P+27648): NEXT

Creating Custom Characters

Creating custom characters is up to you. We won't go into the details here, but there are many good resources available, including the *VIC-20 Programmer's Reference Guide* and articles elsewhere in this book.

If you use Program 1, the 64 characters will go into memory between 5120 and 5631, and will correspond to screen POKE characters 0 (@) to 63 (?). If you use Program 2, the 128 characters will go into 5120–6143, and will correspond to characters 0 (@) to 127 ( ).

Programming Multicolor Characters

Bill McDannell

If you know how to create standard programmable characters, you can create four-color characters and multicolor graphics. Here's how to select colors for the screen, border, character, and auxiliary colors. For the unexpanded VIC.

In order to understand the creation of multicolor characters on the VIC-20, you must first have a working knowledge of standard programmable characters. (If you do not know how to define your own characters, read the first two articles in this section: "Introduction to Custom Characters" and "How to Make Custom Characters.")

For standard programmable characters, drawing is done using an eight-by-eight grid. Each point on the grid represents one bit, which is turned either on or off by designating a value of one or zero for the bit.

You can use as many as four colors in one character when using multicolor graphics. Since you must designate one of four color choices, rather than simply on or off, you cannot program each individual bit. However, if adjacent bits are combined to produce a piece of information, you have four choices:

- Both bits off (00)
- First bit off, second on (01)
- First bit on, second off (10)
- Both bits on (11)

You now have the four possibilities necessary to designate four colors, but you have them at the sacrifice of horizontal resolution. Since it takes two bits to specify a color, you will be able to specify only four individual blocks of color across one horizontal line of your character (as opposed to the eight blocks available with a standard character). You still have eight vertical rows available.

Available Colors

Each possible two-bit value corresponds to a specific selectable color.

00=screen color
01=border color
10=character color
11=auxiliary color

For border and character colors, you have the choice of the eight standard VIC colors. For screen and auxiliary colors, you can choose from the 16 colors depicted in the screen and border color chart (see the appendices). More about selecting individual colors later.

First, let's see how we designate our four initial choices. The figure shows the same programmable character in both standard and multicolor mode. Notice that the numerical value of each horizontal byte is the same. The DATA statements you use to create each character are identical. The difference is that in the multicolor mode, each pair of bits is combined and read as one nybble to identify the appropriate color group.

Getting into Multicolor

Accessing multicolor mode and setting the desired character color are done simultaneously. For standard characters, you POKE the appropriate screen location to the desired color, using the numbers zero (black) through seven (yellow). To go into multicolor mode, you simply add eight to the desired color value. This both selects your character color and sets that particular character to multicolor mode. For example, POKEing screen location 38400 to a value of 15 would both change the character color in the upper left corner of the screen to yellow, and turn on the multicolor mode in that space.

Setting border and screen colors is done the same as always: by POKEing 36879 to the desired value from the color chart (Appendix E). POKE 36879,9 will give you a black screen and a white border.

The choice of auxiliary color is made, believe it or not, in the same memory location you use to control volume, with a POKE to location 36878. There are 256 possible values for this POKE location (0–255), and each of the consecutive 16 values corresponds to one of the 16 available colors, in descending order, from the chart.

In other words, any value between 0 and 15 POKEd into location 36878 will produce an auxiliary color of black. Values 16–31 will produce white, and so forth. This creates a slight problem when we're writing a program where we want to control both volume and multicolor graphics. We can solve it with this formula:

POKE 36878,A*15+V

A is the number of the desired color (0 is black, 1 is white, etc.), and V is the desired volume.

That's what you need to know to create multicolor graphics. The rest of the operation is identical to creating standard graphics.

These two programs illustrate how to use multicolor characters. The first program creates a four-color spaceship and moves it down the screen. The spaceship is drawn using two separate characters and POKEing them side by side.

The second program is a coloring game my children seem to love. It allows you to choose the colors in which the character will be drawn. I created the character using a grid that is five characters wide and five deep, and which yields a 20-by-40 area of programmable blocks. The screen and border colors are set to black and white by the program. You select the auxiliary color and three different areas of character color. Because character color blocks are set individually, a multicolor figure consisting of more than one character can be programmed to more than four colors. In this case, I could have selected up to 28 different colors for the figure. Six were sufficient.

Program 1. Four-Color Spaceship

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

```
10 PRINT"{CLR}"                                 :rem 197
100 POKE36869,255                               :rem 152
105 POKE36879,61                                :rem 105
110 FORI=7168TO7679:POKEI,PEEK(I+25600):NEXT
                                                :rem 147
130 FORI=7176TO7191                              :rem 75
150 READA:POKEI,A:NEXT                          :rem 139
154 X=7690:C=30720                              :rem 187
155 POKEX,1:POKEX+C,10:POKEX+1,2:POKEX+C+1,10
                                                :rem 206
```

```
156 FORT=1TO80:NEXT:POKEX,32:POKEX+1,32    :rem 201
157 X=X+22:IFX>8185THEN154                  :rem 23
158 GOTO155                                 :rem 114
160 DATA8,2,5,23,85,93,85,40,32,128,80,212,85,117,
    85,40                                   :rem 199
```

Program 2. Coloring Game

```
10 PRINT"{CLR}"                             :rem 197
20 PRINT"{10 DOWN}{3 SPACES}JUST A MINUTE..."
                                            :rem 192
110 FORI=7168TO7679:POKEI,PEEK(I+25600):NEXT
                                            :rem 147
120 FORI=7176TO7375                         :rem 78
130 READA:POKEI,A:NEXT                      :rem 137
139 POKEX+89,10:POKEX+89+C,10               :rem 165
140 DATA48,252,239,235,235,235,232,232,235,235,235
    ,59,59,15,3                             :rem 249
141 DATA3,3,3,3,3,3,1,5,21,22,21,21,21,21,5,5,1,1,
    0,0,0,0,0,0,0                           :rem 215
142 DATA0,0,252,255,3,60,255,255,245,213,213,213,2
    17,234,230                              :rem 164
143 DATA231,255,255,255,255,252,92,84,85,85,149,16
    5,138,128,96,96                         :rem 229
144 DATA88,88,89,22,5,5,1,1,0,0,0,0,0,255,255,255,
    255,255,255,125,125,125,125,125         :rem 184
145 DATA255,255,255,255,195,0,65,65,0,65,85,85,85,
    170,20,20                               :rem 155
146 DATA40,170,170,85,170,85,85,85,85,0,0,63,255,1
    92,60,255                               :rem 153
147 DATA255,95,87,87,87,103,171,155,219,255,255,25
    5,255,63,53,21                          :rem 169
148 DATA85,85,86,90,162,2,9,9,37,37,101,148,80,80,
    64,64,0                                 :rem 60
149 DATA12,63,251,171,235,235,43,43,235,235,235,23
    6,236,240,192                           :rem 85
150 DATA192,192,192,192,192,192,64,80,84,148,84,84
    ,84,84,80,80,64,64,0,0,0,0,0,0,0        :rem 247
151 PRINT"{CLR}{5 DOWN}HELLO, THERE! MY NAME
    {DOWN}IS FRED, THE SEE-THRU {DOWN}MOUSE. WHAT'
    S YOURS"                                :rem 144
152 PRINT:INPUTN$                           :rem 97
153 PRINT"{CLR}{2 DOWN}WELL,"N$             :rem 241
154 PRINT"{DOWN}I HAPPEN TO LIVE IN{3 SPACES}
    {DOWN}YOUR COMPUTER. THEY{3 SPACES}{DOWN}CALL
    {SPACE}ME A SEE-THRU"                   :rem 111
155 PRINT"{DOWN}MOUSE BECAUSE I'M{5 SPACES}{DOWN}I
    NVISIBLE!"                              :rem 146
```

```
156 PRINT"{DOWN}BUT YOU CAN SEE ME BY {DOWN}PAINTI
    NG ME DIFFERENT {DOWN}COLORS. JUST PRESS THE"
                                            :rem 155
157 PRINT"SPACE BAR TO BEGIN."              :rem 229
158 GETB$:IFB$=""THEN158                     :rem 97
159 IFB$=" "THEN161                         :rem 220
160 GOTO158                                 :rem 110
161 PRINT"{CLR}{DOWN}FIRST LET'S COLOR MY
    {2 SPACES}{DOWN}FACE. PICK A NUMBER."   :rem 198
162 PRINT"{DOWN}1=RED{6 SPACES}8=LT.OR."    :rem 214
163 PRINT"{DOWN}2=CYAN{5 SPACES}9=PINK"     :rem 190
164 PRINT"{DOWN}3=PURPLE{2 SPACES}10=LT.CYAN"
                                             :rem 92
165 PRINT"{DOWN}4=GREEN{3 SPACES}11=LT.PUR."
                                            :rem 242
166 PRINT"{DOWN}5=BLUE{4 SPACES}12=LT.GRN."
                                            :rem 156
167 PRINT"{DOWN}6=YELLOW{2 SPACES}13=LT.BLUE"
                                            :rem 102
168 PRINT"{DOWN}7=ORANGE{2 SPACES}14=LT.YEL."
                                             :rem 57
171 PRINT:INPUTC$:D=VAL(C$)+2                :rem 10
172 IFD<3ORD>16THEN161                       :rem 45
173 PRINT"{CLR}{DOWN}THANK YOU, "N$          :rem 33
174 PRINT"{DOWN}NOW HOW ABOUT MY EARS":GOSUB185
                                              :rem 4
175 PRINT"{CLR}{DOWN}VERY GOOD! NOW MY EYES":GOSUB
    185                                     :rem 202
176 PRINT"{CLR}OKAY, "N$                    :rem 212
177 PRINT"{DOWN}ONE LAST TIME TO COLOR{DOWN}MY MOU
    TH.":GOSUB185:GOTO193                   :rem 194
185 PRINT"{DOWN}1=BLACK":PRINT"{DOWN}2=WHITE":PRIN
    T"{DOWN}3=RED":PRINT"{DOWN}4=CYAN"      :rem 118
186 PRINT"{DOWN}5=PURPLE":PRINT"{DOWN}6=GREEN":PRI
    NT"{DOWN}7=BLUE":PRINT"{DOWN}8=YELLOW":rem 240
187 Y=Y+1:PRINT:INPUTH$(Y):H(Y)=VAL(H$(Y)) :rem 69
188 IFH(Y)<1ORH(Y)>8ANDY=1THENY=0:GOTO173  :rem 53
189 IFH(Y)<1ORH(Y)>8ANDY=2THENY=1:GOTO175  :rem 58
190 IFH(Y)<1ORH(Y)>8ANDY=3THENY=2:GOTO176  :rem 53
191 H(Y)=H(Y)+7                             :rem 30
192 RETURN                                 :rem 124
193 PRINT"{CLR}{DOWN}OKAY,"N$              :rem 228
194 PRINT"{DOWN}HERE WE GO. IF YOU{4 SPACES}{DOWN}
    WANT TO CHANGE MY{5 SPACES}{DOWN}COLORS, PRESS
     THE"                                   :rem 71
195 PRINT"{DOWN}SPACE BAR.":PRINT"{DOWN}AND WHEN Y
    OU WANT TO{2 SPACES}{DOWN}QUIT, PRESS E."
                                           :rem 107
```

```
196 PRINT"{DOWN}BUT TO SEE ME AS YOU{2 SPACES}
    {DOWN}JUST PAINTED ME, PRESS{DOWN}ANY KEY BUT
    {SPACE}THOSE TWO."                          :rem 47
197 GETF$:IFF$=""THEN197                       :rem 111
198 IFF$=" "THENY=Ø:POKE36869,24Ø:POKE36879,27:GOT
    O161                                       :rem 1Ø2
199 IFF$="E"THEN25Ø                             :rem 4Ø
2ØØ PRINT"{CLR}":POKE36869,255                  :rem 55
2Ø1 PRINT"{CLR}":POKE36869,255                  :rem 56
2Ø2 POKE36879,9                                 :rem 57
21Ø POKE36878,D*15+1                            :rem 46
22Ø X=7887:C=3Ø72Ø                             :rem 189
221 FORA=1TO2                                    :rem Ø
222 FORB=ØTO2ØSTEP5                            :rem 162
223 POKEX,B+A:POKEX+C,H(1)                      :rem 29
224 X=X+1                                      :rem 225
225 NEXTB                                       :rem 26
226 X=X+17:NEXTA                               :rem 212
227 FORA=3TO5                                   :rem 11
228 FORB=ØTO2ØSTEP5                            :rem 168
229 POKEX,B+A:POKEX+C,H(3)                      :rem 37
23Ø X=X+1:NEXTB                                :rem 153
231 X=X+17:NEXTA                               :rem 2Ø8
232 POKE7888+C,H(2):POKE7889+C,H(2):POKE791Ø+C,H(2
    ):POKE7911+C,H(2)                          :rem 247
233 POKE789Ø+C,H(2):POKE7912+C,H(2)            :rem 165
234 GOTO197                                    :rem 115
25Ø POKE36869,24Ø:POKE36879,27                 :rem 167
26Ø PRINT"{CLR}{9 DOWN}SO LONG,"N$"!"          :rem 1Ø6
```

PEEK and PRINT

Carolyn D. Bellah

These two programs let you design and display characters four times normal size. You can store up to seven of these larger characters and recall them later for screen displays or a printout.

These two programs allow you to design large custom characters—twice as high and twice as wide as regular characters—save them, and print them.

Program 1 sets up a programmable character grid in which you move your cursor to a desired location on the grid and hit any letter key to print at that location.

Several options are available in Program 1. There is a color choice (1 to 8), an option to erase your created character, and an option to draw another. Seven characters can be stored in a protected area of RAM and can be recalled, printed, and listed in sequence by using Program 2.

Another useful feature is that the program will display decimal PEEK values (the numeric values that are used for DATA statements) that represent your finished character. (A memory location will be displayed when the data is listed on the screen; the actual data is stored in the 32 memory locations before this address.) You can use Program 1 without Program 2. Program 1 isn't a long program, but it does use most of the available memory in the unexpanded VIC. For this reason, REM statements are not included. When typing in this program, do not use unnecessary spaces.

When you finish designing with Program 1, type NEW (be sure to save a copy first), and enter Program 2. This second program allows you to examine memory to see the decimal values for the data created by Program 1, and to print the values (to screen and graphics printer) along with your created character.

Program 2 also allows you to print a reversed image of your created character and the corresponding decimal PEEK values.

Here's an explanation of the programs.

Program 1	Lines
70–80	Reset top of memory pointers; copy 32 characters into protected RAM; DIMension array to recall marked grid for revision; set variables for characters and display; leave a clean slate to draw on.
90	Create string to draw grid (SHIFTed @ key) and strings for positioning characters, inputs, etc., throughout program.
100–105	Array from which bit values are read.
130–150	Draw grid and set up display.
160–290	Keyboard controls.
300–320	Read grid, store values, display design.
330–350	Offer options and, with 580–620, show figure in chosen color moving around screen.
470	Print figure and values. The last number printed is the next address to be POKEd. If this is 7672, all available characters have been programmed.
480–540	Offer options.
630–690	Redraw marked grid for revision after display in color and motion.

Program 2	Lines
20–40	Set up display of design; print DATA values; show first and last address PEEKed.
45	Get a name for the design being displayed.
50–95	Set up display of character reversed horizontally.
115	Increment CHR$ for next design.
125–150	PEEK registers; reverse values; list them in proper order for program use.
155–175	Restructure the data for use by the screen dump routine.
180–235	Print a copy of the character design on a VIC 1515 or 1525 graphics printer.

(Program 2 is also handy for checking out the contents of other locations. Just change the value of PC in line 20 from 7448 to whatever address interests you.)

Program 1. Character Creator

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

```
10 PRINT"{CLR}"SPC(5)"{DOWN}{RED}PROGRAMMABLE":PRI
   NTSPC(4)"CHARACTER GRID"                   :rem 68
20 PRINT"{2 DOWN}{BLK}BEGIN AT TOP LEFT.";:rem 102
30 PRINT"THECURSOR{2 SPACES}CONTROLS{2 SPACES}AND
   {SPACE}SPACE BAR WILL BEHAVE NORMALLY. APLHANUM
   ERICKEYS ";                               :rem 234
40 PRINT"MARK THE GRID.":PRINT"{2 DOWN}HIT F1 TO S
   EE DESIGNEDCHARACTER.                     :rem 122
50 PRINT"{GRN}{2 DOWN}{RIGHT}HIT RETURN TO BEGIN
                                              :rem 62
60 GETM$:IFM$<>CHR$(13)THEN60                :rem 182
70 POKE52,29:POKE56,29:CLR:FORI=7424TO7679:POKEI,P
   EEK(I+25600):NEXT:DIMCT%(255):J=35         :rem 54
80 PC=7448:FORI=7448TO7679:POKEI,0:NEXT      :rem 148
90 G$="{RVS}@@@@@@@@@@@@@@@@":P$="{HOME}{19 DOWN}"
   :P1$="{22 RIGHT}"                         :rem 106
100 A$(0)="128":A$(1)="64":A$(2)="32":A$(3)="16"
                                             :rem 178
105 A$(4)="8":A$(5)="4":A$(6)="2":A$(7)="1"
                                             :rem 197
110 PRINT"{CLR}":POKE36869,255:FORG=1TO16:PRINT"
    {CYN}"G$:NEXT                            :rem 165
120 PRINT"{HOME}{UP}{RVS}{BLK}8765432187654321":FO
    RX=1TO8:PRINT"{RVS}"LEFT$(P$,1+X)LEFT$(P1$,16)
    X:NEXT                                   :rem 110
130 FORX=1TO8:PRINT"{RVS}"LEFT$(P$,9+X)LEFT$(P1$,1
    6)X:NEXT                                 :rem 222
140 CR$="{2 SPACES}{DOWN}{2 LEFT}"+CHR$(J)+CHR$(J+
    1)+"{DOWN}{2 LEFT}"+CHR$(J+2)+CHR$(J+3)+"
    {DOWN}{2 LEFT}{2 SPACES}"                :rem 193
150 PRINT"{OFF}{HOME}{2 DOWN}"LEFT$(P1$,18)CR$LEFT
    $(P$,20)"{RVS}{GRN}HIT F1 TO SEE DESIGN.{HOME}
    "                                        :rem 174
160 GETM$:IFM$=""THEN160                     :rem 105
170 IFPEEK(211)=16ANDPEEK(210)=31ANDPEEK(209)>90TH
    ENPRINT"{HOME}{DOWN}";                    :rem 60
180 IFPEEK(210)=31ANDPEEK(209)>100THENPRINT"{HOME}
    {DOWN}";                                 :rem 228
185 IF PEEK(210)=30ANDPEEK(209)=0THENPRINT"{HOME}"
                                              :rem 58
190 IFPEEK(211)=16THENPRINT:GOTO160          :rem 157
200 IFM$<>CHR$(20)ANDM$<>CHR$(148)THEN210   :rem 47
210 IFM$=CHR$(13)THENPRINTCHR$(13);:GOTO160:rem 61
220 IFM$=CHR$(17)THENPRINTCHR$(17);:GOTO160:rem 70
230 IFM$=CHR$(29)THENPRINTCHR$(29);:GOTO160:rem 77
```

```
240 IFM$=CHR$(145)THENPRINTCHR$(145);:GOTO160
                                          :rem 172
250 IFM$=CHR$(157)ANDPOS(M$)<>0THENPRINTCHR$(157);
    :GOTO160                              :rem 228
260 IFM$=CHR$(32)THENPRINT"{RVS}{BLK}"CHR$(186);:G
    OTO160                                :rem 100
270 IFM$=CHR$(133)THENPRINTCHR$(133):GOTO300
                                          :rem 106
280 PRINT"{RVS}{BLU}"CHR$(166);           :rem 198
290 GOTO160                               :rem 107
300 B=0:L=7702:FORY=1TO2:FORZ=LTOL+154STEP22:D=0:C
    =0:GOSUB550:PC=PC+1:NEXT:L=L+8:NEXT   :rem 112
310 L=7878:FORY=1TO2:FORZ=LTOL+154STEP22:D=0:C=0:G
    OSUB550:PC=PC+1:NEXT:L=L+8:NEXT       :rem 150
320 FORSC=8076TO8186:POKESC,32:NEXT       :rem 167
330 PRINTLEFT$(P$,19)"{RVS}LIKE IT? ";:INPUT"{RVS}
    Y OR N";N$                            :rem 116
340 IFN$="Y"THEN360                        :rem 58
350 IFN$="N"THENPRINTLEFT$(P$,19)"{RVS}CURSOR IS A
    T TOP LEFT":PC=PC-32:PRINT"{HOME}":GOTO160
                                          :rem 168
360 PRINT"{CLR}":INPUT"{RVS}COLOR - 1TO8";E:rem 85
370 ONEGOTO380,390,400,410,420,430,440,450:rem 167
380 PRINT"{BLK}":GOSUB580:GOTO460          :rem 96
390 POKE36879,110:PRINT"{WHT}":GOSUB580:PRINT"
    {BLK}":GOTO460                        :rem 169
400 PRINT"{RED}":GOSUB580:GOTO460         :rem 229
410 PRINT"{CYN}":GOSUB580:GOTO460         :rem 105
420 PRINT"{PUR}":GOSUB580:GOTO460         :rem 103
430 PRINT"{GRN}":GOSUB580:GOTO460         :rem 234
440 PRINT"{BLU}":GOSUB580:GOTO460         :rem 236
450 PRINT"{YEL}":GOSUB580:GOTO460         :rem 108
460 POKE36879,27:PRINT"{CLR}{DOWN}{8 RIGHT}"CR$:PR
    INTLEFT$(P$,8);                        :rem 57
470 PC=PC-32:FORCH=1TO4:FORX=PCTOPC+7:PRINT"{RVS}
    {BLK}"PEEK(X);:PC=PC+1:NEXT:PRINT:NEXT:rem 202
480 PRINT"{RVS}"PC:INPUT"{RVS}WANT TO SEE IT AGAIN
    ";N$                                  :rem 150
490 IFN$="Y"THEN360                        :rem 64
500 INPUT"{RVS}REVISE IT";Q$              :rem 150
510 IFQ$="Y"THEN640                        :rem 61
520 INPUT"{RVS}DRAW ANOTHER";M$           :rem 104
530 IFM$="Y"THENJ=J+4:GOTO90              :rem 173
540 END                                   :rem 112
550 FORX=0TO7:CT%(B)=PEEK(Z+X):IFPEEK(Z+X)=230THEN
    C=VAL(A$(X))                           :rem 24
560 IFPEEK(Z+X)=250THENC=0                 :rem 48
570 D=D+C:POKEPC,D:B=B+1:NEXT:RETURN       :rem 37
580 PRINT"{CLR}":FORX=1TO18:PRINTLEFT$(P$,X)" "CR$
    :FORT=1TO75:NEXT:NEXT                  :rem 52
```

```
590 FORX=1TO18:PRINTLEFT$(P$,19)LEFT$(P1$,X)"
    {DOWN}{LEFT} {DOWN}{LEFT} {2 UP}"CR$:FORT=1TO7
    5:NEXT:NEXT                               :rem 240
600 FORX=18TO1STEP-1:PRINTLEFT$(P$,X)LEFT$(P1$,18)
    CR$:FORT=1TO75:NEXT:NEXT                  :rem 191
610 FORX=18TO1STEP-1:PRINTLEFT$(P$,1)LEFT$(P1$,X)C
    R$"{2 UP} {DOWN}{LEFT} ":FORT=1TO75:NEXT:NEXT
                                              :rem 156
620 RETURN                                    :rem 120
630 B=0:PRINT"{CLR}"                          :rem 230
640 B=0:PRINT"{CLR}"                          :rem 231
650 FORQ=1TO8:FORX=1TO8:PRINT"{RVS}{BLU}"CHR$(CT%(
    B));:B=B+1:NEXT:PRINT:NEXT:PRINT"{HOME}{DOWN}
    {8 RIGHT}";                                :rem 88
660 FORQ=1TO8:FORX=1TO8:PRINT"{RVS}"CHR$(CT%(B));:
    B=B+1:NEXT:PRINT"{DOWN}{8 LEFT}";:NEXT:PRINT
                                               :rem 39
670 PRINT"{UP}";:FORQ=1TO8:FORX=1TO8:PRINT"{RVS}"C
    HR$(CT%(B));:B=B+1:NEXT:PRINT:NEXT        :rem 192
680 PRINT"{8 UP}{8 RIGHT}";:FORQ=1TO8:FORX=1TO8
                                              :rem 151
690 PRINT"{RVS}"CHR$(CT%(B));:B=B+1:NEXT:PRINT"
    {DOWN}{8 LEFT}";:NEXT:PC=PC-32:GOTO120     :rem 27
```

Program 2. Character Printer and Screen Dump

```
10 PRINT"{CLR}"TAB(48)"{RED}PEEK AND PRINT :rem 28
15 FORT=1TO2500:NEXT:DIMC%(3,8,8)          :rem 186
20 PC=7448:J=35:POKE36869,255              :rem 115
25 CR$=CHR$(J)+CHR$(J+1)+"{DOWN}{2 LEFT}"+CHR$(J+2
   )+CHR$(J+3)                              :rem 31
30 PRINT"{CLR}{3 DOWN}{9 RIGHT}"CR$        :rem 184
35 PRINT"{3 DOWN}{RVS}"PC:FORA=1TO4:FORX=1TO8:PRIN
   T"{RVS}"PEEK(PC);:PC=PC+1:NEXT:PRINT:NEXT
                                           :rem 141
40 PRINT"{RVS}"PC-1                         :rem 56
45 PRINT"{DOWN}":INPUT"{RVS}DESIGN NAME";DN$:GOSUB
   155:GOSUB180                            :rem 223
50 PRINT"{2 DOWN}":INPUT"{RVS}REVERSE: Y OR N";AN$
                                             :rem 4
55 IFAN$="Y"THEN65                          :rem 32
60 GOTO100                                  :rem 48
65 RC$=CHR$(J+1)+CHR$(J)+"{DOWN}{2 LEFT}"+CHR$(J+3
   )+CHR$(J+2)                              :rem 35
70 PRINT"{CLR}{3 DOWN}{8 RIGHT}"CR$:PRINT"{3 DOWN}
   {RVS}"                                  :rem 239
75 L=7168:M=0:RC=PC-24:GOSUB125             :rem 67
80 RC=PC-32:GOSUB125                       :rem 177
85 RC=PC-8:GOSUB125                        :rem 137
90 RC=PC-16:GOSUB125                       :rem 180
```

```
95 RC=PC-32:FORM=ØTO32:POKERC+M,PEEK(L+M):NEXT
                                            :rem 157
1ØØ PRINT"{HOME}"TAB(242)TAB(22Ø);:INPUT"{RVS}DESI
    GN NAME";DN$:GOSUB155:GOSUB18Ø          :rem 192
1Ø5 PRINT"{HOME}"TAB(242)TAB(22Ø)" {RVS}HIT RETURN
     TO GO ON"                              :rem 231
11Ø GETA$:IFA$=""THEN11Ø                     :rem 71
115 J=J+4:GOTO25                            :rem 161
12Ø END                                     :rem 1Ø6
125 FORR=RCTORC+7:X=PEEK(R):B=Ø:C=Ø         :rem 167
13Ø FORA=7TOØSTEP-1:Y=INT(X/2↑A):IFY<=ØTHENZ=Ø:GOT
    O14Ø                                    :rem 2Ø2
135 Z=2↑C                                     :rem 3
14Ø B=B+Z:C=C+1:IFZ=ØTHEN15Ø                :rem 137
145 X=X-2↑A                                 :rem 133
15Ø NEXT:PRINT"{RVS}"B;:POKEL+M,B:M=M+1:NEXT:PRINT
    :RETURN                                 :rem 2Ø9
155 FORA=ØTO3:FORB=ØTO7:FORE=ØTO7:C%(A,B,E)=Ø:NEXT
    E,B,A:DC=PC-32                          :rem 153
16Ø FORA=ØTO3:B=Ø:FORD=DCTODC+6:X=PEEK(D):Y=Ø:FORE
    =7TOØSTEP-1:Y=INT(X/2↑E):IFY<ØTHENY=Ø    :rem 9Ø
165 IFY>ØTHENY=2                            :rem 233
17Ø C%(A,B,E)=Y↑B:X=X-Y↑E:NEXT:B=B+1:NEXT:DC=DC+8:
    NEXT                                    :rem 161
175 FORA=ØTO3:FORB=ØTO6:FORE=7TOØSTEP-1:C%(A,7,E)=
    C%(A,7,E)+C%(A,B,E):NEXTE,B,A:RETURN    :rem 194
18Ø REM SCREEN COPY                         :rem 12Ø
185 R$=CHR$(145):V$=CHR$(146):OPEN4,4:PRINT#4:G=PE
    EK(648)*256:PRINT#4,R$;:A=Ø             :rem 185
187 FORP=GTOG+5Ø5                            :rem 1Ø
19Ø C=PEEK(P):C$="":IF(P-G)/22=INT((P-G)/22)THENPR
    INT#4,CHR$(8)+CHR$(13)+CHR$(14);         :rem 91
195 IFA>3THEN215                            :rem 167
2ØØ IFC=32THEN23Ø                           :rem 2Ø2
2Ø5 FORE=7TOØSTEP-1:IFC%(A,7,E)=ØTHENC$=C$+CHR$(C%
    (A,7,E):NEXT                            :rem 125
21Ø C$=C$+CHR$(C%(A,7,E)+128):NEXT:PRINT#4,CHR$(8)
    C$;:A=A+1:NEXT                          :rem 1Ø8
215 IFC>128THENC=C-128                      :rem 253
22Ø IFC<32ORC>95THENC=C+64:GOTO23Ø           :rem 38
225 IFC>63ANDC<96THENC=C+128:C$=""          :rem 173
23Ø C$=C$+CHR$(C):IFLEN(C$)>1THENC$=C$+V$+R$
                                            :rem 175
235 PRINT#4,C$;:NEXT:PRINT#4:CLOSE4:RETURN:rem 225
```

Spiralizer

Chayim Avinor
Translation by Patrick Parrish

Based on geometrical principles, "Spiralizer" lets you create some dazzling patterns; it is possible to overlay as many as five spiral figures on the high-resolution graphics screen. Requires the Super Expander cartridge.

"Spiralizer" is a program for making patterns on the high-resolution screen of the VIC. It makes patterns very much like those made on the noncomputer game called Spirograph, which has tooth wheels of different sizes. However, Spiralizer can create a far greater number of combinations.

The patterns are actually made by two radii—one of them is turning around a stationary or linearly moving center (depending on your input), and the center of the other radius is the free edge of the first one.

You are given control of the relative speed and length of the radii and some additional handy features.

Running the Program

After typing RUN and RETURN, you are asked to type in the number of figures. Use the DEL/INST key to delete a character. If you simply press RETURN without typing in a number, the program will default to a value of 1.

Next you will be asked to input the spiral speed. This is the actual number of times the program will loop through. If you enter 10 and RETURN, your pattern will have ten complete loops. Large numbers cause the program to draw straight segments because of the large steps. The patterns thus produced are quite nice, too. Speed defaults to 5 if you just press RETURN.

A pattern with three loops is easily understood, but what would a pattern with two loops look like? How about one loop? Could a pattern possibly have zero loops? Try them and see.

The second number you are asked to enter is the radius. This determines the ratio of the radii. You can choose any number between 1 and 60. A small number would make the inner radius small and the outer radius large, and vice versa. Like the first input, you can simply press RETURN and take the default value of 35.

Next you are asked for the spin. An answer larger than 1 will make the pattern rotate while it is drawn, and, of course, the number of loops will change. You can choose between 1 (no spin) and 18. When spinning, the lines remain smooth and curvy, but it takes more time to draw the complete pattern. If you decide to quit while a pattern is being drawn, press any key and the program will return to the menu. To escape from the program, hold down the STOP key and press RESTORE.

Added Features

Now things become more complicated. You are asked MOVEMENT OR DECREMENT (M/D)? If you choose M, the whole pattern will move while it is being drawn. If the spin is 1, the pattern will be drawn five times while it moves. If the spin is greater than 1, the pattern will move until it finishes rotating. If the spin is greater than 1 but less than 9, you will not be asked for this input.

Pressing D will cause the pattern to decrease in size while being drawn. The rules here are the same as above. If you press RETURN, the default value is NONE, and none of the above actions will take place.

Once the display is finished, press any key for the menu.

For a nice sample, I suggest you try the following INPUTs: For speed, enter 7; radius, 50; for spin, 18; then choose M for movement and clear the screen.

Experiment with different values, and you'll see some stunning designs.

Spiralizer

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

```
100 IF FL=1 THEN:GRAPHIC 4                     :rem 187
110 FL=1:PRINT"{CLR}":POKE 36879,28:PRINT"{DOWN}
    {3 RIGHT}{YEL}U**************I":PRINT"
    {3 RIGHT}-{14 SPACES}-"                     :rem 139
120 PRINT"{3 RIGHT}-{PUR}{2 SPACES}SPIRALIZER
    {2 SPACES}{YEL}-":PRINT"{3 RIGHT}-{14 SPACES}-
    "                                           :rem 208
130 PRINT"{3 RIGHT}J**************K{BLU}"  :rem 240
140 POKE198,0:Z=1:PRINT"{2 DOWN}{RED}HOW MANY SPIR
    AL":PRINT"FIGURES (1-5) ?{4 SPACES}{3 LEFT}";:
    GOSUB550:F=Z                                :rem 187
```

```
150 IF F<1 OR F>5 THEN PRINT"{4 UP}";:GOTO 140
                                            :rem 185
160 FORP=1TOF:PRINT"{DOWN}{GRN}SPIRAL FIGURE #";P:
    GOSUB 690                               :rem 243
170 FOR I=7TO0 STEP-1:FORJ=1TO50:NEXT J:POKE 38680
    ,I:NEXT I                               :rem 246
180 Z=5:PRINT"{OFF}{BLU}{DOWN}SPEED (-50,50) ?
    {4 SPACES}{3 LEFT}";:GOSUB 550:K(P)=Z :rem 121
190 IF Z<-50 OR Z>50 THEN PRINT"{2 UP}";:GOTO 180
                                             :rem 88
200 K(P)=K(P)-1                               :rem 5
210 Z=35:PRINT"{DOWN}RADIUS (1,60) ?{4 SPACES}
    {3 LEFT}";:GOSUB 550:R(P)=Z*7            :rem 84
220 IF Z<1 OR Z>60 THEN PRINT"{2 UP}";:GOTO 210
                                            :rem 236
230 S=1                                      :rem 86
240 Z=1:PRINT"{DOWN}SPIN (1,18) ?{4 SPACES}
    {3 LEFT}";:GOSUB 550                    :rem 112
250 IF Z<1 OR Z>18 THEN PRINT"{2 UP}";:GOTO 240
                                            :rem 245
260 FOR I=8120 TO 8163:POKE I,32:NEXT I      :rem 74
270 A(P)=1/Z:IF Z>1 AND Z<9 THEN PRINT"{2 DOWN}":G
    OTO 350                                 :rem 211
280 SM(P)=1:M(P)=2:PRINT"{DOWN}MOVEMENT / DECREMEN
    T{2 SPACES}(M/D) ?{5 SPACES}{4 LEFT}"; :rem 44
290 GET X$:IF X$="" THEN 290                :rem 135
300 IF X$=CHR$(13) THEN M(P)=0:SM(P)=0:PRINT"NONE"
    :GOTO 350                                :rem 73
310 IF X$="M" THEN SM(P)=0:GOTO 340          :rem 84
320 IF X$="D" THEN M(P)=0:GOTO 340          :rem 249
330 GOTO 290                                :rem 106
340 PRINT X$                                :rem 160
350 PRINT"{12 UP}":NEXT P:C5=INT(RND(0)*6)+2
                                             :rem 75
360 FOR P=1 TO F                             :rem 39
370 W=1:Z=400:IF M(P)=2 THEN Z=401:IF A(P)=1 THEN
    {SPACE}W=5:M(P)=1:Z=370                  :rem 87
380 IF SM(P)-A(P)=0 THEN W=5                :rem 223
390 IF A(P)<1 THEN K(P)=K(P)+A(P)           :rem 203
400 C=.001:IF A(P)<1/9 THEN M(P)=M(P)/2:C=C/2
                                             :rem 90
410 J=R(P):I=400-R(P)                       :rem 131
420 GRAPHIC 2:COLOR 1,2,C5,C5                :rem 86
430 POINT C5,125+(Z+100)/1.3,130            :rem 112
440 FOR T=0TO 6.2831/A(P)*W STEP .06283     :rem 20
450 IF PEEK(198)<> 0 THEN 100               :rem 170
460 IF SM(P) THEN J=R(P)*S:I=400*S-J:S=S-C:rem 101
470 X=125+(100+Z+T*M(P)-SIN(T)*J+SIN(T*K(P))*I)/1.
    3                                        :rem 37
```

```
480 Y=520-COS(T)*J-COS(T*K(P))*I                    :rem 52
490 IF X<0 OR Y<0 THEN 650                          :rem 32
500 DRAW 1 TO X,Y:NEXT                             :rem 237
510 NEXT P:GOSUB 690                               :rem 126
520 GET R$:IF R$="" THEN 520                       :rem 115
530 GOTO 100                                        :rem 98
540 REM -INPUT ROUTINE-                            :rem 141
550 L0=0:L1=1:B$=""                                :rem 201
560 GET A$:IF A$="" THEN 560                        :rem 89
570 IF A$="-" AND L0=0 THEN PRINT A$;:B$=A$:L0=1:L
    1=2:GOTO 560                                   :rem 240
580 IF A$=CHR$(13) AND L0>0 THEN Z=VAL(B$):PRINT:R
    ETURN                                          :rem 130
590 IF A$=CHR$(13) THEN PRINT Z:RETURN             :rem 181
600 IF A$=CHR$(20) AND L0>1 THEN PRINT A$;:B$=LEFT
    $(B$,LEN(B$)-1):L0=L0-1:GOTO 560               :rem 115
610 IF A$=CHR$(20) AND L0=1 THEN PRINT A$;:B$="":L
    0=0:GOTO 560                                   :rem 231
620 IF L0>L1 THEN 560                               :rem 40
630 IF A$<"0" OR A$>"9" THEN 560                   :rem 200
640 PRINT A$;:B$=B$+A$:L0=L0+1:GOTO 560            :rem 115
650 REM ERROR TRAPPING ROUTINE                     :rem 148
660 GRAPHIC 4:PRINT"{CLR}{2 DOWN}COORDINATES ARE O
    UT{3 SPACES}OF RANGE"                          :rem 155
670 PRINT"{2 DOWN}TRY ANOTHER FIGURE":PRINT"
    {2 DOWN}RETURNING TO MAIN MENU"                 :rem 74
680 FOR I=1 TO 4000:NEXT:GOTO 110                   :rem 33
690 FORV=15 TO 0 STEP -.5:SOUND245,0,0,0,V:NEXT:RE
    TURN                                           :rem 165
```

Art Museum

Floyd Beaston

Your VIC has graphics characters right on the keys. This program lets you take advantage of these graphics by allowing you to save and load screen pictures made using character graphics.

My eight-year-old son loves to draw artwork on the screen, using combinations of the graphics symbols on the keys. Because the artworks vanished forever when we turned off the computer, my son became more and more frustrated.

This program was written to help with this problem by allowing you to save and load all characters, including graphics symbols, on the screen.

To use this program, first remove any expansion board and then type the program. Then enter this line:

CLR:POKE46,PEEK(46)+4

and SAVE the program to disk or tape.

Using the Art Museum

If you wish to draw a picture (to later save), follow these steps:

1. LOAD the program from disk or tape.
2. Change line 1 to—1 REM.
3. Clear the screen.
4. Draw your picture using the keyboard characters.
5. Once the picture is complete, move the cursor to a clear line (no characters on the line).
6. Change the cursor color to match the background color (probably CTRL-2).
7. Type RUN and press RETURN (remember you will not be able to see what you are typing, so be careful).
8. After 30 seconds, turn the cursor to a color you can see (try CTRL-1).
9. SAVE the program.

To retrieve your picture, LOAD the version of the program you saved after you created your picture. Change line 1 to:

1 GOTO20

This will magically return your picture to the screen.

Art Museum

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

```
0 S=7680:C=38400:GOSUB63999                          :rem 28
1 GOTO20                                             :rem 204
10 FORJ=0TO505:POKEML+J,PEEK(S+J):POKEML+506+J,PEE
   K(C+J):NEXT:END                                   :rem 62
20 FORJ=0TO505:POKES+J,PEEK(ML+J):POKEC+J,PEEK(ML+
   506+J):NEXT:PRINT"{HOME}";                        :rem 135
21 GOTO 21                                           :rem 255
63999 ML=PEEK(61)+PEEK(62)*256+31:RETURN             :rem 64
```

Graph Plotter

Ruth A. Hicks
Translation by Jeff Hamdani

Not only is "Graph Plotter" an interesting tool for drawing 3-D columnar charts, but the accompanying article takes you step by step through the program itself so you can learn how it was written. For the unexpanded VIC.

"Graph Plotter" is a good demonstration of what beginning programmers can accomplish in the way of graphics on the VIC-20. Different graphics techniques were used to create this program. By reading this article and following along with the program listing, you can increase your knowledge of graphics formatting. Of course, if you're not into learning programming, there's no reason why you can't just type in the program listing anyway.

Graph Plotter creates attractive bar graphs with three-dimensional columns. The graphs are particularly exciting in color. There are six columns, each a different color, to which you assign a value from 0 to 15 for the column height. You tell the computer what values each column has, and then you can interpret their meaning.

Modular Programming

Graph Plotter was written with a technique known as *modular* or *block programming*. This means a section at a time was written on the computer and then checked for eye appeal, function, and (of course) that familiar message, ?SYNTAX ERROR. There are five main blocks to this program.

When you're typing the program, omit unnecessary spaces except in any INPUT or PRINT statements between quotation marks. Other spaces are not needed by the computer and only consume more memory. Since this is an article to learn from, let's start some good habits right away by not typing those useless spaces. Also remember it is not necessary to enter the rems at the end of each line. These rems are checksums for "The Automatic Proofreader" (see Appendix J).

Block One

Block one (lines 100–180) creates the graph, including the segments and the outlining border. Instead of using line after line of PRINT statements, we'll be POKEing the information directly into memory inside FOR-NEXT loops.

Line 100 clears the screen and sets the background color to black and the border blue. Line 110 starts the top border at screen memory location 7726 and runs it across the screen to location 7745, drawing a continuous line. Refer to Appendices C and D for screen locations and colors. Each time the FOR-NEXT loop is executed, it places the new value of I into the POKE statement with the symbol number 114 (refer to Appendix H, Screen Codes). The I value tells the computer *where* to put the symbol, and the 114 tells *what* symbol to put in that spot.

The second POKE in line 110 colors the symbol green. Since the color memory (see Appendix D) corresponds to the screen memory, only with a different set of numbers, all we have to do is calculate the offset. The difference between 38400 and 7680 (the starting address of color and screen memory in VIC-20) is 30720, a simple subtraction problem. So we POKE I+30720 with the color code for green (5) and presto, we have a green symbol at the correct location.

Line 120 draws the left border, beginning at screen memory location 7748 and ending at location 8034. The STEP 22 is used because a VIC-20 has 22 characters per line across its screen. If you look at the Screen Location Table (Appendix C) and find screen location 7748, then add 22, you'll find that location 7770 is exactly one line below 7748. The second POKE in line 120 places the color code for green (5) in the color memory for the screen location, as in line 110. Lines 130 and 140 are similiar to lines 110 and 120; line 130 draws the right border, and line 140 draws the bottom border.

The last four lines (150–180) in this section were constructed in the same manner, using FOR-NEXT loops to POKE information directly into screen memory. These lines draw continuous lines on the graph, making it more readable.

Designations

Block two of the program prints a series of numbers on the left side of the graph and letter designations for each of the six columns. Lines 190–220 position the following PRINT statement at the right spot horizontally so the numbers can be displayed along the left side of the graph. We want the two-digit numbers (10–15) to start at the border, so we place a SPC(0) after the

PRINT, and then place the number to be printed inside quotation marks. The one-digit numbers we want to be in the second position, so we place a SPC(1) before the number. This makes for a nice display.

So lines 190–220 label the Y-axis with a sequence of numbers from 15 to 0. Notice that between colons is a complete PRINT statement, and even though they are all crunched together in only four program lines with *no spaces,* they result in 16 lines of vertical display.

The last line of this section (230) puts letter designations along the bottom of the graph beneath the columns. Notice there is only one PRINT since this line is displayed horizontally. The first letter is positioned with TAB(4), and the following letters are all equally spread with SPC(2) statements.

READ-DATA Block

In the third block of the program (lines 240 to 300), DATA is READ that will be used in a later routine to position each vertical bar on the graph and decide its color. Line 240 prevents this DATA from being reREAD unnecessarily with any subsequent passes through the program.

The first statement that READs DATA in this section is in line 260. Here, a READ command is contained in a FOR-NEXT loop, so it is executed six times. This causes six strings, representing the six column labels (A, B, C, D, E, F), to be READ and set equal to the string array variable A$(I).

In line 280, a second set of DATA is READ and assigned to D(I). This string array variable denotes the color code for each vertical bar on the graph.

The last group of DATA in this block is READ from line 300. The values taken from line 290 are the screen memory addresses necessary to properly locate each bar on the graph.

The use of arrays in this section significantly shortens the length of the program. Instead of requiring six separate blocks of code to locate and draw each vertical bar, we will now be able to perform this in one routine.

Input Block

The fourth block of the program (lines 310–420) is the INPUT routine. Notice that much of this routine is contained within a FOR-NEXT loop (lines 310–370).

In this loop, you are asked what value you want for each column. The value that you INPUT determines the height of each vertical bar. Your response is checked in line 360 to make sure it is within the limits of 0 and 15.

After the height of each column is INPUT, the screen memory address (A) for the top of the column is determined by the first statement in line 370.

Here's how it works: A(J) is set as a starting screen location in the first line, then AA (the response) is multiplied by 22, because our screen is 22 characters across. Then AA times 22 is subtracted from A(J), because the columns are drawn upward. So, if the response is 10, the column rises 10 segments high. Then 44 is added to A to bring it down two rows so we have room for our three-dimensional side. Program execution is then transferred to the subroutine at line 430, which actually draws each column on the graph.

In the process, the variables necessary to this subroutine are passed. The variable C defined in 310 is the offset between the screen memory map and the color memory map as explained above. The actual color of each column (variable D) and the starting screen location of each column (variable X) are also transferred.

Once a column has been drawn, the user's previous INPUT is erased in line 320 by POKEing blank spaces into this area of screen memory. If you didn't do this, the prior answer, of course, would remain on the screen.

Line 320 enables you to position a PRINT statement exactly where you want vertically without disrupting any printing already on the screen. The cursor is first homed, and then a blank PRINT statement is placed inside a FOR loop. As the loop is executed, starting at the HOME position, it counts down vertically to the maximum number set by the FOR loop.

The next line is the INPUT statement, now in the right position to be printed. At the end of the INPUT statement is the variable AA, which receives whatever value you enter between the limits of 0 and 15. If the response is less than 0 or greater than 15, the computer erases the answer and asks the same question again.

Once all six vertical bars have been drawn, you will be asked in line 390 if you wish to do another bar graph. If you do, the program will start again at line 100. Otherwise, it will END in line 420.

The Subroutine

The heart of this program is the subroutine beginning at line 430. This is the block which draws the columns by POKEing symbols onto the screen.

Let's start explaining this section with lines 430 and 440. These two lines check to see if the value AA from the INPUT block is a 1 or 0. If AA=1, the program branches to line 530, which draws the top of a column one segment high on the graph. When AA=0, it is a null entry, and the program gets another INPUT.

Lines 470 and 480 begin to actually draw the columns, which are three characters wide. Reflecting back to the INPUT block, you'll recall that variables A and X were set for the starting point and top part of the column. So, by POKEing the screen memory locations with the desired character symbols in a FOR loop, we can draw the columns to any height we've chosen. Notice there are three POKEs, I, I+1, I+2. Each addition to I moves its location over one spot to the right, yielding a three-character-wide column. The different screen display codes create a three-dimensional appearance with reversed characters. The program reverses the character codes by adding 128 to the symbol code (32+128=160, 101+128=229). Line 480 follows up line 470 with the color information by adding the color variables C and D to the same locations from line 470.

Lines 510–540 follow the same format as lines 470 and 480. They draw the three-dimensional top segments of the columns. Six character symbols and six color locations are POKEd into the appropriate locations with the variables A, C, and D. By adding or subtracting numbers from A, we can position the symbols on the row above or to the right.

Once this subroutine is completed, line 550 RETURNs to the INPUT block.

Formatting

By now, you should have Graph Plotter typed in and saved on tape or disk. The difference between this program and others you have typed is that you now know exactly how it was programmed. Remember the techniques of using PRINT statements for displaying characters vertically and horizontally; of blank PRINT statements and SPC() commands for positioning INPUT or PRINT statements exactly where you want them; of directly placing symbols and colors onto the screen with POKEs and variables. In planning your own programs, use these techniques for your screen displays and see how handy and timesaving they can be for you.

Graph Plotter

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

```
100 PRINT"{CLR}":POKE36879,11                    :rem 253
110 FORI=7726TO7745:POKEI,114:POKEI+30720,5:NEXTI
                                                 :rem 191
120 FORI=7748TO8034STEP22:POKEI,107:POKEI+30720,5:
    NEXTI                                        :rem 94
130 FORI=7767TO8053STEP22:POKEI,115:POKEI+30720,5:
    NEXTI                                        :rem 96
140 FORI=8056TO8075:POKEI,113:POKEI+30720,5:NEXTI
                                                 :rem 187
150 FORI=7793TO7810:POKEI,64:POKEI+30720,5:NEXTI
                                                 :rem 148
160 FORI=7859TO7876:POKEI,64:POKEI+30720,5:NEXTI
                                                 :rem 164
170 FORI=7925TO7942:POKEI,64:POKEI+30720,5:NEXTI
                                                 :rem 153
180 FORI=7991TO8008:POKEI,64:POKEI+30720,5:NEXTI:P
    RINT                                         :rem 94
190 PRINTSPC(0)"15":PRINTSPC(0)"14":PRINTSPC(0)"13
    ":PRINTSPC(0)"12":PRINTSPC(0)"11"            :rem 142
200 PRINTSPC(0)"10":PRINTSPC(1)"9":PRINTSPC(1)"8":
    PRINTSPC(1)"7"                               :rem 44
210 PRINTSPC(1)"6":PRINTSPC(1)"5"                :rem 170
220 PRINTSPC(1)"4":PRINTSPC(1)"3":PRINTSPC(1)"2":P
    RINTSPC(1)"1":PRINTSPC(1)"0"                 :rem 147
230 PRINTTAB(4)"A"SPC(2)"B"SPC(2)"C"SPC(2)"D"SPC(2
    )"E"SPC(2)"F"                                :rem 184
240 IFZ$="Y"THEN310                              :rem 64
250 DATAA,B,C,D,E,F                              :rem 34
260 FORI=1TO6:READA$(I):NEXTI                    :rem 38
270 DATA7,6,4,3,5,1                              :rem 201
280 FORI=1TO6:READD(I):NEXTI                     :rem 7
290 DATA8035,8038,8041,8044,8047,8050            :rem 113
300 FORI=1TO6:READA(I):NEXTI                     :rem 253
310 C=30720:FORJ=1TO6                            :rem 194
320 FORK=8138TO8141:POKEK,32:NEXTK:PRINT"{HOME}
    {19 DOWN}"                                   :rem 179
330 PRINT"COLUMN ";A$(J);" (0-15) ";             :rem 62
340 INPUT Y$:IFVAL(Y$)=0THENNEXTJ                :rem 67
350 Y=VAL(Y$):Y=INT(Y+.5):D=D(J):X=A(J)          :rem 23
360 IFY<0ORY>15THEN320                           :rem 79
```

```
370 A=A(J)-(Y*22)+44:GOSUB430:NEXTJ             :rem 0
380 PRINT"{HOME}{20 DOWN}"                       :rem 211
390 PRINT"DO IT AGAIN? (Y/N)"                    :rem 99
400 GETZ$:IFZ$=""THEN400                         :rem 125
410 IFZ$="Y"THEN100                              :rem 60
420 END                                          :rem 109
430 IFY=1THEN530                                 :rem 180
440 IFY=0THENRETURN                              :rem 252
450 POKEX,160:POKEX+1,231:POKEX+2,105            :rem 162
460 POKEX+C,D:POKE(X+1)+C,D:POKE(X+2)+C,D:IFY=2THE
    N490                                         :rem 245
470 FORI=X-22TOASTEP-22:POKEI,160:POKEI+1,231:POKE
    I+2,160                                      :rem 185
480 POKEI+C,D:POKE(I+1)+C,D:POKE(I+2)+C,D:NEXTI:GO
    TO510                                        :rem 56
490 POKEA,227:POKEA+1,208:POKEA+2,105            :rem 105
500 POKEA+C,D:POKE(A+1)+C,D:POKE(A+2)+C,D:GOTO530
                                                 :rem 89
510 POKEA,227:POKEA+1,208:POKEA+2,224            :rem 100
520 POKEA+C,D:POKE(A+1)+C,D:POKE(A+2)+C,D        :rem 80
530 POKEA-22,233:POKEA-21,160:POKEA-20,206       :rem 87
540 POKE(A-22)+C,D:POKE(A-21)+C,D:POKE(A-20)+C,D
                                                 :rem 154
550 RETURN                                       :rem 122
```

Chapter 5

Music and Sound

Musical Scales

Brian H. Lawler

"Scales" is a short, 2K RAM educational program which exploits the sound-generating capabilities of the VIC-20 microcomputer. The program allows you to choose one of nine musical scales in the key of your choice. The computer then plays the scale up and down and assigns eight notes of the scale, in ascending order, to keys 1 through 8 on the VIC keyboard. You may then play any note on the scale by pressing one of these keys.

You will soon be able to play simple tunes on the scales by ear, even if you can't read a note of music. Besides being fun, this exercise will give you some understanding of the scales used in different types of music. You will be able to recognize which scale is commonly used in jazz and which scale has an oriental sound. Get together with three of your computer friends and start a VIC quartet, or be the first composer on your block to write a symphony for cello and VIC.

Program Notes Lines	
5	Dimension an array, N, to store 38 variables, N(x).
10	Initialize variables S1–S4 as the four VIC voice locations and variable VO as volume.
20	Read data into variable N(x). These are all the notes that the VIC can play. The values are from page 135 of the *User's Guide*.
30	Read data into the variables D$(x) that are used in making the scales. A 1 raises the next note 1/2 step, a 2 raises the next note a whole step, etc.
155–160	Get the scale number.
175–240	Input the key and set variable S as the pointer to the first note of the scale.
250–290	Put the notes of the scale into Q(1) to Q(8) by using the data strings established in line 30.
300–390	Play the selected scale up and down once.
420–450	Get your note and POKE it into S2.
460	Wait for you to release the key.
470	Turn off the sound and go back to line 420 to wait for another note.

Scales

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

```
5 DIMN(37)                                           :rem 24
10 VO=36878:S1=36874:S2=S1+1:S3=S2+1:S4=S3+1
                                                    :rem 244
20 FORI=1TO37:READN(I):NEXT                         :rem 196
25 POKE36879,94:POKE646,0                           :rem 165
30 FORI=1TO9:READD$(I):NEXT                         :rem 174
100 PRINT"{CLR}{7 SPACES}{RVS}SCALES"               :rem 194
112 PRINT"{2 DOWN}THIS PROGRAM ALLOWS{3 SPACES}YOU
     TO SELECT A MU-";                               :rem 84
114 PRINT"{3 SPACES}SICAL SCALE IN ANY{4 SPACES}KE
    Y."                                             :rem 209
115 PRINT"THE COMPUTER THEN AS- SIGNS THE NOTE VAL
    UES"                                             :rem 19
116 PRINT"TO KEYS 1 TO 8 ON THE VIC KEYBOARD."
                                                     :rem 51
117 PRINT"{3 DOWN}HIT ANY KEY-"                     :rem 128
118 GOSUB890                                        :rem 187
120 PRINT"{CLR}{2 DOWN}{RVS}1{OFF} MAJOR"           :rem 103
122 PRINT"{DOWN}{RVS}2{OFF} MINOR"                  :rem 210
124 PRINT"{DOWN}{RVS}3{OFF} HARMONIC MINOR":rem 38
126 PRINT"{DOWN}{RVS}4{OFF} PENTATONIC"              :rem 72
128 PRINT"{DOWN}{RVS}5{OFF} WHOLE TONE"              :rem 11
130 PRINT"{DOWN}{RVS}6{OFF} DORIAN"                  :rem 13
132 PRINT"{DOWN}{RVS}7{OFF} PHRYGIAN"               :rem 181
134 PRINT"{DOWN}{RVS}8{OFF} LYDIAN"                  :rem 23
136 PRINT"{DOWN}{RVS}9{OFF} MIXOLYDIAN"              :rem 87
150 PRINT"{2 DOWN}WHICH SCALE?"                     :rem 163
155 GOSUB890:IFA$<"1"ORA$>"9"THEN155                 :rem 38
160 SC=VAL(A$)                                        :rem 3
170 PRINT"{CLR}WHAT KEY?"                            :rem 88
171 PRINT"{DOWN}{7 SPACES}{RVS}ABCDEFG{OFF}"
                                                    :rem 251
172 PRINT"{DOWN}{3 SPACES}{RVS}#{OFF} SHARP
    {2 SPACES}{RVS}-{OFF} FLAT"                     :rem 185
173 PRINT                                            :rem 40
175 INPUT KY$                                       :rem 245
180 K$=LEFT$(KY$,1)                                  :rem 10
190 IFK$<"A"ORK$>"G"THENGOTO170                      :rem 50
200 IFK$="C"THENS=13                                :rem 119
202 IFK$="D"THENS=15                                :rem 124
204 IFK$="E"THENS=17                                :rem 129
206 IFK$="F"THENS=18                                :rem 133
208 IFK$="G"THENS=8                                  :rem 87
210 IFK$="A"THENS=10                                :rem 115
212 IFK$="B"THENS=12                                :rem 120
```

```
220 IFLEN(KY$)=1THEN25Ø                          :rem 79
225 K$=RIGHT$(KY$,1)                             :rem 93
23Ø IFK$="#"THENS=S+1:GOTO25Ø                    :rem 175
235 IFK$="-"THENS=S-1:GOTO25Ø                    :rem 192
24Ø GOTO17Ø                                      :rem 1Ø3
25Ø Q(1)=N(S)                                    :rem 153
26Ø FORI=2TO8                                    :rem 18
27Ø S=S+VAL(MID$(D$(SC),I-1,1))                  :rem 169
28Ø Q(I)=N(S)                                    :rem 18Ø
29Ø NEXTI                                        :rem 35
3ØØ REM-PLAY IT                                  :rem 119
3Ø5 POKEVO,15                                    :rem 254
31Ø FORI=1TO8                                    :rem 13
32Ø POKES2,Q(I)                                  :rem 96
33Ø FORK=1TO35Ø:NEXT                             :rem 234
34Ø NEXTI                                        :rem 31
345 FORK=1TO3ØØ:NEXT:POKEVO,Ø:FORK=1TO5Ø:NEXT:POKE
    VO,15                                        :rem 8Ø
35Ø FORI=8TO1STEP-1                              :rem 171
36Ø POKES2,Q(I)                                  :rem 1ØØ
37Ø FORK=1TO35Ø:NEXT                             :rem 238
38Ø NEXTI                                        :rem 35
385 FORK=1TO35Ø:NEXT                             :rem 244
39Ø POKES2,Ø:POKEVO,Ø                            :rem 22
4ØØ PRINT"{CLR}YOU MAY NOW PLAY THE{2 SPACES}SCALE
     ON YOUR KEY-{4 SPACES}BOARD."               :rem 231
41Ø PRINT"{2 DOWN}{2 SPACES}--HIT {RVS}↑{OFF} TO Q
    UIT--"                                       :rem 9
42Ø GOSUB89Ø:IFA$="↑"THENPOKES2,Ø:POKEVO,Ø:GOTO12Ø
                                                 :rem 115
43Ø IFA$<"1"ORA$>"8"THEN42Ø                      :rem 193
44Ø A=VAL(A$)                                    :rem 175
45Ø POKEVO,15:POKES2,Q(A)                        :rem 252
46Ø IFPEEK(2Ø3)<>64THEN46Ø                       :rem 225
47Ø POKES2,Ø:GOTO42Ø                             :rem 18Ø
89Ø A$="":GETA$:IFA$=""THEN89Ø                   :rem 133
895 RETURN                                       :rem 134
9ØØ DATA 135,143,147,151,159,163,167,175,179,183,1
    87,191                                       :rem 231
91Ø DATA 195,199,2Ø1,2Ø3,2Ø7,2Ø9,212,215,217,219,2
    21,223                                       :rem 197
92Ø DATA 225,227,228,229,231,232,233,235,236,237,2
    38,239,24Ø                                   :rem 153
93Ø DATA "2212221","2122122","2122131","2322323","
    2222222"                                     :rem 141
94Ø DATA "2122212","1222122","2221221","2212212"
                                                 :rem 187
```

Major & Minor: VIC Music Theory

M. J. Winter

You can learn some of the essentials of musicianship on your VIC while it coaches you on keys, signatures, and scales. This program can serve as a model for writing other computer-assisted-education routines for the VIC. It also illustrates a nice memory-saving technique: The computer displays the instructions and then removes them from the program after they are no longer needed.

"This piece has six sharps. What key is that?"

"What's the key signature for B minor?"

Questions like these are asked of every music student. Music teachers feel that the answers should be almost automatic—not hopeful guesses or the result of lengthy calculations.

Major and minor keys is a natural topic for computer-assisted instruction. In order to identify a key, the signature must be seen and the scale heard. (With practice, hearing the scale may be enough.) The program "Major and Minor Scales and Keys" combines instruction and practice on musical keys with ear-training.

Until I saw the "Electric Eraser" technique by Louis Sander (*COMPUTE!'s Second Book of VIC*), the instructions in my program were a separate program by themselves. Now the rules for determining keys are given in lines 79–96. These lines are erased before the arrays are dimensioned, so be sure to save the program before you run it. The program fits in an unexpanded, 5K VIC. When typing in line 90, be sure to use the ? for each PRINT; otherwise, it won't fit.

After reading the rules for determining major and relative minor keys, the user sees a menu of three options:

1. Study signatures—a review of randomly selected keys
2. Give signature when told key
3. Tell key from scale and signature

In each option, ten examples are given, and for every example a scale is played. Options 2 and 3 correct errors and keep a score. At the end of the ten examples, the user may quit or return to the menu. To see the beginning instructions again, however, it is necessary to reload the program. If you quit the program, then wish to run it again, you must delete line 2 or type RUN3 instead of just RUN.

Major and Minor Music

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

```
2 GOTO79                                        :rem 219
3 GOSUB68:POKE36879,31:DEFFNR(X)=INT(RND(1)*X)+1:C
  =36876:VR=C+2:D=C-1                            :rem 99
4 DIM N%(25),MJ%(16,2),S$(13),N$(13,2),T%(13,2),SP
  %(7),FP%(5):E=C-2                              :rem 63
5 SH$="FCGDAEB":FL$="BEADG":D$="{HOME}{11 DOWN}"
                                                  :rem 6
6 SP$="{DOWN}{6 RIGHT}#{3 DOWN}#{4 UP}#{3 DOWN}#
  {3 DOWN}#{4 UP}#{3 DOWN}#"                    :rem 134
7 FP$="{5 RIGHT}{5 DOWN}Z{3 UP}Z{4 DOWN}Z{3 UP}Z
  {4 DOWN}Z"                                    :rem 136
8 SP%(1)=7:SP%(2)=11:SP%(3)=16:SP%(4)=2Ø:SP%(5)=24
  :SP%(6)=29:SP%(7)=33                          :rem 243
9 FP%(1)=9:FP%(2)=13:FP%(3)=19:FP%(4)=24:FP%(5)=29
                                                :rem 1Ø1
1Ø FORI=1TO25:READN%(I):NEXT:B$="{15 SPACES}"
                                                  :rem 6
11 DATA191,195,199,2Ø1,2Ø3,2Ø7,2Ø9,212,215,217,219
   ,221,223,225                                  :rem 25
12 DATA227,228,23Ø,231,232,234,235,236,237,238,239
                                                :rem 211
13 FORI=1TO16:READMJ%(I,1):NEXT:FORI=1TO16:READMJ%
   (I,2):NEXT                                   :rem 243
14 DATAØ,2,4,5,7,9,11,12,12,11,9,7,5,4,2,Ø,Ø,2,3,5
   ,7,9,11,12,12,1Ø,8,7,5,3,2,Ø                 :rem 2Ø7
15 FORI=1TO13:READS$(I),N$(I,1),T%(I,1),N$(I,2),T%
   (I,2):NEXT                                    :rem 79
17 DATANONE,C MAJOR,2,A MINOR,11,1 SHARP,G MAJOR,9
   ,E MINOR,6                                   :rem 252
18 DATA2 SHARPS,D MAJOR,4,B MINOR,1              :rem 31
19 DATA3 SHARPS,A MAJOR,11,F-SHARP MINOR,8        :rem 2
2Ø DATA4 SHARPS,E MAJOR,6,C-SHARP MINOR,3 :rem 2Ø3
21 DATA5 SHARPS,B MAJOR,1,G-SHARP MINOR,1Ø:rem 247
22 DATA6 SHARPS,F-SHARP MAJOR,8,D-SHARP MINOR,5
                                                :rem 128
23 DATA7 SHARPS,C-SHARP MAJOR,3,A-SHARP MINOR,12
                                                :rem 165
```

```
25 DATA1 FLAT,F MAJOR,7,D MINOR,4,2 FLATS,B-FLAT M
   AJOR,12,G MINOR,9                          :rem 123
26 DATA3 FLATS,E-FLAT MAJOR,5,C MINOR,2       :rem 32
27 DATA4 FLATS,A-FLAT MAJOR,10,F MINOR,7      :rem 82
28 DATA5 FLATS,D-FLAT MAJOR,3,B-FLAT MINOR,12
                                              :rem 165
29 FORQ=1TO10:K=FNR(13):MD=FNR(2):PRINT"{CLR}"Q:IF
   OP<>1THENPRINTTAB(16)"{UP}SC"SC            :rem 49
30 IFOP<>3THENPRINT"{RED}{RVS}KEY{OFF}{BLU} ";N$(K
   ,MD)                                       :rem 12
31 IFOP<>2THENPRINT"{DOWN}{RED}{RVS}SIGNATURE{OFF}
   {BLU}{2 SPACES}";S$(K)                     :rem 46
32 PRINTTAB(11);:V=VAL(S$(K))                 :rem 203
33 KS$=LEFT$(SH$,V):IFMID$(S$(K),3,1)="F"THENKS$=L
   EFT$(FL$,V)                                :rem 250
34 IFK=1THENKS$="NONE"                        :rem 81
35 IFOP=1THENPRINTKS$                         :rem 130
36 IFOP=2THENNP=3:GOSUB60                     :rem 99
37 GOSUB50:IFMID$(S$(K),3,1)="S"THEN PRINTD$;LEFT$
   (SP$,SP%(V)+2)                             :rem 200
38 IFMID$(S$(K),3,1)="F"THENPRINTD$;LEFT$(FP$,FP%(
   V)+2)                                      :rem 131
39 GOSUB78:PRINT"{HOME}"LEFT$(D$,10)"{RED}PRESS
   {RVS}S{OFF}"                               :rem 242
40 IFPEEK(197)<>41THEN40                      :rem 124
41 FORJ=1TO16:POKEC,N%(T%(K,MD)+MJ%(J,MD)):POKEVR,
   15:GOSUB78                                 :rem 135
42 IFJ=8THENPOKEC,0:GOSUB78                   :rem 218
43 NEXT:POKEC,0:POKEVR,0:GOSUB78:IFOP=3THENNP=3:GO
   SUB74                                      :rem 126
44 PRINT"{HOME}"LEFT$(D$,10)"PRESS {BLU}{RVS}C
   {OFF}{15 SPACES}"                          :rem 184
45 IFPEEK(197)<>34THEN45                      :rem 136
46 NEXTQ:PRINT"{4 UP}CONTINUE?":PRINT"{DOWN}PRESS
   {SPACE}{RVS}Y{OFF} OR {RVS}N{OFF}"         :rem 38
47 IFPEEK(197)<>11ANDPEEK(197)<>28THEN47      :rem 85
48 IFPEEK(197)=28THENEND                      :rem 191
49 RUN3                                       :rem 149
50 PRINTD$;:PRINT"{BLU}"                      :rem 191
51 PRINT"[A]***[M]M**************"            :rem 27
52 PRINT"-{3 SPACES}[M]N                      :rem 104
53 PRINT"[Q]***N***************"              :rem 50
54 PRINT"-{2 SPACES}N[M]                      :rem 106
55 PRINT"[Q]*N*[M]***************"            :rem 27
56 PRINT"-G                                   :rem 190
57 PRINT"[Q]H***M**************"              :rem 61
58 PRINT"- M[2 @]N                            :rem 220
59 PRINT"[Z]****[G]**************":RETURN     :rem 43
```

```
60 POKE198,Ø:PRINT:PRINT"{BLU}SIGNATURE{RED}":INPU
   TA$                                          :rem 129
61 IFA$=S$(K)THENPRINT"RIGHT":GOTO64              :rem 6
62 NP=NP-1:IFNP=2THENPRINT"{UP}"B$:PRINT"{3 UP}
   {BLU}";:GOTO6Ø                               :rem 253
63 PRINT"{2 UP}{BLK}"S$(K)"{6 SPACES}"           :rem 67
64 IFS$(K)="NONE"THENSC=SC+NP:RETURN             :rem 56
65 POKE198,Ø:A$="":PRINT"{BLU}WHICH?{3 SPACES}
   {RED}":INPUTA$                               :rem 223
66 IFA$=KS$THENPRINT"{DOWN}RIGHT":SC=SC+NP:RETURN
                                                :rem 116
67 NP=NP-1:PRINT"{UP}{BLK}"KS$"{BLU}"B$:GOSUB78:RE
   TURN                                          :rem 62
68 PRINT"{CLR}SELECT ONE":PRINT"{DOWN}{RED}{RVS}1
   {OFF}{BLU} STUDY SIGNATURES"                  :rem 62
69 PRINT"{DOWN}{RED}{RVS}2{OFF}{BLU} GIVE SIGNATUR
   E WHEN{4 SPACES}TOLD KEY"                    :rem 141
7Ø PRINT"{DOWN}{RED}{RVS}3{OFF}{BLU} TELL KEY FROM
    SCALE{4 SPACES}& SIGNATURE"                 :rem 233
71 PRINT"{2 DOWN}PRESS {RED}{RVS}1{OFF} {RVS}2
   {OFF}{BLU} OR{RED} {RVS}3{OFF}":PRINT"{DOWN}
   {BLU}YOU'LL GET 1Ø EXAMPLES"                 :rem 249
72 GET RP$:IFVAL(RP$)<1OR VAL(RP$)>3THEN72:rem 227
73 OP=VAL(RP$):RETURN                            :rem 9Ø
74 POKE198,Ø:PRINT"{UP}{RVS}KEY{OFF}"B$:PRINT"{UP}
   {4 RIGHT}";:INPUTA$                          :rem 161
75 IFA$=N$(K,MD)THENPRINT"{2 DOWN}RIGHT":SC=SC+NP:
   RETURN                                       :rem 142
76 NP=NP-1:IFNP=2THEN74                         :rem 18Ø
77 PRINT"{2 UP}{BLK}"N$(K,MD)"{RED}":GOSUB78:RETUR
   N                                             :rem 95
78 FORKK=1TO499:NEXT:RETURN                      :rem 54
79 A=PEEK(61)+256*PEEK(62)+3:POKE2,INT(A/256):POKE
   1,A-256*PEEK(2)                              :rem 172
8Ø IFERTHENPOKEA-2,Ø:POKEA-1,Ø:POKE45,PEEK(1):POKE
   46,PEEK(2):RUN3                              :rem 25Ø
81 POKE36879,26:PRINT"{CLR}{5 DOWN}"TAB(4)"{BLU}MA
   JOR AND MINOR"                               :rem 192
82 PRINTTAB(5)"{DOWN}SCALES & KEYS":PRINTTAB(7)"
   {5 DOWN}PRESS {RVS}S{OFF}"                     :rem 9
83 IFPEEK(197)<>41THEN83                        :rem 138
84 PRINT"{CLR}A SIGNATURE INDICATES":PRINT"{DOWN}B
   OTH A MAJOR & MINOR":PRINT"{DOWN}KEY.";:rem 115
85 PRINT"THE MINOR IS":PRINT"{DOWN}ALWAYS A THIRD
   {SPACE}LOWER.{DOWN}"                         :rem 177
86 GOSUB51:PRINT"{RED}{DOWN}C MAJOR/A MINOR":PRINT
   "{2 DOWN}{BLU}PRESS {RED}{RVS}C{OFF}";:PRINT"
   {8 UP}Q{LEFT}{2 UP}W"                         :rem 78
87 IFPEEK(197)<>34THEN87                        :rem 148
```

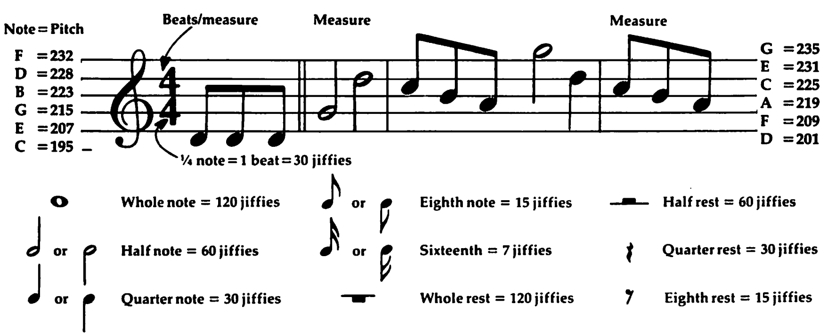
```
88 PRINT"{BLU}{CLR}":GOSUB51:PRINTTAB(6)"{RED}
   {5 UP}Z{2 RIGHT}{3 DOWN}W{2 DOWN}{LEFT}Q":PRINT
   "{DOWN}F MAJOR/D MINOR{DOWN}"              :rem 255
89 PRINT"{BLU}PRESS {RED}{RVS}D{UP}":IFPEEK(197)<>
   18THEN89                                    :rem 84
90 PRINT"{BLK}FOR 2 OR MORE FLATS":PRINT"{DOWN}THE
    MAJOR KEYNOTE IS":PRINT"{DOWN}THE NEXT-TO-LAST
    FLAT{BLU}"                                 :rem 21
91 PRINT"{2 DOWN}PRESS {RVS}B{OFF}":PRINT"{HOME}
   {RED}"TAB(7)"{3 DOWN}Z{2 DOWN}{RIGHT}W{LEFT}
   {2 DOWN}Q{DOWN}{LEFT}¯{2 DOWN}{LEFT} ":PRINT"
   {DOWN}B-FLAT MAJOR/G MINOR"                :rem 243
92 IFPEEK(197)<>35THEN92                      :rem 141
93 PRINT"{BLU}{CLR}":GOSUB51:PRINT"{HOME}{DOWN}"TA
   B(6)"#{3 DOWN}#{RED}{RIGHT}{UP}W{2 DOWN}{LEFT}Q
   ":PRINT"{4 DOWN}D MAJOR/B MINOR"           :rem 140
94 PRINT"{2 DOWN}{BLU}LOOK ABOVE AND BELOW":PRINT"
   {DOWN}THE LAST SHARP":PRINT"{2 DOWN}PRESS {RVS}
   X"                                          :rem 64
95 IFPEEK(197)<>26THEN95                      :rem 147
96 ER=1:GOTO79                                 :rem 87
```

VIC Musician

Blake Wilson

You can have your VIC playing a song in the background while some other BASIC program is running. It will even play while you're programming. This article will also show you how to take music from a printed score and enter it into your VIC.

I have always believed that music in any form was above my head. I own a computer retail store, and several customers have asked how they could produce music that would run *continuously,* without delays, during a program. My wife helped me to understand just what all those incomprehensible symbols on a piece of sheet music actually mean. I wanted the explanation in terms that I (and VIC) could understand. Here's her response:



Don't be concerned that some shafts go up and some down, or that notes may be joined with arcs or barlike flags. These exist only to confuse computer people. You need be concerned only with the duration of a note (filled in or open, whether it has a shaft or not, and the number of bars or flags. By the way, a dot after a note increases its duration by 50 percent) and its pitch (as determined by its altitude on the staff). Any pitch of less than 128 is a rest or silence.

If notes are stacked, just take the highest one. There are generally two staffs (groups of lines). The lower set is often for harmony; usually we can ignore them.

Look at the data at line 190. The first item is duration. If a note is filled in and has a shaft and one flag, it is an eighth note and will be played for 15 jiffies (refer to the figure for other symbols). The next data item represents the pitch. The third data item is duration, again followed by pitch.

Inspect a musical score and write data for each note, first duration and then pitch. Place the data in the program to replace lines 190 through 240. The actual line numbers mean nothing, but the first 71 data items representing the program must come first.

The program is limited to 61 notes due to the size of the cassette buffer. In practice, 61 notes should be enough for most programs. I use only 41 notes in the sample program. If your creation uses fewer than 61 notes, end your data with 1,1. This instructs the computer to replay from the first.

Be sure that the FOR-NEXT loop in line 70 is adjusted up or down so that all the DATA is READ. If when you RUN the program you get an OUT OF DATA error on line 70, try decreasing the loop by lowering the number after the TO in line 70. If the music stops or takes a long pause, be sure the last two DATA items are 1 (if you have fewer than 61 notes), or increase the number after the TO in line 70.

Once the data has been POKEd into the buffer and you've done a SYS to 830, the music plays continuously even if you execute NEW or write another program. The music will continue until:

1. You type a RUN/STOP–RESTORE
2. Attempt to use the cassette
3. Or get so tired of it that you shut off the VIC.

The program is driven by the *hardware interrupt*. Each jiffy (1/60 second) a counter ($033D=829) is reduced by one. If this decrement results in a value of zero, the next duration is placed in the counter, the next note is placed in the sound generator ($900C=36876), and the note count is increased by two and stored at ($033C=828). The computer then goes on its way, updating the realtime clock and scanning its keyboard.

The computer is not slowed appreciably. Most of the time, unless the note is to be changed, the time wasted is 120 microseconds per second or 120 parts per million. If the note must be changed, the time lost is a few thousandths of a second.

Note: A complete listing of musical pitch values, including sharps and flats, is in the *VIC-20 Programmer's Reference Guide.*

VIC Musician

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

```
10 REM**VIC MUSICIAN**                              :rem 40
20 REM CONTINUOUSLY PLAYS                           :rem 139
30 REM MAPLE LEAF RAG                               :rem 168
60 POKE36878,15 : REM SET VOLUME TO MAX             :rem 162
70 FORI=830TO915:READC:POKEI,C:NEXT                 :rem 228
80 SYS 830 :REM STARTS MUSIC                        :rem 130
90 REM ******* SET-UP FOLLOWS *******               :rem 125
100 DATA120,169,5,141,60,3,169,6,141,61,3,169
                                                    :rem 204
110 DATA133,133,0,169,3,133,1,169,93,141            :rem 211
120 DATA20,3,169,3,141,21,3,88,96                   :rem 128
130 REM **** MUSIC PGM FOLLOWS ******               :rem 167
140 DATA206,61,3,208,28,72,152,72,172,60            :rem 219
150 DATA3,200,177,0,141,61,3,200,177,0,201 :rem 32
160 DATA1,240,12,141,12,144,140,60,3,104            :rem 188
170 DATA168,104,76,191,234,160,255,208,243 :rem 77
180 REM *** MUSIC DATA FOLLOWS ******               :rem 184
190 DATA 30,217,30,209,30,217,15,209,30,217,15,209
    ,15,217,15,221,15,225,15,221,15,217     :rem 60
200 DATA 15,209                                     :rem 217
210 DATA 15,203,30,209,90,195,15,203,15,209,15,195
    ,15,203,30,209,30,179,15,187,30,179     :rem 82
220 DATA 30,179,15,187                              :rem 60
230 DATA 30,179,15,187,15,195,15,179,15,187,30,195
    ,30,179,15,187,15,195,30,179,30,187     :rem 127
240 DATA 15,151,30,179,30,179,30,187,30,195,1,1
                                                    :rem 0
```

Chord Organ

Scott Oglesby

"Chord Organ" will change your VIC into a programmable organ with a 99-step memory, 19 user-defined chords, and much more. Requires at least 8K expansion memory.

Since the *VIC-20 Programmer's Reference Guide* and the operator's manual both have tables with musical notes and the corresponding values to POKE into sound registers, building a chord should be easy: Translate the component notes into numbers, POKE them into the registers, and set the volume. There are many note combinations that produce nice chords; once you find a few, you might want to add a three-part fanfare to a deserving videogame or educational program.

But programming the tune, especially if you're not sure exactly how it will sound, can be quite unwieldy. Changing the speed or order of the tune would require much editing and rearranging of DATA statements. "Chord Organ" is a convenient musical utility made for such experimentation, and also a versatile musical instrument.

Chord Organ transforms the VIC into a 73-chord programmable organ with a 99-step memory, programmable duration and articulation, and 19 user-defined chords. There are two major chords, a minor chord, and a seventh chord for each of 12 key signatures; two composite notes for each key; and one diminished fifth chord.

Finding a Chord

To become familiar with the keyboard layout, run the program and have Table 1 handy. The keyboard is arranged in 12 columns (one for each key signature) by 4 rows (one for each chord type). There are two chord types listed for each row in Table 1; the second type is accessed by holding down the SHIFT key when pressing the key for your chord.

To find a chord, find the key that is in the column for its key signature and the row for its type. For example, press the 0 key, which is the key for a D major chord. You should hear the chord, and see a D appear after the 1 in the lower left area of your screen.

Move down to the next row and press P. A low D major chord will sound, and D L will replace D on the screen. Down another row to the : key will give you D minor (none of the minor chords sound as good as the majors), and D M will replace D L. The slash key (/) in the seventh-chord row will play a D7, and D 7 will take the place of D M.

Each column operates this way except for G and C. For these columns, the SHIFT key is in the way. You find the seventh chord for G and C by using CRSR DOWN and CRSR RIGHT, respectively.

There is some overlap in the first and second rows, where there are more than 12 keys; so two keys will produce the same chord. Also note that the DEL key plays a rest (RST on the screen), which is used in programming a tune.

Table 1. Modified Keyboard

Type 1 /Type 2	Key Signature											
	F	B^b	E^b	A^b	D^b	G^b	B	E	A	D	G	C
Major /High	1	2	3	4	5	6	7	8	9	0	+	-
Low /Low	Q	W	E	R	T	Y	U	I	O	P	@	↑
Minor /Custom	A	S	D	F	G	H	J	K	L	:	;	=
Seventh/Custom	Z	X	C	V	B	N	M	,	.	/	↑ CRSR ↓	← CRSR →

The Second Keyboard

To play the chords on the second keyboard, hold down the SHIFT key when playing. SHIFT-0 will play what sounds like a single note, but is really three notes with nearly identical pitches. The harmonics give this note a sharper, stronger tone. A D in reverse will appear after the 1 on the screen. SHIFT-P gives a lower note of the same timbre and displays D L in reverse. SHIFT-: and SHIFT-/ are in the custom rows and have no effect because they're not yet defined.

One permanent chord in the custom rows is G major diminished 5th, whose name is /SA in reverse and is played by pressing SHIFT-A.

Notice that the chord columns are arranged in a circle of fourths instead of chromatically: It is easier to play that way once you have mastered the format. For any key signature, all the common chords (tonic, dominant, and subdominant) are adjacent, and transposition is simplified. The only way to master the keyboard is to experiment with it.

Entering a Tune

Even though you've been playing the keyboard like a normal instrument, the organ has been in program mode, waiting for you to enter the chord for step 1. Since the computer allows you to change your chord before entering it, it has been in a built-in manual play mode.

Find the first chord of your planned tune (use the Fanfare from Table 2 if you don't have one in mind), play it so its name appears after the 1 on the screen, and hit RETURN. An underline, acting as a cursor, will appear to the right, asking you for the chord's duration with respect to the unit time. The unit time, whose chronological value you will enter later, is the length of the shortest note of the tune. If the present note is twice as long, enter 2; three times, enter a 3; and so on. If the note is the shortest, enter a 1, which is the default.

Once you've entered the duration, the underline will appear farther to the right, asking for the articulation code. A 0 here, the default, signifies a slur (no break) between the present chord and the next. A 1 will produce a short break between chords. A 2 will produce a gradual decay at the end of the chord.

Once you've entered the articulation, a prompt, with the next step number, will appear on the line below and you can enter the next chord. If you make a mistake, look up JUMP in the following text and see how to move back to the step with the error in it.

If you are entering Fanfare from Table 2, your first six keystrokes are: 9; 3; 1; 9; 3; 0.

A square after the letter in a chord name signifies a flat. If your VIC has sufficient memory, you may want to add a custom character set that includes a conventional flat sign.

Using the Function Keys

If you wish to use one of the commands, simply press f6, Review, anytime the computer is ready for a chord—f6 will bring the command list on the screen. A touch of the appropriate key enters the command. The commands cannot be used as steps in your tune.

PLAY (f1)—plays the tune you've programmed in. The computer will ask for the unit time, which is simply a value in a FOR-NEXT delay loop. Since the durations of all the chords

Table 2. Fanfare

1.	A	3	1
2.	A	3	0
3.	D L	2	0
4.	E L	1	0
5.	A	3	0
6.	D L	2	0
7.	E L	1	0
8.	A	3	0
9.	G	3	0
10.	B	3	0
11.	C	3	1
12.	C	3	0
13.	F L	2	0
14.	G L	1	0
15.	C	3	0
16.	Bb L	3	0
17.	D 7	3	0
18.	G L	3	0
19.	D 7	3	0
20.	A	4	2

TIME: 100

Tune

(all chords are SHIFTed)

1.	F	3	2
2.	F	3	2
3.	F	3	2
4.	Ab	3	0
5.	G	1	1
6.	G	2	0
7.	F	2	1
8.	F	2	0
9.	E	2	0
10.	F	4	2

TIME: 150

are relative to this value, you can easily speed up or slow down the entire song.

LIST (f2)—lists the chords that are in memory. You'll be asked for a start and end, and you can use CTRL to slow down the scrolling. If you enter a number less than 1 or greater than 99, you will get an error message.

SEARCH (f3)—enter a chord in the same way you would when programming a tune. The computer will list all steps that contain the chord.

DIAGRAM (f4)—enter a chord and the computer will display the values, in order, that comprise the chord.

JUMP (f5)—enter a number (if it is greater than 99, you will get an error message), and the computer will bring you to that step.

REVIEW (f6)—displays the list of commands.

CLEAR (f7)—clears the tune and returns you to step 1. A fail-safe feature (you must hit RETURN to verify that you want to clear) prevents you from accidentally losing your song.

EXIT (f8)—ends the program. This command has the same protection as CLEAR.

REVERSE (CTRL-1)—reverses your tune; another REVERSE will restore it to the way it was. If your tune doesn't sound right, maybe you've been going at it the wrong way.

EXTEND (CTRL-3)—JUMPs you to one step beyond the end of the tune so you can add to it. The same error message will appear if this brings you to beyond 99.

CUSTOM (CTRL-2)—in this mode, you can add up to 19 of your own chords to the 73 built in, for a total of 92 chords. Hitting CTRL-2 will bring you first to the tuning mode, where you select and tune one voice at a time. An instruction list remains on the screen for the duration of the tuning.

f1 selects the voice; voice 1 is location 36874, voice 2 is 36875, and voice 3 is 36876. f3 and f5 add and subtract 10 from the value of the voice you are tuning (all three values and the voice number are displayed continuously), and the cursor keys add and subtract 1 for fine tuning. f7 shuts off the voice by changing its value to 127.

When you are done tuning, hit RETURN. You will be given the option to save your chord. If you do save it, the computer will ask, WHAT KEY? The key you choose will be on the SHIFTed keyboard, but do not hold down the SHIFT key when choosing your key. The keys available are: S through ; (third row) and Z through / (bottom row). Once you have selected a valid key, the computer will give you the chord name, the way it will appear in a tune listing, and the chord number. It is wise to keep track of what keys have been used up.

How the Chords Are Arranged

For chords 0–63, the number for the chord is the value of PEEK(197) for the key that plays it (see Appendix I). The values for these chords are listed, four per line, starting at line 8000.

Chords 64–127 (the same PEEK number plus 64 for each one) of the second keyboard are listed, three per line, starting at line 8200. For each chord, the name, in quotes, is listed first, followed by the values for the first, second, and third registers. Some silent chords may have a 0, 1, or a 3 for the first register value; this may not make sense since they all are soundless when POKED into a register. These are codes that tell whether the chord is a rest, can be defined by the user, or is an invalid key. When you are typing in this program, be sure to enter each number as is.

Chord Organ

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

```
10 Q=127:V=36874:POKEV+4,15:POKEV+5,110    :rem 239
20 DIMN$(Q),A%(Q+1),B%(Q),C%(Q),P(99),R(99),S(99):
   FORX=.TOQ:READN$(X),A%(X),B%(X),C%(X)    :rem 10
30 NEXT:DE$=CHR$(20):LF$=CHR$(141):CM$="{UP}"+LF$+
   "{CYN}>":CR$="{@}{LEFT}":A%(128)=1    :rem 107
40 PI=1:PT=1:FORX=1TO99:P(X)=15:NEXT           :rem 2
50 PRINT"{CLR}{WHT}VIC-20 CHORD ORGAN{DOWN}":PRINT
   " LIST OF COMMANDS{5 SPACES}{19 I}{DOWN}"
                                              :rem 124
60 GOTO4800                                   :rem 107
200 PI=PI+1:IFPI>PTTHENPT=PI                   :rem 74
201 IFPI>99THENPI=99:PT=99:GOTO4700           :rem 173
203 PRINT"{WHT}>"MID$(STR$(PI),2)TAB(5);:IFPI>PTTH
    ENPT=PI                                   :rem 226
205 GOSUB4000:IFA$=CHR$(13)THEN300            :rem 190
210 IFA$="{F1}"THEN4200                       :rem 130
220 IFA$="{F2}"THEN5050                       :rem 139
230 IFA$="{F3}"THEN5400                       :rem 136
240 IFA$="{F4}"THEN5450                       :rem 146
250 IFA$="{F5}"THEN5100                       :rem 136
260 IFA$="{F6}"THEN5250                       :rem 147
270 IFA$="{F7}"THEN5300                       :rem 141
280 IFA$="{F8}"THEN5350                       :rem 151
282 IFA$="{BLK}"THEN5550                      :rem 159
284 IFA$="{WHT}"THEN9000                       :rem 16
286 IFA$="{RED}"THEN5200                       :rem 39
290 IFA$=""ANDA%(P)>2THENP(PI)=P:PRINTN$(P)"
    {3 LEFT}";                                :rem 128
295 GOTO205                                   :rem 112
300 PRINTN$(P(PI))TAB(10)CR$;:GOSUB4900        :rem 33
350 R(PI)=BN:PRINT" {LEFT}";:IFBN=.THENR(PI)=1:PRI
    NT"1";                                    :rem 152
400 PRINTTAB(15)CR$;:GOSUB4900                 :rem 42
450 S(PI)=BN:PRINT" {LEFT}";:IFBN=.THENPRINT"0";
                                              :rem 181
500 PRINTLF$;:GOTO200                          :rem 24
4000 A$="":P=PEEK(197):POKEV+4,15:IFP=64THENP=15
                                               :rem 26
4002 IFPEEK(653)AND3THENP=P+64                 :rem 16
4004 POKEV,A%(P):POKEV+1,B%(P):POKEV+2,C%(P)
                                               :rem 32
4005 IFA%(P)<2THENGETA$                        :rem 65
4008 RETURN                                   :rem 172
4200 PRINTCM$"PLAY{3 SPACES}":PRINT"TIME:"CR$;:GOS
     UB4900:T=BN:PRINT                        :rem 146
```

```
4210 FORQ=1TOPT:POKEV+4,15:P=P(Q):POKEV+4,15:POKEV
     ,A%(P):POKEV+1,B%(P):POKEV+2,C%(P)     :rem 95
4220 FORX=1TOR(Q)*T:NEXT:IFS(Q)=1THENPOKEV,.:POKEV
     +1,.:POKEV+2,.:GOTO4250               :rem 102
4230 IFS(Q)=2THENFORY=15TO.STEP-1:POKEV+4,Y:NEXT:P
     OKEV,.:POKEV+1,.:POKEV+2,.             :rem 89
4250 NEXT:POKEV,.:POKEV+1,.:POKEV+2,.:GOTO203
                                            :rem 22
4700 PRINTLF$"{GRN}**{CYN}NUMBER OUT OF RANGE":GOT
     O203                                   :rem 46
4800 PRINT"F1-PLAY":PRINT"F2-LIST":PRINT"F3-SEARCH
     ":PRINT"F4-DIAGRAM"                   :rem 113
4801 PRINT"F5-JUMP":PRINT"F6-REVIEW":PRINT"F7-CLEA
     R":PRINT"F8-EXIT"                      :rem 20
4805 PRINT"{DOWN}C1-REVERSE":PRINT"C2-CUSTOM":PRIN
     T"C3-EXTEND{DOWN}":GOTO203            :rem 135
4900 B$="":REM NUMERIC INPUT ROUTINE       :rem 155
4910 GETA$:IFA$=""THEN4910                 :rem 191
4920 IFA$>"/"ANDA$<":"THENB$=B$+A$:PRINTA$CR$;
                                            :rem 78
4930 IFA$=DE$ANDLEN(B$)>.THENB$=LEFT$(B$,LEN(B$)-1
     ):PRINTDE$CR$;                        :rem 227
4940 IFA$=CHR$(13)THENBN=VAL(B$):RETURN    :rem 104
4950 GOTO4910                              :rem 217
5050 PRINTCM$"LIST{3 SPACES}":PRINT"FROM:"CR$;:GOS
     UB4900:B1=BN                          :rem 249
5060 PRINT"{2 SPACES}TO:"CR$;:GOSUB4900:B2=BN
                                           :rem 111
5090 IFB1*B2<1THEN4700                     :rem 213
5092 IFB1>99ORB2>99THEN4700                 :rem 65
5095 PRINT"{WHT}":FORQ=B1TOB2:PRINTQ;TAB(5)N$(P(Q)
     );TAB(10)R(Q);TAB(15)S(Q):NEXT:GOTO203:rem 62
5100 P(PI)=15:PRINTCM$"JUMP TO: "CR$;:GOSUB4900
                                           :rem 246
5103 IFBN>99THEN4700                       :rem 146
5105 PI=BN:PRINT:IFPT<PITHENPT=PI          :rem 227
5110 GOTO203                               :rem 149
5200 PRINTCM$"EXTEND ":IFPT>99THEN4700      :rem 43
5210 PI=PT+1:GOTO203                       :rem 166
5250 PRINT"{CLR}":GOTO4800                 :rem 111
5300 PRINTCM$">>CLEAR>>":PRINT"RETURN VERIFIES"
                                           :rem 244
5310 PRINT"OTHER KEY ABORTS"               :rem 208
5320 GETA$:IFA$=""THEN5320                 :rem 183
5330 IFA$=CHR$(13)THENPRINT"CLEARING":GOTO40
                                           :rem 206
5340 PRINT"CLEAR ABORTED":GOTO203           :rem 13
5350 PRINTCM$">>EXIT>>":PRINT"RETURN VERIFIES"
                                           :rem 204
```

```
5360 PRINT"OTHER KEY ABORTS"                    :rem 213
5370 GETA$:IFA$=""THEN5370                      :rem 193
5380 IFA$=CHR$(13)THENEND                       :rem 189
5390 GOTO203                                    :rem 159
5400 PRINTCM$"SEARCH ";                          :rem 63
5410 GOSUB4000:IFA$=""ANDA%(P)>2THENPRINTN$(P)"
     {3 LEFT}";:J=P                              :rem 61
5420 IFA$<>CHR$(13)THEN5410                     :rem 232
5430 PRINT:FORQ=1TO99:IFP(Q)=JTHENPRINTQ;N$(P(Q));
     R(Q);S(Q)                                  :rem 236
5440 NEXT:GOTO203                                :rem 20
5450 PRINTCM$"DIAGRAM ";                        :rem 131
5455 GOSUB4000:IFA$=""ANDA%(P)>2THENPRINTN$(P)"
     {3 LEFT}";:J=P                              :rem 70
5470 IFA$<>CHR$(13)THEN5455                     :rem 246
5480 PRINT:PRINTA%(J);B%(J);C%(J):GOTO203 :rem 169
5550 PRINTCM$"REVERSE":FORQ=1TOINT(PT/2)+.5
                                                :rem 194
5560 J=P(Q):P(Q)=P(PT-Q):P(PT-Q)=J             :rem 249
5570 J=R(Q):R(Q)=R(PT-Q):R(PT-Q)=J               :rem 2
5580 J=S(Q):S(Q)=S(PT-Q):S(PT-Q)=J               :rem 7
5590 NEXT:GOTO203                                :rem 26
7980 REM CHORDS 0-63{2 SPACES}BY 4'S            :rem 142
8000 DATA"F{2 SPACES}",240,237,164,"E{V} ",238,2
     35,153,"D{V} ",227,217,210,"B{2 SPACES}",22
     3,212,204                                   :rem 58
8010 DATA"A{2 SPACES}",219,207,198,"G{2 SPACES}",2
     39,175,201,"F{2 SPACES}",240,237,164,"RST",3,
     0,0                                         :rem 83
8020 DATA"C{2 SPACES}",235,231,134,"B{V}L",187,2
     09,200,"A{V}L",217,204,194,"G{V}L",212,19
     8,187                                      :rem 237
8030 DATA"E L",207,191,179,"D L",219,212,147,"C L"
     ,195,175,160,"RST",1,0,0                    :rem 56
8040 DATA"",1,0,0,"F M",225,217,165,"E{V}M",221,
     212,153,"D{V}M",227,217,208                :rem 174
8050 DATA"B M",223,212,202,"A M",240,183,160,"G M"
     ,228,221,175,"C 7",225,215,187             :rem 222
8060 DATA"",1,0,0,"",1,0,0,"B{V}7",187,209,216,"
     A{V}7",217,204,212                         :rem 180
8070 DATA"G{V}7",212,231,198,"E 7",231,223,228,"
     D 7",219,212,194,"G 7",228,175,209         :rem 119
8080 DATA"",0,0,0,"F 7",225,229,164,"E{V}7",238,
     227,153,"D{V}7",227,217,223                :rem 124
8090 DATA"B 7",223,212,218,"A 7",219,207,214,"",1,
     0,0,"",1,0,0                                :rem 53
8100 DATA"",1,0,0,"B{V}M",187,209,140,"A{V}M",
     212,204,190,"G{V}M",212,198,182            :rem 106
8110 DATA"E M",207,191,176,"D M",219,210,147,"C M"
     ,191,175,155,"",1,0,0                       :rem 60
```

```
8120 DATA"F L",225,219,164,"E{V}L",2Ø3,187,173,"
     D{V}L",199,179,165,"B L",191,17Ø,152
                                           :rem 145
8130 DATA"A L",183,2Ø7,198,"G L",228,175,191,"F L"
     ,225,219,164,"",Ø,Ø,Ø                  :rem 71
814Ø DATA"B{V} ",187,232,2ØØ,"A{V} ",236,2Ø4,1
     94,"G{V} ",212,198,221,"E{2 SPACES}",2Ø7,22
     3,217                                   :rem 2
815Ø DATA"D{2 SPACES}",2Ø1,219,212,"C{2 SPACES}",2
     35,231,134,"B{V} ",187,232,2ØØ,"",1,Ø,Ø
                                           :rem 249
816Ø REMCHORDS 64-127 BY 3'S               :rem 239
82ØØ DATA"{RVS}F{OFF}{2 SPACES}",232,2Ø9,164,"
     {RVS}E{V}{OFF} ",23Ø,2Ø4,154,"{RVS}D{V}
     {OFF} ",227,199,144                    :rem 52
821Ø DATA"{RVS}B{OFF}{2 SPACES}",239,223,128,"
     {RVS}A{OFF}{2 SPACES}",237,219,184,"{RVS}G
     {OFF}{2 SPACES}",235,215,176          :rem 192
822Ø DATA"{RVS}F{OFF}{2 SPACES}",232,2Ø9,164,"RST"
     ,3,Ø,Ø,"{RVS}C{OFF}{2 SPACES}",225,195,134
                                           :rem 144
823Ø DATA"{RVS}B{V}L{OFF}",221,186,187,"{RVS}A
     {V}L{OFF}",178,179,178,"{RVS}G{V}L{OFF}",
     212,168,169                           :rem 244
824Ø DATA"{RVS}E L{OFF}",157,157,158,"{RVS}D L
     {OFF}",2Ø1,146,147,"{RVS}C L{OFF}",194,133,13
     4                                     :rem 163
825Ø DATA"",1,Ø,Ø,"",1,Ø,Ø,"{RVS}/SA{OFF}",239,175
     ,226                                  :rem 247
826Ø DATA"",,,,"",,,,"",,,Ø                :rem 2Ø2
827Ø DATA"",,,,"",,,,"",1,,Ø               :rem 252
828Ø DATA"",1,,,"",1,,,"",,,Ø               :rem 46
829Ø DATA"",,,,"",,,,"",,,Ø                :rem 2Ø5
83ØØ DATA"",,,,"",1,,,"",1,,Ø               :rem 39
831Ø DATA"",,,,"",,,,"",,,Ø                :rem 198
832Ø DATA"",,,,"",,,,"",,,Ø                :rem 199
833Ø DATA"",1,,,"",,,,"",,,Ø               :rem 249
834Ø DATA"",,,,"",,,,"",,,Ø                :rem 2Ø1
835Ø DATA"",,,,"",,,,"",1,,Ø               :rem 251
836Ø DATA"{RVS}F L{OFF}",2Ø9,164,164,"{RVS}E{V}L
     {OFF}",2Ø4,153,154,"{RVS}D{V}L{OFF}",199,14
     3,144                                  :rem 36
837Ø DATA"{RVS}B L{OFF}",128,129,128,"{RVS}A L
     {OFF}",184,183,184,"{RVS}G L{OFF}",215,174,17
     5                                     :rem 175
838Ø DATA"{RVS}F L{OFF}",2Ø9,164,164,"",1,,,"{RVS}
     B{V}{OFF} ",238,187,186                :rem 85
839Ø DATA"{RVS}A{V}{OFF} ",236,217,18Ø,"{RVS}G
     {V}{OFF} ",234,234,17Ø,"{RVS}E{OFF}
     {2 SPACES}",231,2Ø7,158                :rem 56
```

```
8400 DATA"{RVS}D{OFF}{2 SPACES}",228,202,148,"
     {RVS}C{OFF}{2 SPACES}",225,195,134,"{RVS}B
     {V}{OFF} ",238,187,186,"",1,,0          :rem 215
9000 PRINT"{CLR}":POKE36878,15                :rem 56
9010 PRINT"{2 SPACES}CUSTOM CHORD TUNER{DOWN}"
                                             :rem 133
9020 PRINT"F1-CHANGE VOICE":PRINT"F3-ADD 10 TO VOI
     CE":PRINT"F5-SUBTRACT 10"               :rem 204
9030 PRINT"F7-SHUT OFF VOICE":PRINT"CSR DOWN-SUBTR
     ACT 1":PRINT"CSR RIGHT-ADD 1"           :rem 101
9040 PRINT"RETURN-EXIT TUNING{2 DOWN}":PRINT"
     {2 SPACES}1{4 SPACES}2{4 SPACES}3{2 SPACES}VO
     ICE"                                    :rem 243
9050 FORX=1TO3:A(X)=127:NEXT:R=1             :rem 191
10000 PRINT"{HOME}{14 DOWN}"A(1);A(2);A(3);R:GETA$
                                             :rem 145
10010 IFA$="{F1}"THENR=R+1:IFR=4THENR=1      :rem 211
10020 A(R)=A(R)-(A$="{RIGHT}")+(A$="{DOWN}")-10*(A
      $="{F3}")+10*(A$="{F5}")               :rem 213
10030 IFA(R)<127ORA$="{F7}"THENA(R)=127       :rem 54
10040 IFA(R)>255THENA(R)=255                  :rem 46
10050 IFA$=CHR$(13)THEN11000                 :rem 254
10060 FORX=1TO3:POKE36873+X,A(X):NEXT:GOTO10000
                                             :rem 100
11000 PRINT"{2 DOWN}SAVE CHORD? (Y/N)"       :rem 234
11010 GETA$:IFA$=""THEN11010                   :rem 9
11020 IFA$<>"Y"THEN11080                      :rem 40
11030 POKE198,0:FORX=1TO50:NEXT:PRINT"WHAT KEY?"
                                             :rem 241
11040 R=PEEK(197):GETA$:IFA%(R+64)THEN11040
                                             :rem 167
11050 R=R+64:N$(R)="{RVS}/S"+A$+"{OFF}":PRINT"CHOR
      D NAME:"N$(R)                           :rem 77
11060 A%(R)=A(1):B%(R)=A(2):C%(R)=A(3)       :rem 141
11070 PRINT"CHORD NUMBER:"R:PRINT"{DOWN}HAVE FUN W
      ITH YOUR NEWCHORD.{DOWN}"              :rem 220
11080 POKE198,0:FORX=1TO500:NEXT:GOTO203     :rem 199
```

Music Writer

Robert D. Heidler

This flexible music-composition utility works on the unexpanded VIC-20. With it, you can compose and play songs, edit your music, and add the tunes to your own programs.

Music can be a welcome addition to a computer program, particularly if the program is educational or recreational in nature. (Who wouldn't like to have the theme from *Close Encounters of the Third Kind* playing softly in the background as your flying saucer glides across the screen?) Unfortunately, adding music to a program can be a long and complex task that many new programmers hesitate to attempt. That's where "Music Writer" comes in.

Music Writer is a program designed to make composing at your VIC keyboard *easy*. Here are some of its features:

1. Music Writer allows you to easily enter any combination of notes from a two-octave range, and to hold each note for any duration.
2. It allows you to hear each note played as you enter it.
3. It allows you to hear your entire song played back at anytime while you are composing.
4. It allows you to insert, delete, or change notes anywhere in the song at any time.
5. When your song is complete, Music Writer will display the data necessary to reproduce your song in a program.

With this brief overview of the program's capabilities, let's explore in detail how to use Music Writer.

Entering Notes

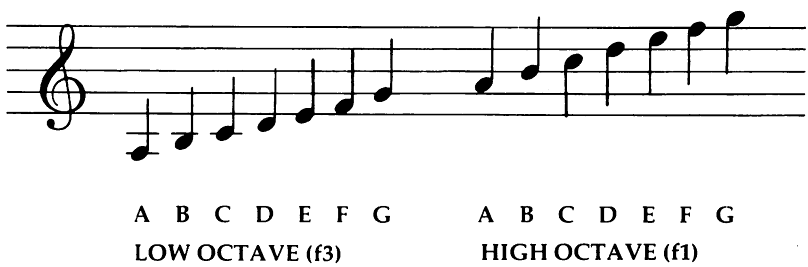
When you run the program, the words PLEASE ENTER NOTES appear at the top of the screen. To enter a note, simply press any valid note key from A through G. The VIC will respond by sounding the corresponding note and displaying its letter name in blue.

To move to a higher octave, press the f1 special function key. Now, pressing any key A through G plays the note one octave higher. The note's name is printed on the screen in red. To return to the lower octave, press f3.

This program requires you to convert all flats to sharps (B-flat becomes A-sharp, and so on). To play a sharp note, hold down the SHIFT key while pressing the key of the desired note. For example, to enter a C-sharp, hold down SHIFT and hit the C. The VIC will play a C-sharp note and the symbols C# are displayed on the screen.

If you aren't sure what note to use, press f5. This puts you in the search mode. You can now strike any combination of keys in either octave. The notes will sound as before, but no new notes will be added to the screen. When you have found the combination of notes you want, press f7. This returns you to the write mode.

Figure 1. The Range of Notes Possible with Music Writer



Duration

A note's duration is determined by the number of times the note is entered. Normally, count each keystroke as one beat. Thus, pressing the C once plays and holds a C note for one beat. Pressing the C twice plays and holds the C for two beats. If you want to play two notes of the same pitch but don't want the VIC to run them together, enter the first note, press the up-arrow key, and enter the second. The up-arrow key places a momentary break between the notes—just long enough to distinguish between them.

Sometimes you'll want to count each keystroke as one-half beat and double the playback speed in your program. (This allows you to use eighth notes in a song written in 4/4 time, and so on.)

If you want to place a rest in your song, press R. The duration of rests is handled in the same way as the duration of notes.

Anytime you want the VIC to play back what you have written, press P. The computer plays your song, momentarily illuminating the symbol of each note as it is played.

Editing Your Song

To insert, delete, or change a note anywhere in your song, use the left-right cursor key to move the cursor back to where you wish to make the change. (While the cursor itself is invisible, the color of the notes will change as the cursor passes over them.) To change a note, simply position the cursor over the old note and press the key for the desired note. Insertions and deletions are made with the INST/DEL key.

To clear a song from the screen, press the left-arrow key at the upper-left corner of the keyboard and then press the S key.

Data Display

When your song is complete, press the left-arrow key. This clears the screen and displays the data necessary to reproduce the song in your program. Simply copy these numbers off the screen and include them in DATA statements in your own program.

To make your program play your song exactly as you have written it, use the following subroutine:

```
10 POKE 36878,15:READ A
20 FOR B=1 TO A:READ C:IF C=0 THENPOKE 36
   876,0:GOTO 40
30 POKE 36876,C:FOR D=1 TO 250:NEXTD
40 NEXT B:POKE 36876,0:POKE 36878,0:RETUR
   N
```

The value 250 in line 30 controls the playback speed. You can substitute your own number here. Try starting with 250 and then increasing or decreasing the tempo to suit your taste. If you want to synchronize any kind of graphics on the screen while the song is playing, you will want to decrease the value of 250 and place the instructions for the screen display between lines 30 and 40.

If you want to play a song several times in a program, you may want to include a RESTORE statement at the beginning of line 10.

Figure 2. Sample Songs for Music Writer

"Mary Had A Little Lamb"

E D C D E ↑ E ↑ E E D ↑ D ↑ D D E
G ↑ G G E D C D E ↑ E ↑ E ↑ E D ↑
D E D C C P

"London Bridge"

f1 A B A f3 G F# G f1 A R f3 E F# G R F#
G f1 A R A B A f3 G F# G f1 A R f3 E E
f1 ↑ A A f3 F# D P

Typing the Program

When you are typing in Music Writer, leave out line 5 until you have tested your program to be sure you've typed it correctly. Line 5 disables the RUN/STOP key, preventing you from accidentally destroying your work. To exit the program without turning off the power, you must hit the left-arrow key.

Since this program uses a good deal of memory, it's a good idea to type it in without any spaces, apart from those within quotation marks.

Though this program was written primarily to aid in writing programs, it is also a lot of fun to play around with. It is very user-friendly, and the editing features let you change notes and durations to get different effects.

Music Writer

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

```
5 POKE808,100                                   :rem 193
50 DIMA%(209):PRINT"{CLR}":H%=1:POKE36878,15
                                                :rem 79
90 PRINT"{RED}{RVS}{UP}{24 SPACES}PLEASE ENTER NOT
   ES{24 SPACES}{OFF}{BLU}"                     :rem 107
100 FORL=1TO200:NEXT:POKE36876,0                :rem 179
102 B%=PEEK(197):IFB%=64THEN102                 :rem 231
103 IFB%=39THENPRINT"{RED}";:H%=2:GOTO100       :rem 164
104 IFB%=47THENPRINT"{BLU}";:H%=1:GOTO100       :rem 166
105 IFN>N%THENN%=N                              :rem 81
106 IFB%=55THEN600                              :rem 249
108 IFPEEK(653)=1THEN200                        :rem 107
112 IFB%=23THEN450                              :rem 244
120 IFB%=13THEN300                              :rem 236
130 IFB%=8THEN500                               :rem 195
140 IFB%=7THEN375                               :rem 205
```

```
145 IFN%=208THEN100                                :rem 51
150 IFB%=35THENN=N+1:A%(N)=191:PRINT"B ";:IFH%=2TH
    ENA%(N)=223                                    :rem 225
152 IFB%=34THENN=N+1:A%(N)=195:PRINT"C ";:IFH%=2TH
    ENA%(N)=225                                    :rem 233
156 IFB%=18THENN=N+1:A%(N)=201:PRINT"D ";:IFH%=2TH
    ENA%(N)=228                                    :rem 231
160 IFB%=49THENN=N+1:A%(N)=207:PRINT"E ";:IFH%=2TH
    ENA%(N)=231                                    :rem 231
162 IFB%=42THENN=N+1:A%(N)=209:PRINT"F ";:IFH%=2TH
    ENA%(N)=232                                    :rem 230
166 IFB%=19THENN=N+1:A%(N)=215:PRINT"G ";:IFH%=2TH
    ENA%(N)=235                                    :rem 239
170 IFB%=17THENN=N+1:A%(N)=183:PRINT"A ";:IFH%=2TH
    ENA%(N)=219                                    :rem 232
174 IFB%=10THENN=N+1:A%(N)=50:PRINT"{RVS}R{OFF} ";
                                                   :rem 177
176 IFB%=54THENN=N+1:A%(N)=0:PRINT"{RVS}↑{OFF} ";
                                                   :rem 146
178 POKE36876,A%(N)                                :rem 14
180 GOTO100                                        :rem 99
200 IFB%=23THEN425:REM *SHIFTED KEYS*              :rem 169
203 IFN%=208THEN100                                :rem 46
204 IFB%=7THEN400                                  :rem 195
205 IFB%=34THENN=N+1:A%(N)=199:PRINT"C#";:IFH%=2TH
    ENA%(N)=227                                    :rem 17
210 IFB%=18THENN=N+1:A%(N)=203:PRINT"D#";:IFH%=2TH
    ENA%(N)=229                                    :rem 4
215 IFB%=42THENN=N+1:A%(N)=212:PRINT"F#";:IFH%=2TH
    ENA%(N)=233                                    :rem 3
220 IFB%=19THENN=N+1:A%(N)=217:PRINT"G#";:IFH%=2TH
    ENA%(N)=236                                    :rem 12
225 IFB%=17THENN=N+1:A%(N)=187:PRINT"A#";:IFH%=2TH
    ENA%(N)=221                                    :rem 9
230 POKE36876,A%(N)                                :rem 3
250 GOTO100                                        :rem 97
300 FORL=1TON:REM *PLAYBACK*                       :rem 222
320 J=2*L+38486:J%=PEEK(J):POKEJ,5:POKEJ+1,5
                                                   :rem 38
330 IFA%(L)=0THENPOKE36876,0:GOTO350               :rem 104
340 POKE36876,A%(L):FORQ=1TO250:NEXTQ              :rem 231
350 POKEJ,J%:POKEJ+1,J%:NEXTL                      :rem 27
360 GOTO100                                        :rem 99
375 IFN=N%ANDN=0THEN100:REM *DELETE*               :rem 159
376 IFN=N%THENN=N-1:N%=N:PRINT"{2 LEFT}{2 SPACES}
    {2 LEFT}";:GOTO100                             :rem 137
380 FORT=N+1TON%                                   :rem 211
382 TP=2*T+7766:TC=2*T+38486                       :rem 41
384 T1%=PEEK(TP+2):T2%=PEEK(TP+3):POKETP,T1%:POKET
    P+1,T2%                                        :rem 186
```

```
386 T3%=PEEK(TC+2)AND7:POKETC,T3%:POKETC+1,T3%
                                              :rem 10
388 A%(T)=A%(T+1):NEXTT                       :rem 31
390 POKETP,32:POKETP+1,32:N%=N%-1            :rem 181
395 GOTO102                                  :rem 109
400 FORT=N%TON+1STEP-1:REM *INSERT*          :rem 173
405 TP=2*T+7766:TC=2*T+38486                  :rem 37
410 T1%=PEEK(TP):T2%=PEEK(TP+1):POKETP+2,T1%:POKET
    P+3,T2%                                  :rem 176
412 T3%=PEEK(TC)AND7:POKETC+2,T3%:POKETC+3,T3%
                                               :rem 2
414 A%(T+1)=A%(T):NEXTT                       :rem 21
416 N%=N%+1:POKETP,32:POKETP+1,32            :rem 178
420 GOTO102                                   :rem 98
425 IFN=0THEN100:REM *CURSOR LEFT*            :rem 32
427 N=N-1:Y=2*N+38488:Y%=PEEK(Y)AND7:IFY%=2THENPOK
    EY,7:POKEY+1,7                           :rem 167
428 IFY%=6THENPOKEY,5:POKEY+1,5              :rem 181
430 FORL=1TO100:NEXTL:PRINT"{2 LEFT}";:GOTO102
                                             :rem 183
450 Y=2*N+7768:IFPEEK(Y)=32THEN100:REM *CURSOR RIG
    HT*                                      :rem 168
452 Y=2*N+38488:Y%=PEEK(Y)AND7:IFY%=5THENPOKEY,6:P
    OKEY+1,6                                  :rem 53
453 IFY%=7THENPOKEY,2:POKEY+1,2              :rem 174
455 FORL=1TO100:NEXTL:PRINT"{2 RIGHT}";:N=N+1:GOTO
    102                                       :rem 45
500 PRINT"{CLR}{UP}DATA FOR SONG:";N;:REM *PRINT D
    ATA*                                     :rem 217
520 FORL=1TON:PRINT"{LEFT},";A%(L);          :rem 118
525 IFL=70ORL=144THENPRINT:PRINT"{RVS}{3 SPACES}PR
    ESS ANY KEY TO{6 SPACES}CONTINUE{11 SPACES}
    {OFF}";                                  :rem 234
527 IFL=70ORL=144THENM=PEEK(197):IFM=64THEN527
                                              :rem 58
530 NEXTL                                     :rem 35
535 PRINT:PRINT"{RVS}{RED}PRESS D TO REPEAT DATAPR
    ESS S TO START AGAINPRESS X TO STOP{7 SPACES}
    {BLU}{OFF}"                              :rem 162
540 LL=PEEK(197):IFLL=41THENRUN              :rem 172
545 IFLL=18THEN500                            :rem 47
547 IFLL<>26THEN540                          :rem 113
550 POKE198,0:POKE808,112:STOP                :rem 16
600 FORL=1TO200:NEXT:POKE36876,0:REM *SEARCH MODE*
                                               :rem 5
602 B%=PEEK(197):IFB%=64THEN602              :rem 241
603 IFB%=39THENPRINT"{RED}";:H%=2:GOTO600 :rem 174
604 IFB%=47THENPRINT"{BLU}";:H%=1:GOTO600 :rem 176
606 IFB%=63THEN100                           :rem 248
```

```
608 IFPEEK(653)=1THEN650                         :rem 121
610 IFB%=35THENSE%=191:IFH%=2THENSE%=223          :rem 91
612 IFB%=34THENSE%=195:IFH%=2THENSE%=225          :rem 98
614 IFB%=18THENSE%=201:IFH%=2THENSE%=228          :rem 93
616 IFB%=49THENSE%=207:IFH%=2THENSE%=231          :rem 99
618 IFB%=42THENSE%=209:IFH%=2THENSE%=232          :rem 97
620 IFB%=19THENSE%=215:IFH%=2THENSE%=235          :rem 94
622 IFB%=17THENSE%=183:IFH%=2THENSE%=219         :rem 100
624 POKE36876,SE%:GOTO600                        :rem 203
650 IFB%=34THENSE%=199:IFH%=2THENSE%=227         :rem 106
652 IFB%=18THENSE%=203:IFH%=2THENSE%=229          :rem 98
654 IFB%=42THENSE%=212:IFH%=2THENSE%=233          :rem 92
656 IFB%=19THENSE%=217:IFH%=2THENSE%=236         :rem 106
658 IFB%=17THENSE%=187:IFH%=2THENSE%=221         :rem 106
660 POKE36876,SE%:GOTO600                        :rem 203
```

Chapter 6

Utilities

Quickfind

Harvey B. Herman

If you use tape, you'll love "Quickfind." It lets you locate and load programs off cassette tape in a snap.

"Quickfind" permits programs to be loaded after the tape has been positioned by Fast Forward *under computer control.* It was originally written for the Commodore PET back before floppy disk drives were available. The Datassette recorders made then did not even have tape counters, so finding programs on a cassette tape was a time-consuming task.

Although today's Datassettes have digital tape counters, Quickfind is still a useful utility. It works by storing a directory as the first program on each tape. This directory calculates the proper amount of time to Fast Forward to locate each file. Quickfind is self-prompting and easy to use, even for beginners.

How to Prepare a Tape

1. Load a copy of Quickfind into the computer and change the program names in line 350 to those of your own programs. A *filename,* as it's called, may be up to 16 characters long (including spaces). *Do not* remove the word DATA from line 350. The first word on this line must be DATA so the computer knows that the subsequent characters are data to be read. The VIC accepts only 87 characters per program line, so if you run out of room, start a new line 360 with DATA as the first word and continue entering your filenames.

2. At line 140, set the variable N equal to the number of programs you are storing on that side of the cassette (in other words, the number of program names you included in line 350). For instance, if you are recording six programs, change line 140 so N=6. This tells the computer to expect six programs on that side of the tape.

3. Now SAVE your modified version of Quickfind as the first program on a new tape. *Do not rewind.* Remove this tape and insert the cassette from which you want to load your first program.

4. LOAD the program into the computer. Remove the cassette.

5. Insert the Quickfind tape and SAVE the program. It should now be recorded just following the Quickfind program itself on the new tape.

6. Rewind the tape, and LOAD and RUN Quickfind. Select the next program and let the computer fast forward to the proper place. *Do not rewind.* Remove the Quickfind cassette, and again, insert the tape from which you want to load the next program.

7. LOAD the next program.

8. Put back the Quickfind tape and SAVE.

9. Repeat steps 6 through 8 as many times as necessary. (This will depend on how many programs you are storing on that side of the cassette.)

This procedure is easier than it appears and will become second nature if you do it often.

How Quickfind Works

You can skip this part if you want. You already know everything you need to use Quickfind.

But for those who are interested, Quickfind works because the Commodore Datassette is more sophisticated than it might appear at first glance. The computer can control its drive motor and detect if a switch is pressed. It cannot differentiate, however, between the press of Fast Forward or Play. That's why, after running Quickfind and selecting your program, you are prompted to press the right buttons. Here are the steps in that sequence:

1. Is a button pressed? If yes, prompt for release and wait until no. If no, continue.

2. First program? If yes, skip ahead to step 8. If no, continue.

3. Prompt for press of Fast Forward.

4. Fast Forward pressed? If yes, continue. If no, wait until yes.

5. Turn off Datassette motor when time is up.

6. Prompt for release.

7. Fast Forward released? If yes, continue. If no, wait until yes.

8. LOAD program, using dynamic keyboard technique.

The programs are spaced six seconds apart in this version of Quickfind (see line 280). Time is kept by the built-in jiffy clock. (A *jiffy* is a sixtieth of a second.) The variable TI always contains the value of this clock.

Dynamic keyboard is a technique for loading programs from within another program. It is similar to the trick the computer uses when you press the SHIFT and RUN/STOP keys and get an automatic LOAD and RUN. If you want Quickfind to do automatic LOADs and RUNs, you can change the 13 in line 340 to a 131.

Don't be discouraged if you fail to understand the technical details. We are all in that boat at some point. Quickfind can be used even if you don't understand all the technical details.

Quickfind

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

```
140 N=5:DIM A$(N):REM N IS # OF PROGRAMS ON TAPE
                                            :rem 95
150 FOR I=1TO N: READ A$(I):NEXT I          :rem 60
160 PRINT CHR$(147);CHR$(144);"{2 SPACES}PROGRAM":
    PRINT "NUMBER/NAME":PRINT              :rem 186
170 FOR I=1 TO N:PRINT CHR$(157);I;"{2 SPACES}";A$
    (I):NEXT I:PRINT                       :rem 163
180 INPUT "FIND NUMBER";J:PRINT            :rem 163
190 IF J<1 OR J>N THEN 160                  :rem 29
200 IF J=1 THEN 330                        :rem 158
210 REM START OF FAST FORWARD ROUTINE        :rem 3
220 REM WAIT FOR RELEASE IF NECESSARY      :rem 209
230 IF (PEEK(37151) AND 64)=0 THEN PRINT "PRESS ST
    OP ON CASSETTE"                         :rem 98
240 IF (PEEK(37151) AND 64)=0 THEN 240      :rem 92
250 PRINT "PRESS FAST FORWARD":PRINT       :rem 255
260 IF(PEEK(37151) AND 64)=64 THEN 260:REM CHECK F
    OR PRESS                               :rem 138
270 PRINT "OK":PRINT:A=TI                   :rem 32
280 IF ABS(TI-A)<(J-1)*360 THEN 280:REM FAST FORWA
    RD 6 SEC PER PROGRAM                   :rem 141
290 POKE 37148,PEEK(37148) AND 247:REM STOP MOTOR
                                           :rem 223
300 PRINT "RELEASE FAST FORWARD"           :rem 168
310 IF (PEEK(37151) AND 64)=0 THEN 310:REM WAIT FO
    R RELEASE                              :rem 147
320 REM DYNAMIC KEYBOARD LOAD              :rem 239
330 PRINT CHR$(147);CHR$(17);CHR$(17);CHR$(17);"LO
    AD ";CHR$(34);A$(J);CHR$(34);CHR$(19) :rem 169
340 POKE 198,1:POKE 631,13:END             :rem 105
350 DATA PROGRAM1,PROGRAM2,PROGRAM3,PROGRAM4,PROGR
    AM5                                    :rem 217
```

Automatic Appending

Mark Niggemann

It's quite simple to combine two programs to make a single, larger program. This brief tutorial shows how and explains how the VIC automatically relocates programs in memory.

One of the nice features of the VIC is the auto-relocation of BASIC programs during a LOAD. Depending on its memory size, a VIC can have as many as three different locations to store programs in RAM, but the VIC has the capacity to automatically put a program in the correct place. If you saved a program on a 3K VIC and later on bought a 3K expander, it would be next to impossible to RUN that program if the relocator didn't make an adjustment.

BASIC on a 3.5K machine expects the starting memory address to be 4097. All programs are saved with this memory address as their starting point. On a VIC with 3K expansion, the starting memory address is 1025. So since the starting point of BASIC can vary, it's left up to the relocator to set things right.

How the Relocator Works

The relocator first checks to see where the start of BASIC is. This is an address POKEd by the computer into locations 43 and 44 when the VIC is switched on. This start-of-BASIC address is where the relocator will begin to store any program that the VIC is loading. (*Note:* This does not include programs that are saved using absolute save mode, as in Jim Butterfield's "Tinymon.")

Since the relocator depends on the start-of-BASIC memory locations (called *pointers*) to know where to start storing a program during a LOAD, it is possible to join two separate programs by using a method that I will describe later. Note that the two programs to be joined must not have overlapping line numbers, and that the program in memory at the time must have lower line numbers than the program you are *appending* onto it from tape.

Type in this example program:

```
50 REM PART 2 OF TEST PRG.
60 PRINT "THIS IS A TEST"
70 PRINT "TO SEE A VIC"
80 PRINT "APPENDING!"
```

Now save this example on tape and clear the memory using NEW to make way for the next program:

```
SAVE"PART 2"
PRESS PLAY & RECORD ON TAPE
OK
SAVING PART 2
READY.
NEW
```

Now type in this example program:

```
10 REM PART 1 OF TEST PRG.
20 PRINT "WILL THIS WORK?"
30 PRINT "I HOPE IT DOES."
40 PRINT "I KNOW IT WILL!"
```

I had you type in the second part first so that part 1, the program we are appending, is in memory, and part 2 is on tape.

Clear the screen and type the following in direct mode:

```
PRINT PEEK(43),PEEK(44)
```

On a 3.5K machine you should get 1 and 16, respectively. Write down these printed values because you're going to need them again later on.

Now type in the following in direct mode:

```
POKE 43,PEEK(45)-2:POKE 44,PEEK(46)
LOAD"PART 2"
```

The above lines typed in direct mode set the start of BASIC to the end of the current program already in memory. Then you load part 2 as you would any other program. The key to the whole thing is that the relocator will use as its starting location the start of BASIC, which is directed by locations 43 and 44.

There is one final step before the two programs are finally appended. You must reset the start of BASIC to what it was before you loaded in part 2. To do this, you simply POKE the two values that you previously PEEKed into their respective memory locations.

For a 3.5K machine this step would look like the following:

POKE 43,1:POKE 44,16

If you made no error in the procedure outlined above, you should be able to list the whole program with both parts together. Appending can be a very powerful programming aid, since it allows you to develop several sets of subroutines and then later on lets you patch them into a main program at will.

Why not create a library of frequently used subroutines—for instance, to shuffle cards, round off numbers, accept INPUT in a special way, or to do anything you use in lots of different programs? Simply use high line numbers for the subroutines (50000 and up) and then use this technique to add them painlessly to your main programs.

Ultrasort

John W. Ross

This is probably the fastest sorting program ever published for any home computer. It will alphabetize 1000 items in less than nine seconds. Requires 8K or more expansion.

This article is a sequel to my earlier article "Super Shell Sort for Pet/CBM" (*COMPUTE!*, February 1983). In that article, I described a shell sort program for the CBM 8032 written entirely in machine language. It performed as expected and was very fast; however, it had a couple of shortcomings. First of all, it had a rather clumsy interface with BASIC; that is, the calling sequence was not very neat; and second, sorting was performed by the shell sort algorithm. This method of sorting is actually quite efficient, certainly far better than a bubble sort, for instance. Nevertheless, there are better sorts.

C.A.R. Hoare's Quicksort algorithm is possibly the fastest yet developed for most applications. So I rewrote this machine language sort program, "Ultrasort," based on the Quicksort algorithm.

Speed Improvements

How much better is it? In order to test the program, I wrote a small sort test program (Program 2), similar to the one in my original article. This program generates a character array containing N items (line 110).

Different items are generated depending on the value of the random number seed, SD, in line 140. SD must be a negative number.

I generated six 1000 element arrays and sorted them using both the shell sort and Ultrasort. Super Shell Sort required an average of 29.60 seconds to sort all 1000 elements, while Ultrasort required an average of only 8.32 seconds. The sorting time decreased 72 percent. I don't believe you will find a faster sort for an eight-bit machine.

The loader for the VIC version (Program 1) is designed to relocate itself to the top of available memory, which will vary according to the amount of expansion memory added to your VIC. (Ultrasort is too long for the unexpanded VIC.) The loader program will tell you the proper SYS address to use on your VIC.

Running the Program

Ultrasort can be used either from within a program or in immediate mode. Running Ultrasort causes N elements from array AA$, starting with element K, to be sorted into ascending order. The sort occurs in place; there is no additional memory overhead. N and K can be constants or variables, and any character array name can be substituted for AA$.

Before running the sort, though, it must be LOADed by BASIC. The appropriate loader is supplied in Program 1. The tradeoff for the increased speed of Ultrasort is increased complexity, especially in machine language, so check your typing carefully.

Notice that the first thing the loader program does is to reset the top-of-memory pointer. This is very important—you must use the BASIC loader before running the sort program.

Once Program 1 is loaded, you might wish to save it using a monitor program or cartridge (for example, VICMON or Supermon 64).

If you save Ultrasort using a monitor, you can reload it by typing:

LOAD"ULTRASORT",8,1　　for disk, or
LOAD"ULTRASORT",1,1　　for tape.

If you do not have a monitor, you can still use Ultrasort by first loading and running the BASIC loader and then loading a second program that uses Ultrasort. The test program is an example of how you can use Ultrasort. Be sure to adjust the SYS in line 300 to your memory. The loader will tell you the correct memory location to SYS to.

Program 1. Ultrasort: BASIC Loader

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

```
5 I1=PEEK(56)*256-1024                             :rem 136
6 POKE 55,0:HI=INT(I1/256):POKE 56,HI:CLR           :rem 97
7 I1=PEEK(55)+PEEK(56)*256                         :rem 161
8 HI=INT(I1/256)                                   :rem 136
10 I=I1                                             :rem 97
20 READ A:IF A=256 THEN PRINT"TO RUN SORT, USE:SYS
   "I1:END                                          :rem 96
22 IF A<0 THEN A=ABS(A)-26+HI                        :rem 6
25 IF A=257 THEN I=I1+100:GOTO 20                  :rem 210
```

```
30 POKE I,A:I=I+1:GOTO 20                          :rem 130
6656 DATA 76,100,-26,257                           :rem 166
6768 DATA 32,253,206,32,158                         :rem 63
6776 DATA 205,32,247,215,165,20,141                :rem 193
6784 DATA 12,-26,165,21,141,13,-26                 :rem 129
6792 DATA 32,253,206,32,158,205,56                 :rem 150
6800 DATA 165,71,233,3,133,75,165                   :rem 94
6808 DATA 72,233,0,133,76,162,1                    :rem 247
6816 DATA 173,12,-26,157,20,-26,173                :rem 185
6824 DATA 13,-26,157,40,-26,169,1                   :rem 86
6832 DATA 157,60,-26,169,0,157,80                   :rem 98
6840 DATA -26,189,60,-26,141,16,-26                :rem 184
6848 DATA 189,80,-26,141,17,-26,189                :rem 208
6856 DATA 20,-26,141,18,-26,189,40                 :rem 140
6864 DATA -26,141,19,-26,32,47,-29                 :rem 140
6872 DATA 173,11,-26,48,4,202,208                   :rem 89
6880 DATA 221,96,189,60,-26,141,16                 :rem 150
6888 DATA -26,189,80,-26,141,17,-26                :rem 199
6896 DATA 169,1,141,18,-26,169,0                    :rem 52
6904 DATA 141,19,-26,32,101,-29,189                :rem 187
6912 DATA 20,-26,141,18,-26,141,14                 :rem 122
6920 DATA -26,189,40,-26,141,19,-26                :rem 184
6928 DATA 141,15,-26,32,47,-29,173                 :rem 143
6936 DATA 11,-26,48,3,76,167,-27                    :rem 49
6944 DATA 32,131,-29,173,16,-26,141                :rem 184
6952 DATA 3,-26,173,17,-26,141,4                    :rem 34
6960 DATA -26,173,14,-26,141,5,-26                 :rem 129
6968 DATA 173,15,-26,141,6,-26,32                   :rem 91
6976 DATA 132,-28,32,180,-28,173,11                :rem 189
6984 DATA -26,48,218,173,16,-26,141                :rem 198
6992 DATA 3,-26,173,17,-26,141,4                    :rem 38
7000 DATA -26,173,18,-26,141,16,-26                :rem 169
7008 DATA 173,19,-26,141,17,-26,169                :rem 190
7016 DATA 1,141,18,-26,169,0,141                    :rem 27
7024 DATA 19,-26,32,101,-29,173,16                 :rem 127
7032 DATA -26,141,18,-26,173,17,-26                :rem 175
7040 DATA 141,19,-26,173,3,-26,141                 :rem 123
7048 DATA 16,-26,173,4,-26,141,17                   :rem 83
7056 DATA -26,32,47,-29,173,11,-26                 :rem 131
7064 DATA 16,35,173,14,-26,141,3                    :rem 32
7072 DATA -26,173,15,-26,141,4,-26                 :rem 124
7080 DATA 173,18,-26,141,5,-26,173                 :rem 133
7088 DATA 19,-26,141,6,-26,32,132                   :rem 84
7096 DATA -28,32,180,-28,173,11,-26                :rem 182
7104 DATA 48,152,32,47,-29,173,11                   :rem 87
7112 DATA -26,16,18,173,16,-26,141                 :rem 127
7120 DATA 3,-26,173,17,-26,141,4                    :rem 22
7128 DATA -26,32,132,-28,32,31,-29                 :rem 124
7136 DATA 76,241,-26,234,189,20,-26                :rem 190
7144 DATA 141,3,-26,189,40,-26,141                 :rem 129
```

```
7152 DATA 4,-26,173,16,-26,141,5          :rem 28
7160 DATA -26,173,17,-26,141,6,-26        :rem 126
7168 DATA 32,132,-28,32,31,-29,173        :rem 134
7176 DATA 16,-26,141,18,-26,141,3         :rem 80
7184 DATA -26,173,17,-26,141,19,-26       :rem 184
7192 DATA 141,4,-26,32,81,-29,189         :rem 92
7200 DATA 20,-26,141,18,-26,189,40        :rem 124
7208 DATA -26,141,19,-26,32,101,-29       :rem 172
7216 DATA 173,11,-26,48,15,189,60         :rem 94
7224 DATA -26,141,18,-26,189,80,-26       :rem 185
7232 DATA 141,19,-26,32,101,-29,169       :rem 180
7240 DATA 1,141,18,-26,169,0,141          :rem 26
7248 DATA 19,-26,173,3,-26,141,16         :rem 86
7256 DATA -26,173,4,-26,141,17,-26        :rem 130
7264 DATA 173,11,-26,16,52,189,60         :rem 93
7272 DATA -26,232,157,60,-26,202,189      :rem 238
7280 DATA 80,-26,232,157,80,-26,32        :rem 134
7288 DATA 101,-29,173,16,-26,157,20       :rem 187
7296 DATA -26,173,17,-26,157,40,-26       :rem 189
7304 DATA 32,131,-29,32,131,-29,202       :rem 168
7312 DATA 173,16,-26,157,60,-26,173       :rem 185
7320 DATA 17,-26,157,80,-26,76,128        :rem 141
7328 DATA -28,32,131,-29,232,173,16       :rem 184
7336 DATA -26,157,60,-26,173,17,-26       :rem 186
7344 DATA 157,80,-26,202,189,20,-26       :rem 187
7352 DATA 232,157,20,-26,202,189,40       :rem 184
7360 DATA -26,232,157,40,-26,202,32       :rem 173
7368 DATA 101,-29,32,101,-29,173,16       :rem 181
7376 DATA -26,157,20,-26,173,17,-26       :rem 186
7384 DATA 157,40,-26,232,76,162,-26       :rem 192
7392 DATA 160,3,165,75,133,79,133         :rem 103
7400 DATA 81,165,76,133,80,133,82         :rem 95
7408 DATA 24,165,79,109,3,-26,133         :rem 96
7416 DATA 79,165,80,109,4,-26,133         :rem 98
7424 DATA 80,24,165,81,109,5,-26          :rem 42
7432 DATA 133,81,165,82,109,6,-26         :rem 93
7440 DATA 133,82,136,208,223,96,160       :rem 195
7448 DATA 0,140,11,-26,177,79,141         :rem 90
7456 DATA 7,-26,177,81,141,8,-26          :rem 47
7464 DATA 200,152,205,7,-26,240,2         :rem 76
7472 DATA 176,13,205,8,-26,240,21         :rem 85
7480 DATA 144,19,238,11,-26,76,30         :rem 92
7488 DATA -29,205,8,-26,240,2,176         :rem 95
7496 DATA 62,206,11,-26,76,30,-29         :rem 91
7504 DATA 140,9,-26,160,1,177,79          :rem 44
7512 DATA 133,77,200,177,79,133,78        :rem 157
7520 DATA 172,9,-26,136,177,77,141        :rem 149
```

```
7528 DATA 10,-26,140,9,-26,160,1              :rem 25
7536 DATA 177,81,133,77,200,177,81           :rem 158
7544 DATA 133,78,172,9,-26,177,77            :rem 113
7552 DATA 200,205,10,-26,208,3,76             :rem 78
7560 DATA 195,-28,144,184,76,224,-28         :rem 252
7568 DATA 96,160,2,177,79,72,177              :rem 75
7576 DATA 81,145,79,104,145,81,136           :rem 160
7584 DATA 16,243,96,169,0,141,11              :rem 49
7592 DATA -26,173,17,-26,205,19,-26          :rem 188
7600 DATA 144,6,240,8,238,11,-26              :rem 32
7608 DATA 96,206,11,-26,96,173,16            :rem 102
7616 DATA -26,205,18,-26,144,244,208         :rem 237
7624 DATA 238,96,173,16,-26,24,109           :rem 152
7632 DATA 18,-26,141,16,-26,173,17           :rem 135
7640 DATA -26,109,19,-26,141,17,-26          :rem 180
7648 DATA 96,169,0,141,11,-26,56              :rem 50
7656 DATA 173,16,-26,237,18,-26,141          :rem 193
7664 DATA 16,-26,173,17,-26,237,19           :rem 147
7672 DATA -26,141,17,-26,176,3,206           :rem 137
7680 DATA 11,-26,96,238,16,-26,208           :rem 144
7688 DATA 3,238,17,-26,96,256                :rem 172
```

Program 2. Ultrasort: Test Sort

```
100 PRINT"{CLR}"                              :rem 245
110 N=100                                     :rem 174
120 DIM AA$(N)                                :rem 178
130 PRINT "CREATING"N" RANDOM STRINGS"         :rem 47
140 SD=-TI : A=RND(SD)                        :rem 183
150 FOR I=1TO N                                :rem 37
160 PRINT I"{UP}"                              :rem 66
170 N1=INT(RND(1)*10+1)                       :rem 221
180 A$=""                                     :rem 127
190 FOR J=1 TO N1                              :rem 91
200 B$=CHR$(INT(RND(1)*26+65))                 :rem 81
210 A$=A$+B$                                   :rem 43
220 NEXT J                                     :rem 29
230 AA$(I)=A$                                 :rem 119
240 NEXTI                                      :rem 30
250 PRINT "HIT ANY KEY TO START SORT"         :rem 151
260 GET A$:IF A$="" THEN 260                   :rem 83
270 PRINT "SORTING..."                         :rem 26
280 T1=TI                                     :rem 249
290 REM                                       :rem 127
292 REM USE SYS VALUE GENERATED BY THE LOADER FOR
    {SPACE}VIC                                :rem 117
300 SYS 15360,N,AA$(1)                         :rem 95
```

```
310 T2=TI                                          :rem 244
320 PRINT "DONE"                                   :rem 140
330 PRINT "HIT ANY KEY TO PRINT SORTED STRINGS"
                                                    :rem 72
340 GET A$:IF A$=""THEN340                          :rem 81
350 FORI=1TON:PRINT I,AA$(I):NEXT                   :rem 28
360 PRINT:PRINT N" ELEMENTS SORTED IN"(T2-T1)/60"S
    ECONDS"                                        :rem 181
```

Disk Menu

Wayne Mathews

"Load-and-go" disk menu programs are popular with users of all types of computers. Now there's a version for the VIC-20.

The VIC 1540/1541 disk drive is a great boon to computing. But when you have a large number of programs on a single disk, the task of loading and listing the directory to find a certain program leaves much to be desired. I originally started this program to create a neater directory display, and then added the autoload and autorun features. The program, "Disk Menu," is self-explanatory and easy to use.

How It Works

The "directory peeper" (starting at line 150) opens the $ (directory) file and then reads character by character until it comes to quotes. The first word enclosed in quotes in the directory is the name of the disk, and each subsequent name is that of a file. These are collected in the P$ array.

To load or load and run programs, first the LOAD statement is printed on the screen. Then the cursor is moved up three lines so that it falls on the line of the LOAD statement after the program ends. Line 510 shortens the file name so that the load statement fits on one line if the name is long. The rest of the program automatically executes a RETURN (for the load option), or a RETURN–RUN–RETURN (for load and run). The program then ends, the keyboard buffer is emptied, and the program is loaded. If load and run were selected, the keyboard buffer continues to empty, and the program is run.

The Disk Menu program does not discriminate between programs and other types of files. For my own use, this is an advantage because I can examine the names of data files with the menu program. If you don't want data files listed, the directory peeper can be modified so that byte 0 of each file entry is examined to determine the file type, and only the program files will be loaded into the P$ array.

The current menu will hold 100 programs. If this number is not suitable, you should change the value 100 in lines 140 and 330.

The autorun can also be used to chain programs, with one program loading the next, or loading a menu which ties together several related programs.

If the menu program is saved as the first program on the disk, it can be loaded using the statement LOAD "*",8.

Disk Menu

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

```
100 REM VIC DISK MENU                              :rem 183
120 CLR                                            :rem 116
130 PRINT"{CLR}{4 DOWN}{GRN}READING DISK DIRECTORY
    {BLU}"                                          :rem 83
140 DIMP$(100)                                     :rem 197
150 REM DIRECTORY                                   :rem 47
160 REM{2 SPACES}PEEPER                             :rem 60
170 OPEN1,8,0,"$"                                   :rem 79
180 REM LOAD ARRAY P$                              :rem 144
190 GET#1,B$:IFST<>0THEN270                         :rem 66
200 IFB$<>CHR$(34)THEN190                          :rem 131
210 P$=""                                          :rem 136
220 GET#1,B$:IFB$<>CHR$(34)THENP$=P$+B$:GOTO220
                                                   :rem 168
230 GET#1,B$:IFB$=CHR$(32)THEN230                   :rem 66
240 P$(X)=P$:PRINTX;"{UP}"                         :rem 147
250 GET#1,B$:IFB$<>""THEN250                        :rem 16
260 IFST=0THENX=X+1:GOTO180                        :rem 191
270 CLOSE1                                          :rem 64
280 REM{2 SPACES}DISPLAY MENU                      :rem 201
290 N=10                                           :rem 135
300 PRINT"{CLR}{RED}VIC DISK MENU"                  :rem 85
310 PRINT"FOR ";P$(0);"{BLU}":PRINT                :rem 225
320 IFN<10THENN=10                                  :rem 42
330 IFN>100THENN=100                               :rem 141
340 FORJ=(N-9)TON:PRINTJ;"- ";P$(J):NEXTJ          :rem 197
350 PRINT:PRINT"PRESS{DOWN}{5 LEFT}{RED}N{BLU} FOR
     NEXT SCREEN"                                   :rem 78
360 PRINT"{RED}L{BLU} FOR LAST SCREEN"             :rem 204
370 PRINT"{RED}E{BLU} TO EXIT TO BASIC"            :rem 205
380 PRINT"{PUR}S{BLU} TO LOAD ONLY"                :rem 127
390 PRINT"{GRN}R{BLU} TO LOAD AND RUN"             :rem 135
400 Z$="":GETZ$:IFZ$=""THEN400                     :rem 182
410 IFZ$="N"THENN=N+10:GOTO300                      :rem 11
420 IFZ$="L"THENN=N-10:GOTO300                      :rem 12
430 IFZ$="E"THENPRINT"MENU STILL PRESENT":END
                                                    :rem 89
```

```
440 IFZ$="S"THENGOSUB470:GOTO570                  :rem 210
450 IFZ$="R"THENGOSUB470:GOTO540                  :rem 207
460 GOTO400                                       :rem 103
470 REM SELECT AND{12 SPACES}PRINT LOAD           :rem 191
480 PRINT"{DOWN}PROGRAM #";                       :rem 244
490 INPUTS:PRINT"{17 DOWN}"                       :rem 172
500 IFS<1ORS>XTHEN480                              :rem 57
510 IFLEN(P$(S))>12THENP$(S)=LEFT$(P$(S),12)+"*"
                                                  :rem 114
520 PRINT"LOAD"+CHR$(34)+P$(S)+CHR$(34)+",8{3 UP}"
                                                   :rem 25
530 RETURN                                        :rem 120
540 REM LOAD AND RUN                              :rem 101
550 POKE631,13:POKE632,82:POKE633,85:POKE634,78:PO
    KE635,13:POKE198,5                            :rem 207
560 END                                           :rem 114
570 REM LOAD ONLY                                 :rem 226
580 POKE631,13:POKE198,1                           :rem 94
```

UNNEW

Vern Buis

If you have ever lost a BASIC program by accidentally typing NEW, read on. This short machine language routine for the VIC-20 (any memory size) 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—you type NEW to clear out the memory. You *think* your program has been saved. But a split second after pressing RETURN, you realize that you've just lost it.

But on the VIC-20, 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 "VIC 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

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 VIC 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 VIC with any memory configuration and to minimize the danger of it loading atop a BASIC program.

Program Lifesaver

For mistake-proof program entry, be sure to use "The 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
```

Tiny Aid

David A. Hook

Add five invaluable commands to BASIC: renumber, delete, find, change, and kill. This enhancement to BASIC automatically locates itself, protects itself, and becomes part of the computer. It requires 1200 bytes of RAM, a little more than 1K. "Tiny Aid" will run on any VIC, but requires the use of the MLX program (Appendix K) and at least 8K memory expansion to enter.

Since the early days of the PET, various enhancements for BASIC have been available. Bill Seiler, then of Commodore, produced the first public-domain version, called "BASIC-Aid."

Many updates and improvements have been made over the past couple of years. The PET/CBM program has ballooned to a 4K package for almost every possible PET/CBM equipment configuration.

As has been customary in the Commodore community, Jim Butterfield developed a version of BASIC-Aid. He called this "Tinyaid2" (or "Tinyaid4," for BASIC 4.0). This offered the six most useful commands from the full-fledged program.

Following is my modification of that work, designed to provide VIC users with the same benefits. After using this for a while, I think you will find the added commands nearly indispensable.

Features

"Tiny Aid" is a machine language program which consumes about 1200 bytes of RAM memory. After you have loaded the program, type RUN and hit RETURN. The program repacks itself into high memory. The appropriate pointers are set so that BASIC will not clobber it. Tiny Aid is now alive.

Once activated, five commands become attached to BASIC. They will function only in direct mode, so don't include them in a program.

(1) NUMBER 1000,5
NUMBER 100,10

Renumbers a BASIC program with a given starting line number and a given increment between line numbers. The maximum increment is 255.

All references after GOTO, THEN, GOSUB, and RUN are automatically corrected. These lines are displayed onscreen as the command works. If a GOTO refers to a nonexistent line number, it is changed to 65535. This is an illegal line number, and must be corrected before the BASIC program is used.

(2)
```
DELETE 100-200
DELETE-1500
DELETE 5199-
```

Deletes a range of lines from a BASIC program. Uses the same syntax as the LIST command, so any line range may be specified for removal. DELETE with no range will perform like a NEW command, so be careful.

(3)
```
FIND/PRINT/
FIND/A$/,150-670
FIND "PRINT",20000-
```

Will locate any occurrences of the characters between the / marks. Almost any character may mark the start/end of the string to be found, so long as both are the same. The first example will find all the PRINT instructions in the program.

If you are looking for a string of text which contains a BASIC keyword, you must use the quote characters as markers. This will prevent the search string from being tokenized.

If a limited line-range is desired, use the same syntax as for LIST. Note that a comma must separate the line-range from the end marker.

All lines containing the string are printed to the screen. If a line has more than one of them, each occurrence will cause a repetition of that line.

(4)
```
CHANGE-PRINT-PRINT#4,-
CHANGE/ABC/XYZ/,6000-
CHANGE/DS$/D1$/,-5000
```

Using the same syntax as FIND, you may change any string to any other string in a BASIC program. This command is very powerful and was not part of the early versions of BASIC-Aid or "Toolkit."

As before, you may indicate a line-range. As the changes are made, the revised lines are displayed on the screen.

Watch out for the difference between BASIC keywords and strings of text within quotes. You may use the quote characters to differentiate, as with FIND.

(5) KILL

This command disables Tiny Aid and its associated commands. A syntax error will be the result if any of the above commands are now tried.

Since the routine is safe from interference from BASIC, you may leave it active for as long as your machine stays on. It is possible that Tiny Aid may interfere with other programs that modify BASIC's internal CHRGOT routine. The KILL command allows you to avoid this conflict.

Entering Tiny Aid

Tiny Aid will run on any VIC with any amount of memory, but requires the use of "The Machine Language Editor" (Appendix K) for error-proof entry. The Machine Language Editor (MLX) requires a VIC with at least 8K memory expansion. You should read the article about the MLX program, and you must type it in and save it before entering Tiny Aid.

Once you have a working copy of MLX, entering Tiny Aid is simple. Reset your VIC by turning it off and then on again. Type in the following line in direct mode before loading the MLX program:

POKE 44,25:POKE 6400,0:NEW

Press RETURN. LOAD and RUN the MLX program. MLX will ask for the starting and ending addresses. They are:

Starting address: 4609
Ending address: 6326

Now start entering the data for Tiny Aid. If you want to type in Tiny Aid in more than one sitting, you can; but be sure to enter the line above before loading MLX each time you want to use MLX to enter Tiny Aid.

Once you have saved a complete copy of Tiny Aid to disk or tape, it will run on any VIC. Simply load Tiny Aid and type RUN.

Tiny Aid

```
4609 :029,018,001,000,153,034,236
4615 :147,032,032,032,032,018,044
4621 :032,086,073,067,032,084,131
4627 :073,078,089,032,065,073,173
4633 :068,032,034,000,058,018,235
4639 :002,000,153,034,017,032,013
```

```
4645 :032,065,068,065,080,084,175
4651 :069,068,032,070,079,082,187
4657 :032,086,073,067,032,066,149
4663 :089,058,000,083,018,003,050
4669 :000,153,034,032,032,032,088
4675 :032,032,068,065,086,073,167
4681 :068,032,065,046,032,072,132
4687 :079,079,075,000,112,018,186
4693 :004,000,153,034,017,032,069
4699 :032,070,082,079,077,032,207
4705 :039,084,073,078,089,032,236
4711 :065,073,068,039,032,066,190
4717 :089,058,000,138,018,005,161
4723 :000,153,034,032,032,032,142
4729 :032,074,073,077,032,066,219
4735 :085,084,084,069,082,070,089
4741 :073,069,076,068,000,167,074
4747 :018,006,000,153,034,017,111
4753 :032,032,065,078,068,032,196
4759 :039,066,065,083,073,067,032
4765 :032,065,073,068,039,032,210
4771 :066,089,058,000,191,018,073
4777 :007,000,153,034,032,032,171
4783 :032,032,032,032,066,073,186
4789 :076,076,032,083,069,073,078
4795 :076,069,082,000,217,018,137
4801 :008,000,153,034,017,018,167
4807 :083,065,077,080,076,069,137
4813 :032,032,067,079,077,077,057
4819 :065,078,068,083,058,000,051
4825 :244,018,009,000,153,034,163
4831 :017,067,072,065,078,071,081
4837 :069,032,047,063,047,080,055
4843 :082,073,078,084,035,052,127
4849 :044,047,000,013,019,010,118
4855 :000,153,034,070,073,078,143
4861 :068,032,046,071,079,083,120
4867 :085,066,046,044,032,050,070
4873 :048,048,045,000,034,019,203
4879 :011,000,153,034,068,069,094
4885 :076,069,084,069,032,049,144
4891 :051,048,045,054,050,053,072
4897 :000,053,019,012,000,153,014
4903 :034,078,085,077,066,069,192
4909 :082,032,049,048,048,044,092
4915 :053,000,081,019,013,000,217
4921 :153,034,075,073,076,076,032
4927 :032,032,032,032,032,032,255
4933 :032,032,040,086,073,067,143
```

```
4939 :Ø32,Ø65,Ø73,Ø68,Ø41,ØØØ,Ø98
4945 :1Ø8,Ø19,Ø14,ØØØ,158,Ø4Ø,164
4951 :194,Ø4Ø,Ø52,Ø51,Ø41,17Ø,123
4957 :194,Ø4Ø,Ø52,Ø52,Ø41,172,132
4963 :Ø5Ø,Ø53,Ø54,17Ø,Ø51,Ø56,Ø21
4969 :Ø51,Ø41,ØØØ,ØØØ,ØØØ,17Ø,111
4975 :17Ø,17Ø,17Ø,17Ø,17Ø,17Ø,1Ø7
4981 :17Ø,17Ø,17Ø,17Ø,17Ø,17Ø,113
4987 :17Ø,17Ø,17Ø,17Ø,17Ø,165,114
4993 :Ø45,133,Ø34,165,Ø46,133,173
4999 :Ø35,165,Ø55,133,Ø36,165,212
5ØØ5 :Ø56,133,Ø37,16Ø,ØØØ,165,18Ø
5Ø11 :Ø34,2Ø8,ØØ2,198,Ø35,198,Ø54
5Ø17 :Ø34,177,Ø34,2Ø8,Ø6Ø,165,Ø63
5Ø23 :Ø34,2Ø8,ØØ2,198,Ø35,198,Ø66
5Ø29 :Ø34,177,Ø34,24Ø,Ø33,133,Ø48
5Ø35 :Ø38,165,Ø34,2Ø8,ØØ2,198,Ø48
5Ø41 :Ø35,198,Ø34,177,Ø34,Ø24,167
5Ø47 :1Ø1,Ø36,17Ø,165,Ø38,1Ø1,Ø26
5Ø53 :Ø37,Ø72,165,Ø55,2Ø8,ØØ2,216
5Ø59 :198,Ø56,198,Ø55,1Ø4,145,183
5Ø65 :Ø55,138,Ø72,165,Ø55,2Ø8,126
5Ø71 :ØØ2,198,Ø56,198,Ø55,1Ø4,Ø52
5Ø77 :145,Ø55,Ø24,144,182,2Ø1,196
5Ø83 :223,2Ø8,237,165,Ø55,133,216
5Ø89 :Ø51,165,Ø56,133,Ø52,1Ø8,Ø22
5Ø95 :Ø55,ØØØ,17Ø,17Ø,17Ø,17Ø,198
51Ø1 :17Ø,17Ø,17Ø,17Ø,17Ø,17Ø,233
51Ø7 :17Ø,17Ø,17Ø,17Ø,17Ø,17Ø,239
5113 :17Ø,17Ø,17Ø,17Ø,17Ø,17Ø,245
5119 :17Ø,223,173,254,255,ØØØ,Ø5Ø
5125 :133,Ø55,173,255,255,ØØØ,1Ø8
5131 :133,Ø56,169,Ø76,133,124,19Ø
5137 :173,217,251,ØØØ,133,125,148
5143 :173,218,251,ØØØ,133,126,156
5149 :Ø76,143,252,ØØØ,24Ø,ØØ3,231
5155 :Ø76,ØØ8,2Ø7,169,2Ø1,133,Ø61
5161 :124,169,Ø58,133,125,169,Ø51
5167 :176,133,126,Ø96,219,251,Ø24
5173 :ØØØ,133,139,134,151,186,Ø28
5179 :189,ØØ1,ØØ1,2Ø1,14Ø,24Ø,Ø63
5185 :Ø16,2Ø8,ØØ2,164,14Ø,166,249
5191 :151,165,139,2Ø1,Ø58,176,193
5197 :ØØ3,Ø76,128,ØØØ,ØØØ,Ø96,124
52Ø3 :189,ØØ2,ØØ1,2Ø1,196,2Ø8,112
52Ø9 :237,165,139,Ø16,ØØ2,23Ø,11Ø
5215 :122,132,14Ø,162,ØØØ,ØØØ,139
5221 :134,165,2Ø2,232,164,122,Ø96
5227 :185,ØØØ,ØØØ,ØØ2,Ø56,253,Ø91
```

```
5233 :217,255,000,240,019,201,021
5239 :128,240,019,230,165,232,109
5245 :189,216,255,000,016,250,027
5251 :189,217,255,000,208,228,204
5257 :240,191,232,200,208,224,152
5263 :132,122,165,165,010,170,139
5269 :189,245,255,000,072,189,075
5275 :244,255,000,072,032,233,223
5281 :251,000,076,115,000,000,091
5287 :032,178,253,000,165,095,122
5293 :166,096,133,036,134,037,007
5299 :032,019,198,165,095,166,086
5305 :096,144,010,160,001,177,005
5311 :095,240,004,170,136,177,245
5317 :095,133,122,134,123,165,201
5323 :036,056,229,122,170,165,213
5329 :037,229,123,168,176,030,204
5335 :138,024,101,045,133,045,189
5341 :152,101,046,133,046,160,091
5347 :000,000,177,122,145,036,195
5353 :200,208,249,230,123,230,193
5359 :037,165,046,197,037,176,129
5365 :239,032,051,197,165,034,195
5371 :166,035,024,105,002,133,204
5377 :045,144,001,232,134,046,091
5383 :032,089,198,076,103,228,221
5389 :032,124,197,032,115,000,001
5395 :000,133,139,162,000,000,197
5401 :134,073,032,140,253,000,145
5407 :165,165,201,000,000,208,002
5413 :007,162,002,134,073,032,191
5419 :140,253,000,032,115,000,071
5425 :000,240,003,032,253,206,015
5431 :032,178,253,000,165,095,010
5437 :166,096,133,122,134,123,067
5443 :032,215,202,208,011,200,167
5449 :152,024,101,122,133,122,215
5455 :144,002,230,123,032,202,044
5461 :255,000,240,005,032,220,069
5467 :253,000,176,003,076,143,230
5473 :252,000,132,085,230,085,113
5479 :164,085,166,049,165,050,014
5485 :133,139,177,122,240,216,112
5491 :221,000,000,002,208,237,015
5497 :232,200,198,139,208,241,059
5503 :136,132,011,132,151,165,086
5509 :073,240,091,032,240,253,038
5515 :000,165,052,056,229,050,179
5521 :133,167,240,040,200,240,141
```

```
5527 :202,177,122,208,249,024,109
5533 :152,101,167,201,002,144,156
5539 :064,201,075,176,060,165,136
5545 :167,016,002,198,139,024,203
5551 :101,011,133,151,176,005,240
5557 :032,036,254,000,240,003,234
5563 :032,012,254,000,165,151,033
5569 :056,229,052,168,200,165,039
5575 :052,240,015,133,140,166,177
5581 :051,189,000,000,002,145,080
5587 :122,232,200,198,140,208,031
5593 :245,024,165,045,101,167,196
5599 :133,045,165,046,101,139,084
5605 :133,046,165,122,166,123,216
5611 :133,095,134,096,166,067,158
5617 :165,068,032,061,254,000,053
5623 :032,225,255,169,000,000,160
5629 :133,198,164,151,076,242,193
5635 :252,000,164,122,200,148,121
5641 :049,169,000,000,149,050,170
5647 :185,000,000,002,240,021,207
5653 :197,139,240,005,246,050,130
5659 :200,208,242,132,122,096,003
5665 :201,171,240,004,201,045,127
5671 :208,001,096,076,008,207,123
5677 :144,005,240,003,032,166,123
5683 :253,000,032,107,201,032,164
5689 :019,198,032,121,000,000,171
5695 :240,011,032,166,253,000,253
5701 :032,115,000,000,032,107,099
5707 :201,208,224,165,020,005,130
5713 :021,208,006,169,255,133,105
5719 :020,133,021,096,032,202,079
5725 :255,000,133,067,032,202,014
5731 :255,000,133,068,056,165,008
5737 :020,229,067,165,021,229,068
5743 :068,096,165,122,133,034,217
5749 :165,123,133,035,165,045,015
5755 :133,036,165,046,133,037,161
5761 :096,165,034,197,036,208,097
5767 :004,165,035,197,037,096,157
5773 :164,011,200,177,034,164,123
5779 :151,200,145,034,032,001,198
5785 :254,000,208,001,096,230,174
5791 :034,208,236,230,035,208,086
5797 :232,164,011,177,036,164,181
5803 :151,145,036,032,001,254,022
5809 :000,208,001,096,165,036,171
5815 :208,002,198,037,198,036,094
```

```
5821 :076,036,254,000,160,000,203
5827 :000,132,165,132,015,032,159
5833 :205,221,169,032,164,165,133
5839 :041,127,032,210,255,201,049
5845 :034,208,006,165,015,073,202
5851 :255,133,015,200,177,095,070
5857 :240,025,016,236,201,255,174
5863 :240,232,036,015,048,228,006
5869 :132,165,032,124,254,000,176
5875 :200,177,174,048,214,032,064
5881 :210,255,208,246,032,215,135
5887 :202,056,096,160,157,132,034
5893 :174,160,192,132,175,056,126
5899 :233,127,170,160,000,000,189
5905 :202,240,238,230,174,208,029
5911 :002,230,175,177,174,016,029
5917 :246,048,241,032,107,201,136
5923 :165,020,133,053,165,021,080
5929 :133,054,032,253,206,032,239
5935 :107,201,165,020,133,051,212
5941 :165,021,133,052,032,142,086
5947 :198,032,202,255,000,032,010
5953 :202,255,000,208,033,032,027
5959 :172,255,000,032,202,255,219
5965 :000,032,202,255,000,208,006
5971 :003,076,143,252,000,032,077
5977 :202,255,000,165,099,145,187
5983 :122,032,202,255,000,165,103
5989 :098,145,122,032,183,255,168
5995 :000,240,226,032,202,255,038
6001 :000,032,202,255,000,032,122
6007 :202,255,000,201,034,208,251
6013 :011,032,202,255,000,240,097
6019 :197,201,034,208,247,240,234
6025 :238,170,240,188,016,233,198
6031 :162,004,221,212,255,000,229
6037 :240,005,202,208,248,240,012
6043 :221,165,122,133,059,165,252
6049 :123,133,060,032,115,000,112
6055 :000,176,211,032,107,201,126
6061 :032,081,255,000,165,060,254
6067 :133,123,165,059,133,122,146
6073 :160,000,000,162,000,000,251
6079 :189,000,000,001,201,048,118
6085 :144,017,072,032,115,000,065
6091 :000,144,003,032,130,255,255
6097 :000,104,160,000,000,145,106
6103 :122,232,208,232,032,115,132
6109 :000,000,176,008,032,145,070
```

```
6115 :255,000,032,121,000,000,123
6121 :144,248,201,044,240,184,014
6127 :208,150,032,172,255,000,032
6133 :032,202,255,000,032,202,200
6139 :255,000,208,008,169,255,122
6145 :133,099,133,098,048,014,014
6151 :032,202,255,000,197,020,201
6157 :208,015,032,202,255,000,213
6163 :197,021,208,011,032,209,185
6169 :221,169,032,076,210,255,220
6175 :032,202,255,000,032,183,223
6181 :255,000,240,210,032,162,168
6187 :255,000,230,151,032,036,235
6193 :254,000,230,045,208,002,020
6199 :230,046,096,032,162,255,108
6205 :000,198,151,032,012,254,196
6211 :000,165,045,208,002,198,173
6217 :046,198,045,096,032,240,218
6223 :253,000,160,000,000,132,112
6229 :011,132,151,096,165,053,181
6235 :133,099,165,054,133,098,005
6241 :076,142,198,165,099,024,033
6247 :101,051,133,099,165,098,238
6253 :101,052,133,098,032,202,215
6259 :255,000,208,251,096,160,061
6265 :000,000,230,122,208,002,171
6271 :230,123,177,122,096,137,244
6277 :138,141,167,067,072,065,015
6283 :078,071,197,068,069,076,186
6289 :069,084,197,070,073,078,204
6295 :196,075,073,076,204,078,085
6301 :085,077,066,069,210,000,152
6307 :000,165,252,000,065,252,129
6313 :000,165,252,000,198,251,011
6319 :000,152,254,000,172,251,236
6325 :000,013,013,013,013,013,246
```

Hexmon

Malcolm J. Clark

The use of a machine language monitor is often the easiest way to enter machine language programs. "Hexmon" was developed to satisfy this need. The program will save machine language programs to disk or tape.

"Hexmon," a machine language monitor written entirely in BASIC, will run on all VICs.

For the unexpanded VIC the program can be typed in exactly as listed. Of course the REM statements may be left out if desired. For an expanded VIC of 8K or more, the REMs *must* be left out along with the colons preceding them.

There is reason for all this stinginess. When combining a machine language program with a BASIC program as Hexmon does, you need to set an artificial top of memory. This is done by the POKEs in line 1. Without this the BASIC program will overwrite the machine language program area and nothing will work. In order for Hexmon to work and still leave some room for machine language programming in an unexpanded machine, compromise and code crunching are necessary.

Save It First!

Once you have the program entered into memory, it is imperative that you save it to tape or disk before you run it. Any program that contains POKE statements such as this one does is manipulating memory directly. A typo here can cause a system crash and you will lose the program. Better to be safe than sorry. If the program crashes, reload it, list it, find the errors, and make the corrections. If you get an out of memory error (you probably will at some point if you have an expanded VIC), type SYS 64802 before loading the program.

Once you get Hexmon to run, the first thing you will encounter is a menu of the function key operation. Study it for a moment and when you are ready to go on, hit f6. Hexmon will now ask you at what memory address you'd like to start. Enter the four digit hexadecimal address. It must be between 0000 and FFFF. Hexmon now jumps to its operating mode displaying the memory address and its value in hexadecimal. It should look like this:

LOC VAL ←
C000 78

Try entering C000. To do so, hit f1. This will reset the initial memory value. You should come up with 78 in the VAL column for C000. Now hit f7. LOC should have incremented by one to C001, and VAL should show E3. Next hit f5. You should be back to C000 and 78.

Hexadecimal Entry

If everything checks out so far, let's get out of the ROM and go to the RAM. Hit f1 and enter 1D11. VAL will be some random value. Let's put the hexadecimal equivalent of 255 in 1D11. That's FF and all you should have to do is hit the F key twice. Notice that LOC incremented automatically to 1D12, VAL is now displaying the value of 1D12, FF appears under the back arrow, and you were rewarded by a short beep. And you didn't even have to hit return.

To save keystrokes when entering long programs, Hexmon automatically enters the number when the second digit is hit. A beep is generated to indicate that a valid hexadecimal value was entered. This allows the user to enter digits from a printed hex dump without always having to look up at the screen. If an illegal entry is attempted, a rather rude low frequency tone sounds and the entire number must be reentered. Should the user realize that the first digit was not what he wanted, he can abort the entry by hitting f3 before he hits the second digit.

Now let's try out the key repeat feature. Hit f4. Now all VIC keys repeat. Try it by holding down the F key. LOC should increment steadily, accompanied by a string of beeps. Once LOC gets to 1D20, release the F key. You can turn off the auto repeat by hitting f4 again. Next we'll try a machine language SAVE. But first let's make sure FF got entered where it was supposed to. Step back to 1D11 using the F5 key. FF should appear under VAL for each location.

Machine Language Saves

To save our string of FFs, hit f2. Hexmon will now prompt PROG NAME=?. Enter any name up to 10 characters long, then hit return. Don't use quotes. Next comes the device number—1 for the tape cassette and 8 for the disk drive. Hit RETURN. Now we enter the starting address of the section of memory we want to save. Enter 1D11 <RETURN>. TO is always the address of the end of our save plus 1. Enter 1D21 <RETURN>. At this point the SAVE will start automatically if you are using a disk drive. If you're using a cassette, you will be instructed to PRESS RECORD & PLAY ON TAPE. Note that you will not get a prompt such as SAVING "TEST". The VIC does not generate a prompt when the save is within a BASIC program.

The end of the save will be indicated by the reappearance of the prompt ENTER INITIAL MEMORY LOCATION IN HEX. Alternatively changing line 93 to read GOTO 24 instead of GOTO13 will cause Hexmon to jump back to the point immediately before the save. I prefer this myself.

To make sure the SAVE routine is working properly, go back to 1D11 through 1D20 and change them to AA the same way you changed them to FF. Now exit Hexmon by hitting f8.

Next let's load those FF's back in memory. To load a machine language program where it came from, you must use the form LOAD "name", 1,1. The second 1 tells VIC that this program is to be loaded back in memory at exactly the same address it was saved from. Of course for the disk the first 1 should be changed to 8. After the program is loaded, type SYS 64802 and then reload Hexmon. Now use Hexmon to examine memory locations 1D11 through 1D20. They all should contain FF. Congratulations. You now have a machine language monitor.

Locating Machine Language

The area 1C41 through top of memory is available for machine language programming. On an unexpanded VIC this is 447 bytes. While this is not a great deal, it is more than adequate for short learning programs. The area 033C to 03FB is also available to disk users since it's the cassette buffer and is not used in disk operations. To run a machine language program, first exit Hexmon then SYS to the start of the ML program. To jump to 1C41, type SYS7233. 7233 is 1C41 in decimal. Before running a machine language program for the first time, you should always save it for the same reasons given earlier. If you don't, you may lose the program should it crash.

For information on machine language programming, you should obtain one of the many excellent books on the subject. The *VIC Programmer's Reference Guide* contains a lot of useful information for the machine language programmer. It contains invaluable reference data as well as an introduction to machine language. *Machine Language for Beginners* (COMPUTE! Books) is also another good source of information.

A good memory map is also a useful tool. A complete memory map can be found in *COMPUTE!'s Second Book of VIC* as well as *Mapping the VIC* by G. Russ Davies.

Hexmon

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

```
1 POKE55,255:POKE56,27:CLR:PRINT"{CLR}{BLK}":PRINT
  :POKE36879,232:PRINT                         :rem 15
2 PRINT"*******HEXMON********":PRINT           :rem 57
3 PRINT"{RVS}F1{OFF} RESET MEMORY LOC":PRINT
                                               :rem 32
4 PRINT"{RVS}F2{OFF} SAVE ML PROGRAM":PRINT
                                              :rem 200
5 PRINT"{RVS}F3{OFF} RESET ENTRY VALUE":PRINT
                                              :rem 124
6 PRINT"{RVS}F4{OFF} KEY REPEAT ON/OFF":PRINT
                                               :rem 61
7 PRINT"{RVS}F5{OFF} DECREMENT MEMORY":PRINT
                                               :rem 94
8 PRINT"{RVS}F6{OFF} START PROGRAM":PRINT :rem 150
9 PRINT"{RVS}F7{OFF} INCREMENT MEMORY":PRINT
                                              :rem 112
10 PRINT"{RVS}F8{OFF} EXIT TO BASIC":PRINT :rem 90
11 GETA$:IFA$=""THEN11                        :rem 231
12 IFASC(A$)<>139THEN11                        :rem 39
13 CLR:PRINT"{CLR}":ME=0:PRINT"ENTER INITIAL MEMOR
   Y"                                         :rem 136
14 PRINT"LOCATION IN HEX."                     :rem 57
15 PRINT:PRINT:FORX=3TO0STEP-1:REM GET HEX MEMORY
   {SPACE}ADDRESS                             :rem 210
16 GETA$:IFA$=""THEN16                        :rem 241
17 A=ASC(A$):IFA=133THEN13                    :rem 228
18 IFA>47ANDA<58ORA>64ANDA<71THEN20           :rem 114
19 GOTO13                                       :rem 7
20 IFA<58THENA=A-48                            :rem 98
21 IFA>57THENA=A-55                            :rem 98
22 PRINTA$;:ME=ME+A*16↑X:NEXT                  :rem 88
23 IF ME<0ORME>65535THEN13                    :rem 254
24 PRINT"{CLR}"                               :rem 202
25 PRINT"{HOME}"                               :rem 75
26 PRINT"LOC"TAB(6)"VAL"TAB(17)"←*":PRINT:PRINT:
   PRINT                                      :rem 228
```

```
27 IFME<ØTHENME=65535                          :rem 252
28 IFME>65535THENME=Ø                          :rem 255
29 V=ME:GOSUB58                                :rem 183
3Ø PRINTV4$+V3$+V2$+V1$;:REM PRINT ADDRESS IN HEX
                                               :rem 139
31 V=PEEK(ME):GOSUB58:REM GET VALUE IN HEX :rem 29
32 PRINTTAB(6)V2$+V1$;                          :rem 13
33 CT=1:DEC=Ø:REM GET NEW VALUE OR INCREMENT/DECRE
   MENT                                         :rem 79
34 GETA$:IFA$=""THEN34                         :rem 241
35 PRINTTAB(16+CT)A$;                          :rem 23Ø
36 A=ASC(A$):IFA=135THENME=ME-1:GOTO25          :rem 28
37 IFA=136THENME=ME+1:GOTO25                   :rem 215
38 IFA=133THEN13:REM CHECK FUNCTION KEYS       :rem 192
39 IFA=137THEN67                               :rem 176
4Ø IFA=134THEN25                               :rem 159
41 IFA=138THENRP=PEEK(65Ø)+128:IF RP=256THENRP=Ø
                                               :rem 118
42 IFA=138THENPOKE65Ø,RPT:GOTO25                 :rem 4
43 IFA=14ØTHENPRINT"{CLR}{RVS}BYE!!!{OFF}":FORX=1T
   O18:PRINT:NEXT:END                          :rem 199
44 IFA>47ANDA<58ORA>64ANDA<71THEN48:REM ERROR CHEC
   K KEY PRESS                                 :rem 247
45 POKE36878,15:POKE36874,16Ø                  :rem 114
46 FORX=1TO1ØØ:NEXT                            :rem 196
47 POKE36878,Ø:POKE36874,Ø:GOTO25              :rem 177
48 IFA<58THENA=A-48:REM CONVERT TO DECIMAL AND POK
   E                                            :rem 63
49 IFA>57THENA=A-55                            :rem 1Ø8
5Ø DE=DE+A*16↑CT                               :rem 166
51 CT=CT-1                                      :rem 47
52 IFCT<ØTHENPOKEME,DE:ME=ME+1:GOTO54          :rem 113
53 GOTO34                                        :rem 8
54 POKE36878,7:POKE36875,24Ø                    :rem 67
55 FORX=1TO25:NEXT                             :rem 154
56 POKE36878,Ø:POKE36875,Ø                     :rem 216
57 GOTO25                                       :rem 12
58 V4=INT(V/16↑3):REM DEIMAL TO HEX SUBROUTINE
                                                :rem 79
59 V=V-V4*16↑3:V4=V4+48:IFV4>57THENV4=V4+7 :rem 49
6Ø V3=INT(V/16↑2)                              :rem 228
61 V=V-V3*16↑2:V3=V3+48:IFV3>57THENV3=V3+7 :rem 35
62 V2=INT(V/16)                                 :rem 85
63 V1=V-V2*16:V2=V2+48:IFV2>57THENV2=V2+7 :rem 192
64 V1=V1+48:IFV1>57THENV1=V1+7                  :rem 34
65 V4$=CHR$(V4):V3$=CHR$(V3):V2$=CHR$(V2):V1$=CHR$
   (V1)                                         :rem 41
66 RETURN                                       :rem 76
```

```
67 PRINT"{CLR}":INPUT"PROG NAME=";NA$:REM PREPARE
   {SPACE}FOR ML SAVE                              :rem 63
68 L=LEN(NA$):POKE7180,L                           :rem 139
69 FORX=1TOL:POKE7169+X,ASC(MID$(NA$,X,1)):NEXT:PO
   KE7180,L                                        :rem 47
70 PRINT:INPUT"SAVE TO {RVS}1{OFF} OR {RVS}8{OFF}"
   ;DE                                             :rem 234
71 IFDE<>1ANDDE<>8THEN70                           :rem 207
72 PRINT:INPUT"FROM";AD$(1):PRINT                  :rem 101
73 INPUT"TO";AD$(2)                                :rem 72
74 FORX=1TO2:IFLEN(AD$(X))<>4THEN72                :rem 126
75 FORY=1TO4                                       :rem 241
76 A=ASC(MID$(AD$(X),Y,1))                         :rem 150
77 IFA>47ANDA<58ORA>64ANDA<71THEN79                :rem 133
78 PRINT:GOTO72                                    :rem 216
79 NEXTY                                           :rem 8
80 NEXTX                                           :rem 255
81 AD$=AD$(1)+AD$(2)                               :rem 209
82 FORX=1TO8:AD(X)=ASC(MID$(AD$,X,1))              :rem 152
83 IFAD(X)<58THENAD(X)=AD(X)-48:GOTO85             :rem 18
84 AD(X)=AD(X)-55                                  :rem 156
85 NEXTX                                           :rem 4
86 FORX=4TO1STEP-1                                 :rem 140
87 Z(X)=AD(X*2)+16*AD(X*2-1)                       :rem 221
88 NEXTX                                           :rem 7
89 POKE7181,Z(2):POKE7182,Z(1):POKE7183,Z(4):POKE7
   184,Z(3):POKE7169,DE                            :rem 64
90 FORX=0TO42:READM:POKE7190+X,M:NEXT:RESTORE:SYS7
   190:REM ML SAVE PROGRAM                         :rem 214
91 DATA173,1,28,174,1,28,160,255,32,186,255,173,12
   ,28,162,2,160,28,32,189,255,173,13,28  :rem 248
92 DATA133,251,173,14,28,133,252,169,251,174,15,28
   ,172,16,28,32,216,255,96                :rem 126
93 GOTO13                                          :rem 9
```

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 VIC-20s.

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 VIC's manual, *Personal Computing on the VIC.*

A Quick Review

1. Type in the program a line at a time, in order. Press RETURN at the end of each line. Use the INST/DEL key to erase 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.
3. Make sure you've entered statements in braces as the appropriate control key (see Appendix B, "How to Type In Programs").

How to Type In Programs

Many of the programs in this book contain special control characters (cursor control, color keys, reverse characters, and so on). 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, VIC-20 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.

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	

When You Read:	Press:	See:
{GRN}	CTRL 6	
{BLU}	CTRL 7	
{YEL}	CTRL 8	
{F1}	f1	
{F2}	SHIFT f1	
{F3}	f3	
{F4}	SHIFT f3	
{F5}	f5	
{F6}	SHIFT f5	
{F7}	f7	
{F8}	SHIFT f7	
←	←	
↑	SHIFT ↑	

Screen Location Table

Row	Address
0	7680 (4096)
	7702 (4118)
	7724 (4140)
	7746 (4162)
	7768 (4184)
5	7790 (4206)
	7812 (4228)
	7834 (4250)
	7856 (4272)
	7878 (4294)
10	7900 (4316)
	7922 (4338)
	7944 (4360)
	7966 (4382)
	7988 (4404)
15	8010 (4426)
	8032 (4448)
	8054 (4470)
	8076 (4492)
	8098 (4514)
20	8120 (4536)
	8142 (4558)
22	8164 (4580)

0 5 10 15 20

Column

Note: Numbers in parentheses are for VICs with 8K or more of memory expansion.

Screen Color Memory Table

Row	
0	38400 (37888)
	38422 (37910)
	38444 (37932)
	38466 (37954)
	38488 (37976)
5	38510 (37998)
	38532 (38020)
	38554 (38042)
	38576 (38064)
	38598 (38086)
10	38620 (38108)
	38642 (38130)
	38664 (38152)
	38686 (38174)
	38708 (38196)
15	38730 (38218)
	38752 (38240)
	38774 (38262)
	38796 (38284)
	38818 (38306)
20	38840 (38328)
	38862 (38350)
22	38884 (38372)

0 5 10 15 20

Column

Note: Numbers in parentheses are for VICs with 8K or more of memory expansion.

Screen Color Codes

Color:	BLK	WHT	RED	CYN	PUR	GRN	BLU	YEL
Code:	0	1	2	3	4	5	6	7

Screen and Border Colors

	Border							
Screen	Black	White	Red	Cyan	Purple	Green	Blue	Yellow
Black	8	9	10	11	12	13	14	15
White	24	25	26	27	28	29	30	31
Red	40	41	42	43	44	45	46	47
Cyan	56	57	58	59	60	61	62	63
Purple	72	73	74	75	76	77	78	79
Green	88	89	90	91	92	93	94	95
Blue	104	105	106	107	108	109	110	111
Yellow	120	121	122	123	124	125	126	127
Orange	136	137	138	139	140	141	142	143
Light Orange	152	153	154	155	156	157	158	159
Pink	168	169	170	171	172	173	174	175
Light Cyan	184	185	186	187	188	189	190	191
Light Purple	200	201	202	203	204	205	206	207
Light Green	216	217	218	219	220	221	222	223
Light Blue	232	233	234	235	236	237	238	239
Light Yellow	248	249	250	251	252	253	254	255

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

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	π
90	Z	127	
91	[	133	f1
92	£	134	f3
93	]	135	f5
94	↑	136	f7
95	←	137	f2
96		138	f4
97	♠	139	f6
98		140	f8
99		141	SHIFTED RETURN
100		142	UPPERCASE
101		144	BLACK
102		145	CURSOR UP
103		146	REVERSE VIDEO OFF
104		147	CLEAR SCREEN
105		148	INSERT
106		156	PURPLE
107		157	CURSOR LEFT
108		158	YELLOW
109		159	CYAN
110		160	SHIFTED SPACE
111		161	
112		162	
113	●	163	
114		164	
115	♥	165	
116		166	
117		167	
118		168	
119	○	169	

ASCII	CHARACTER	ASCII	CHARACTER
170		207	
171		208	
172		209	
173		210	
174		211	♥
175		212	
176		213	
177		214	
178		215	
179		216	♣
180		217	
181		218	♦
182		219	
183		220	
184		221	
185		222	π
186		223	
187		224	SPACE
188		225	
189		226	
190		227	
191		228	
192		229	
193	♠	230	
194		231	
195		232	
196		233	
197		234	
198		235	
199		236	
200		237	
201		238	
202		239	
203		240	
204		241	
205		242	
206		243	

Appendix G

ASCII	CHARACTER
244	
245	
246	
247	
248	
249	
250	
251	
252	
253	
254	
255	π

1. 0-4, 6-7, 10-12, 15-16, 21-27, 128-132, 143, and 149-155 have no effect.
2. 192-223 same as 96-127, 224-254 same as 160-190, 255 same as 126.

Screen Codes

POKE	Uppercase and Full Graphics Set	Lower- and Uppercase
0	@	@
1	A	a
2	B	b
3	C	c
4	D	d
5	E	e
6	F	f
7	G	g
8	H	h
9	I	i
10	J	j
11	K	k
12	L	l
13	M	m
14	N	n
15	O	o
16	P	p
17	Q	q
18	R	r
19	S	s
20	T	t
21	U	u
22	V	v
23	W	w
24	X	x
25	Y	y
26	Z	z
27	[	[
28	£	£
29	]	]
30	↑	↑
31	←	←
32	-space-	
33	!	!
34	"	"
35	#	#
36	$	$
37	%	%
38	&	&
39	'	'
40	(	(
41	)	)
42	*	*
43	+	+
44	,	,
45	-	-
46	.	.
47	/	/
48	0	0
49	1	1
50	2	2
51	3	3
52	4	4
53	5	5
54	6	6
55	7	7
56	8	8
57	9	9
58	:	:
59	;	;
60	<	<
61	=	=

POKE	Uppercase and Full Graphics Set	Lower- and Uppercase	POKE	Uppercase and Full Graphics Set	Lower- and Uppercase
62	>	>	95		
63	?	?	96	- -space- -	
64			97		
65	♠	A	98		
66		B	99		
67		C	100		
68		D	101		
69		E	102		
70		F	103		
71		G	104		
72		H	105		
73		I	106		
74		J	107		
75		K	108		
76		L	109		
77		M	110		
78		N	111		
79		O	112		
80		P	113		
81	●	Q	114		
82		R	115		
83	♥	S	116		
84		T	117		
85		U	118		
86		V	119		
87		W	120		
88	♣	X	121		
89		Y	122		
90	♦	Z	123		
91			124		
92			125		
93			126		
94	π		127		

128–255 reverse of 0–127

VIC-20 Keycodes

Values Stored at Location 197

Code	Key pressed	Code	Key pressed
0	1	33	Z
1	3	34	C
2	5	35	B
3	7	36	M
4	9	37	.
5	+	39	f1
6	£	41	S
7	DEL	42	F
8	←	43	H
9	W	44	K
10	R	45	:
11	Y	46	=
12	I	47	f3
13	P	48	Q
14	*	49	E
15	RETURN	50	T
17	A	51	U
18	D	52	O
19	G	53	@
20	J	54	↑
21	L	55	f5
22	;	56	2
23	{cursor left-right}	57	4
24	RUN-STOP	58	6
26	X	59	8
27	V	60	0
28	N	61	-
29	,	62	CLR/HOME
30	/	63	f7
31	{cursor up-down}	64	{No key pressed}
32	{space bar}		

The following key codes cannot occur: 16, 25, 38, 40.

Values Stored at Location 653

Code	Key(s) pressed
0	(No key pressed)
1	SHIFT
2	Commodore
3	SHIFT and Commodore
4	CTRL
5	SHIFT and CTRL
6	Commodore and CTRL
7	SHIFT, Commodore, and CTRL

The Automatic Proofreader

Charles Brannon

"The Automatic Proofreader" will help you type in program listings without typing mistakes. It is a short error-checking program that hides itself in memory. When activated, it lets you know immediately after typing a line from a program listing if you have made a mistake. Please read these instructions carefully before typing any programs in this book.

Preparing the Proofreader

1. Using the listing below, type in the Proofreader. Be very careful when entering the DATA statements—don't type an l instead of a 1, an O instead of a 0, extra commas, etc.

2. SAVE the Proofreader on tape or disk at least twice *before running it for the first time.* This is very important because the Proofreader erases part of itself when you first type RUN.

3. After the Proofreader is saved, type RUN. It will check itself for typing errors in the DATA statements and warn you if there's a mistake. Correct any errors and SAVE the corrected version. Keep a copy in a safe place—you'll need it again and again, every time you enter a program from this book, *COMPUTE!'s Gazette*, or *COMPUTE!* magazine.

4. When a correct version of the Proofreader is run, it activates itself. You are now ready to enter a program listing. If you press RUN/STOP–RESTORE, the Proofreader is disabled. To reactivate it, just type the command SYS 886 and press RETURN.

Using the Proofreader

All listings in this book have a *checksum number* appended to the end of each line, for example, :rem 123. *Don't enter this statement when typing in a program.* It is just for your information. The rem makes the number harmless if someone does type it in. It will, however, use up memory if you enter it, and it will confuse the Proofreader, even if you entered the rest of the line correctly.

When you type in a line from a program listing and press RETURN, the Proofreader displays a number at the top of your screen. *This checksum number must match the checksum number in the printed listing.* If it doesn't, it means you typed the line differently than the way it is listed. Immediately recheck your typing. Remember, don't type the rem statement with the checksum number; it is published only so you can check it against the number which appears on your screen.

The Proofreader is not picky with spaces. It will not notice extra spaces or missing ones. This is for your convenience, since spacing is generally not important. But occasionally proper spacing *is* important, so be extra careful with spaces, since the Proofreader will catch practically everything else that can go wrong.

There's another thing to watch out for: If you enter a line by using abbreviations for commands, the checksum will not match up. But there is a way to make the Proofreader check it. After entering the line; LIST it. This eliminates the abbreviations. Then move the cursor up to the line and press RETURN. It should now match the checksum. You can check whole groups of lines this way.

Special Tape SAVE Instructions

When you're done typing a listing, you must disable the Proofreader before SAVEing the program on tape. Disable the Proofreader by pressing RUN/STOP–RESTORE (hold down the RUN/STOP key and sharply hit the RESTORE key). This procedure is not necessary for disk SAVEs, *but you must disable the Proofreader this way before a tape SAVE.*

SAVE to tape erases the Proofreader from memory, so you'll have to LOAD and RUN it again if you want to type another listing. SAVE to disk does not erase the Proofreader.

Hidden Perils

The proofreader's home in the VIC is not a very safe haven. Since the cassette buffer is wiped out during tape operations, you need to disable the Proofreader with RUN/STOP–RESTORE before you SAVE your program. This applies only to tape use. Disk users have nothing to worry about.

What if you type in a program in several sittings? The next day, you come to your computer, load and run the Proofreader, then try to load the partially completed program so you can add to it. But since the Proofreader is trying to hide in the cassette buffer, it is wiped out!

What you need is a way to load the Proofreader after you've loaded the partial program. The problem is, a tape load to the buffer destroys what it's supposed to load.

After you've typed in and RUN the Proofreader, enter the following lines in direct mode (without line numbers) exactly as shown:

A$="PROOFREADER.T": B$="{10 SPACES}": FOR X = 1 TO 4: A$=A$+B$: NEXTX

FOR X = 886 TO 1018: A$=A$+CHR$ (PEEK(X)): NEXTX

OPEN 1, 1,1,A$:CLOSE1

After you enter the last line, you will be asked to press record and play on your cassette recorder. Put this program at the beginning of a new tape. This gives you a new way to load the Proofreader. Anytime you want to bring the Proofreader into memory without disturbing anything else, put the cassette in the tape drive, rewind, and enter:

OPEN1:CLOSE1

You can now start the Proofreader by typing SYS 886. To test this, PRINT PEEK (886) should return the number 173. If it does not, repeat the steps above, making sure that A$ ("PROOFREADER.T") contains 13 characters and that B$ contains 10 spaces.

You can now reload the Proofreader into memory whenever LOAD or SAVE destroys it, restoring your personal typing helper.

Replace Original Proofreader

If you typed in the original version of the Proofreader from the October 1983 issue of *COMPUTE!'s Gazette*, you should replace it with the improved version below.

Automatic Proofreader

```
100 PRINT"{CLR}PLEASE WAIT...":FORI=886TO1018:READ
    A:CK=CK+A:POKEI,A:NEXT
110 IF CK<>17539 THEN PRINT"{DOWN}YOU MADE AN ERRO
    R":PRINT"IN DATA STATEMENTS.":END
```

```
120 SYS886:PRINT"{CLR}{2 DOWN}PROOFREADER ACTIVATED.":NEW
886 DATA 173,036,003,201,150,208
892 DATA 001,096,141,151,003,173
898 DATA 037,003,141,152,003,169
904 DATA 150,141,036,003,169,003
910 DATA 141,037,003,169,000,133
916 DATA 254,096,032,087,241,133
922 DATA 251,134,252,132,253,008
928 DATA 201,013,240,017,201,032
934 DATA 240,005,024,101,254,133
940 DATA 254,165,251,166,252,164
946 DATA 253,040,096,169,013,032
952 DATA 210,255,165,214,141,251
958 DATA 003,206,251,003,169,000
964 DATA 133,216,169,019,032,210
970 DATA 255,169,018,032,210,255
976 DATA 169,058,032,210,255,166
982 DATA 254,169,000,133,254,172
988 DATA 151,003,192,087,208,006
994 DATA 032,205,189,076,235,003
1000 DATA 032,205,221,169,032,032
1006 DATA 210,255,032,210,255,173
1012 DATA 251,003,133,214,076,173
1018 DATA 003
```

Using the Machine Language Editor: MLX

Charles Brannon

"The Machine Language Editor" will make the entering of "SpeedScript" and "Tiny Aid" easy and fast. MLX and SpeedScript require at least 8K memory expansion. Tiny Aid requires 8K or more memory expansion to be entered with MLX, but will RUN on a VIC with any amount of memory.

Remember the last time you typed in the BASIC loader for a long machine language program? You typed in hundreds of numbers and commas. Even then, you couldn't be sure if you typed it in right. So you went back, proofread, tried to run the program, crashed, went back and proofread again, corrected a few typing errors, ran again, crashed again, rechecked your typing. Frustrating, wasn't it?

Until now, though, that has been the best way to get machine language into your computer. Unless you happen to have an assembler and are willing to wrangle with machine language on the assembly level, it is much easier to enter a BASIC program that reads DATA statements and POKEs the numbers into memory.

Some of these "BASIC loaders" will use a *checksum* to see if you've typed the numbers correctly. The simplest checksum is just the sum of all the numbers in the DATA statements. If you make an error, your checksum will not match up with the total. Some programmers make your task easier by including checksums every few lines, so you can locate your errors more easily.

Now, MLX comes to the rescue. MLX is a great way to enter all those long machine language programs with a minimum of fuss. MLX lets you enter the numbers from a special list that looks similar to DATA statements. It checks your typing on a line-by-line basis. It won't let you enter illegal characters when you should be typing numbers. It won't let you enter numbers greater than 255. It will prevent you from entering the numbers on the wrong line. In short, MLX will make proofreading obsolete.

Tape or Disk Copies

In addition, MLX will generate a ready-to-use copy of your machine language program on tape or disk. You can then use the LOAD command to read the program into the computer, just like a BASIC program. Specifically, you enter:

LOAD "program name",1,1 (for tape)

or

LOAD "program name",8,1 (for disk)

To start the program, you need to enter a SYS command that transfers control from BASIC to your machine language program. The starting SYS will always be given in the article which presents the machine language program in MLX format.

Using MLX

Type in and SAVE MLX (you'll want to use it in the future). When you're ready to type in the machine language program "SpeedScript," enter this line in direct mode before loading the MLX program:

POKE 44,37:POKE 9472,0:NEW

When you're ready to type in the machine language program "Tiny Aid," enter this line in direct mode before loading the MLX program:

POKE 44,25:POKE 6400,0:NEW

Then RUN MLX. MLX will ask you for two numbers: the starting address and the ending address. For Speedscript, these numbers should be: 4609 and 9348, respectively; for Tiny Aid the numbers should be 4609 and 6326.

You'll then see a prompt. The prompt is the current line you are entering from the MLX-format listing. Each line is six numbers plus a checksum. If you enter any of the six numbers wrong, or enter the checksum wrong, the VIC will sound a buzzer and prompt you to reenter the entire line. If you enter the line correctly, a pleasant bell tone will sound and you may go on to enter the next line.

Each time you use MLX to enter SpeedScript or Tiny Aid, you must enter the appropriate line above before LOADing MLX. These lines are necessary only when using MLX and are not necessary once you have saved a completed copy of SpeedScript and Tiny Aid.

A Special Editor

You are not using the normal VIC BASIC editor with MLX. For example, it will only accept numbers as input. If you need to make a correction, press the INST/DEL key; the entire number is deleted. You can press it as many times as necessary, back to the start of the line. If you enter three-digit numbers as listed, the computer will automatically print the comma and go on to accept the next number in the line. If you enter less than three digits, you can press either the comma, space bar, or RETURN key to advance to the next number. The checksum will automatically appear in inverse video; don't worry—it's highlighted for emphasis.

When testing it, I've found MLX to be an extremely easy way to enter long listings. With the audio cues provided, you don't even have to look at the screen if you're a touch-typist.

Done at Last!

When you get through typing, assuming you type your machine language program all in one session, you can then save the completed and bug-free program to tape or disk. Follow the instructions displayed on the screen. If you get any error messages while saving, you probably have a bad disk, or the disk was full, or you made a typo when entering the MLX program. (Sorry, MLX can't check itself!)

Command Control

What if you don't want to enter the whole program in one sitting? MLX lets you enter as much as you want, save the completed portion, and then reload your work from tape or disk when you want to continue. MLX recognizes these few commands:

SHIFT-S: Save
SHIFT-L: Load
SHIFT-N: New Address
SHIFT-D: Display

Hold down SHIFT while you press the appropriate key. You will jump out of the line you've been typing, so I recommend you do it at a prompt. Use the Save command to store what you've been working on. It will write the tape or disk file as if you've finished. Remember what address you stop on. The next time you RUN MLX answer all the prompts as you did before, then insert the disk or tape containing the stored file. When you get the entry prompt press SHIFT-L to reload the file into memory. You'll then use the New Address command (SHIFT-N) to resume typing.

New Address and Display

After you press SHIFT-N, enter the address where you previously stopped. The prompt will change, and you can then continue typing. Always enter a New Address that matches up with one of the line numbers in the special listing, or else the checksums won't match up. You can use the Display command to display a section of your typing. After you press SHIFT-D, enter two addresses within the line number range of the listing. You can stop the display by pressing any key.

Tricky Stuff

The special commands may seem a little confusing, but as you work with MLX, they will become valuable. For example, what if you forgot where you stopped typing? Use the Display command to scan memory from the beginning to the end of the program. When you reach the end of your typing, the lines will contain a random pattern of numbers. When you see the end of your typing, press any key to stop the listing. Use the New Address command to continue typing from the proper location.

You can use the Save and Load commands to make copies of the complete machine language program. Use the Load command to reload the tape or disk, then insert a new tape or disk and use the Save command to create a new copy. When resaving on disk, it is best to use a different filename each time you save. For example, I like to number my work and use filenames such as SCRIPT1, SCRIPT2, SCRIPT3, etc.

One quirk about tapes made with the MLX Save command: When you load them, the message "FOUND program" may appear twice. The tape will load just fine, however.

Programmers will find MLX to be an interesting program which protects the user from most typing mistakes. Some screen-formatting techniques are also used. Most interesting is the use of ROM Kernal routines for LOADing and SAVEing blocks of memory. To use these routines, just POKE in the starting address (low byte/high byte) into memory locations 251 and 252, and POKE the ending address into locations 254 and 255. Any error code for the SAVE or LOAD can be found in location 253 (an error would be a code less than ten).

I hope you will find MLX to be a true labor-saving program. Since it has been tested by entering actual programs, you can count on it as an aid for generating bug-free machine language. Be sure to save MLX; it will be used for future applications in *COMPUTE!* magazine, *COMPUTE!'s Gazette* and COMPUTE! Books.

MLX

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

```
100 PRINT"{CLR}{PUR}";CHR$(142);CHR$(8);    :rem 181
101 POKE 788,194:REM DISABLE RUN/STOP      :rem 174
110 PRINT"{RVS}{14 SPACES}"                :rem 117
120 PRINT"{RVS} {RIGHT}{OFF}{*}£{RVS}{RIGHT}
    {RIGHT}{2 SPACES}{*}{OFF}{*}£{RVS}£{RVS} "
                                           :rem 191
130 PRINT"{RVS} {RIGHT} {G}{RIGHT} {2 RIGHT} {OFF}
    £{RVS}£{*}{OFF}{*}{RVS} "             :rem 232
140 PRINT"{RVS}{14 SPACES}"                :rem 120
200 PRINT"{2 DOWN}{PUR}{BLK}A FAILSAFE MACHINE":PR
    INT"LANGUAGE EDITOR{5 DOWN}"           :rem 141
210 PRINT"{BLK}{3 UP}STARTING ADDRESS":INPUTS:F=1-
    F:C$=CHR$(31+119*F)                     :rem 97
220 IFS<256ORS>32767THENGOSUB3000:GOTO210    :rem 2
225 PRINT:PRINT:PRINT:PRINT                :rem 123
230 PRINT"{BLK}{3 UP}ENDING ADDRESS":INPUTE:F=1-F:
    C$=CHR$(31+119*F)                      :rem 158
240 IFE<256ORE>32767THENGOSUB3000:GOTO230  :rem 234
250 IFE<STHENPRINTC$;"{RVS}ENDING < START
    {2 SPACES}":GOSUB1000:GOTO 230         :rem 176
260 PRINT:PRINT:PRINT                      :rem 179
300 PRINT"{CLR}";CHR$(14):AD=S              :rem 56
310 PRINTRIGHT$("0000"+MID$(STR$(AD),2),5);":";:FO
    RJ=1TO6                                :rem 234
320 GOSUB570:IFN=-1THENJ=J+N:GOTO320       :rem 228
390 IFN=-211THEN 710                        :rem 62
400 IFN=-204THEN 790                        :rem 64
410 IFN=-206THENPRINT:INPUT"{DOWN}ENTER NEW ADDRES
    S";ZZ                                   :rem 44
415 IFN=-206THENIFZZ<SORZZ>ETHENPRINT"{RVS}OUT OF
    {SPACE}RANGE":GOSUB1000:GOTO410        :rem 225
417 IFN=-206THENAD=ZZ:PRINT:GOTO310        :rem 238
420 IF N<>-196 THEN 480                    :rem 133
430 PRINT:INPUT"DISPLAY:FROM";F:PRINT,"TO";:INPUTT
                                           :rem 234
```

```
440 IFF<SORF>EORT<SORT>ETHENPRINT"AT LEAST";S;"
    {LEFT}, NOT MORE THAN";E:GOTO430          :rem 159
450 FORI=FTOTSTEP6:PRINT:PRINTRIGHT$("ØØØØ"+MID$(S
    TR$(I),2),5);":";                          :rem 30
455 FORK=ØTO5:N=PEEK(I+K):IFK=3THENPRINTSPC(1Ø);
                                               :rem 34
457 PRINTRIGHT$("ØØ"+MID$(STR$(N),2),3);",";
                                              :rem 157
46Ø GETA$:IFA$>""THENPRINT:PRINT:GOTO31Ø     :rem 25
47Ø NEXTK:PRINTCHR$(2Ø);:NEXTI:PRINT:PRINT:GOTO31Ø
                                               :rem 5Ø
48Ø IFN<Ø THEN PRINT:GOTO31Ø                 :rem 168
49Ø A(J)=N:NEXTJ                              :rem 199
5ØØ CKSUM=AD-INT(AD/256)*256:FORI=1TO6:CKSUM=(CKSU
    M+A(I))AND255:NEXT                        :rem 2ØØ
51Ø PRINTCHR$(18);:GOSUB57Ø:PRINTCHR$(2Ø) :rem 234
515 IFN=CKSUMTHEN53Ø                          :rem 255
52Ø PRINT:PRINT"LINE ENTERED WRONG":PRINT"RE-ENTER
    ":PRINT:GOSUB1ØØØ:GOTO31Ø                :rem 129
53Ø GOSUB2ØØØ                                 :rem 218
54Ø FORI=1TO6:POKEAD+I-1,A(I):NEXT            :rem 8Ø
55Ø AD=AD+6:IF AD<E THEN 31Ø                  :rem 212
56Ø GOTO 71Ø                                  :rem 1Ø8
57Ø N=Ø:Z=Ø                                    :rem 88
58Ø PRINT"{+}";                                :rem 79
581 GETA$:IFA$=""THEN581                       :rem 95
585 PRINTCHR$(2Ø);:A=ASC(A$):IFA=13ORA=44ORA=32THE
    N67Ø                                      :rem 229
59Ø IFA>128THENN=-A:RETURN                    :rem 137
6ØØ IFA<>2Ø THEN 63Ø                           :rem 1Ø
61Ø GOSUB69Ø:IFI=1ANDT=44THENN=-1:PRINT"{LEFT}
    {LEFT}";:GOTO69Ø                          :rem 172
62Ø GOTO57Ø                                   :rem 1Ø9
63Ø IFA<48ORA>57THEN58Ø                       :rem 1Ø5
64Ø PRINTA$;:N=N*1Ø+A-48                      :rem 1Ø6
65Ø IFN>255 THEN A=2Ø:GOSUB1ØØØ:GOTO6ØØ      :rem 229
66Ø Z=Z+1:IFZ<3THEN58Ø                         :rem 71
67Ø IFZ=ØTHENGOSUB1ØØØ:GOTO57Ø               :rem 114
68Ø PRINT",";:RETURN                          :rem 24Ø
69Ø S%=PEEK(2Ø9)+256*PEEK(21Ø)+PEEK(211)    :rem 149
692 FORI=1TO3:T=PEEK(S%-I)                     :rem 68
695 IFT<>44ANDT<>58THENPOKES%-I,32:NEXT      :rem 2Ø5
7ØØ PRINTLEFT$("{3 LEFT}",I-1);:RETURN          :rem 7
71Ø PRINT"{CLR}{RVS}*** SAVE ***{3 DOWN}"   :rem 236
72Ø INPUT"{DOWN} FILENAME";F$                :rem 228
73Ø PRINT:PRINT"{2 DOWN}{RVS}T{OFF}APE OR {RVS}D
    {OFF}ISK: (T/D)"                          :rem 228
74Ø GETA$:IFA$<>"T"ANDA$<>"D"THEN74Ø          :rem 36
75Ø DV=1-7*(A$="D"):IFDV=8THENF$="Ø:"+F$     :rem 158
```

```
760 T$=F$:ZK=PEEK(53)+256*PEEK(54)-LEN(T$):POKE782
    ,ZK/256                                   :rem 3
762 POKE781,ZK-PEEK(782)*256:POKE780,LEN(T$):SYS65
    469                                     :rem 109
763 POKE780,1:POKE781,DV:POKE782,1:SYS65466:rem 69
765 POKE254,S/256:POKE253,S-PEEK(254)*256:POKE780,
    253                                      :rem 12
766 POKE782,E/256:POKE781,E-PEEK(782)*256:SYS65496
                                            :rem 124
770 IF(PEEK(783)AND1)OR(ST AND191)THEN780 :rem 111
775 PRINT"{DOWN}DONE.":END                  :rem 106
780 PRINT"{DOWN}ERROR ON SAVE.{2 SPACES}TRY AGAIN.
    ":IFDV=1THEN720                         :rem 171
781 OPEN15,8,15:INPUT#15,E1$,E2$:PRINTE1$;E2$:CLOS
    E15:GOTO720                             :rem 103
782 GOTO720                                 :rem 115
790 PRINT"{CLR}{RVS}*** LOAD ***{2 DOWN}"  :rem 212
800 INPUT"{2 DOWN} FILENAME";F$            :rem 244
810 PRINT:PRINT"{2 DOWN}{RVS}T{OFF}APE OR {RVS}D
    {OFF}ISK: (T/D)"                        :rem 227
820 GETA$:IFA$<>"T"ANDA$<>"D"THEN820        :rem 34
830 DV=1-7*(A$="D"):IFDV=8THENF$="0:"+F$   :rem 157
840 T$=F$:ZK=PEEK(53)+256*PEEK(54)-LEN(T$):POKE782
    ,ZK/256                                   :rem 2
841 POKE781,ZK-PEEK(782)*256:POKE780,LEN(T$):SYS65
    469                                     :rem 107
845 POKE780,1:POKE781,DV:POKE782,1:SYS65466:rem 70
850 POKE780,0:SYS65493                       :rem 11
860 IF(PEEK(783)AND1)OR(ST AND191)THEN870 :rem 111
865 PRINT"{DOWN}DONE.":GOTO310               :rem 96
870 PRINT"{DOWN}ERROR ON LOAD.{2 SPACES}TRY AGAIN.
    {DOWN}":IFDV=1THEN800                   :rem 172
880 OPEN15,8,15:INPUT#15,E1$,E2$:PRINTE1$;E2$:CLOS
    E15:GOTO800                             :rem 102
1000 REM BUZZER                             :rem 135
1001 POKE36878,15:POKE36874,190             :rem 206
1002 FORW=1TO300:NEXTW                      :rem 117
1003 POKE36878,0:POKE36874,0:RETURN          :rem 74
2000 REM BELL SOUND                          :rem 78
2001 FORW=15TO0STEP-1:POKE36878,W:POKE36876,240:NE
     XTW                                     :rem 22
2002 POKE36876,0:RETURN                     :rem 119
3000 PRINTC$;"{RVS}NOT ZERO PAGE OR ROM":GOTO1000
                                             :rem 89
```

Index

Notes

If you've enjoyed the articles in this book, you'll find the same style and quality in every monthly issue of **COMPUTE!** Magazine. Use this form to order your subscription to **COMPUTE!**.

COMPUTE! Books

P.O. Box 5406 Greensboro, NC 27403

Ask your retailer for these **COMPUTE! Books**. If he or she has sold out, order directly from **COMPUTE!**

For Fastest Service
Call Our **TOLL FREE US Order Line**
800-334-0868
In NC call 919-275-9809

Quantity	Title	Price	Total
______	Machine Language for Beginners	**$14.95***	______
______	Home Energy Applications	**$14.95***	______
______	COMPUTE!'s First Book of VIC	**$12.95***	______
______	COMPUTE!'s Second Book of VIC	**$12.95***	______
______	COMPUTE!'s First Book of VIC Games	**$12.95***	______
______	COMPUTE!'s First Book of 64	**$12.95***	______
______	COMPUTE!'s First Book of Atari	**$12.95***	______
______	COMPUTE!'s Second Book of Atari	**$12.95***	______
______	COMPUTE!'s First Book of Atari Graphics	**$12.95***	______
______	COMPUTE!'s First Book of Atari Games	**$12.95***	______
______	Mapping The Atari	**$14.95***	______
______	Inside Atari DOS	**$19.95***	______
______	The Atari BASIC Sourcebook	**$12.95***	______
______	Programmer's Reference Guide for TI-99/4A	**$14.95***	______
______	COMPUTE!'s First Book of TI Games	**$12.95***	______
______	Every Kid's First Book of Robots and Computers	**$ 4.95†**	______
______	The Beginner's Guide to Buying A Personal Computer	**$ 3.95†**	______

* Add $2 shipping and handling. Outside US add $5 air mail; $2 surface mail.

† Add $1 shipping and handling. Outside US add $5 air mail; $2 surface mail.

Please add shipping and handling for each book ordered. ______

Total enclosed or to be charged. ______

All orders must be prepaid (money order, check, or charge). All payments must be in US funds. NC residents add 4% sales tax.

☐ Payment enclosed Please charge my: ☐ VISA ☐ MasterCard
☐ American Express Acc't. No. ______ Expires ____/____

Name ______

Address ______

City ______ State ______ Zip ______

Country ______

Allow 4-5 weeks for delivery.

COMPUTE!'s
Third Book of VIC

COMPUTE!'s Third Book of VIC continues the popular tradition of bringing you the best from *COMPUTE!* magazine and *COMPUTE!'s Gazette*, as well as several never-before-published articles and programs. Both *COMPUTE!'s First Book* and *Second Book of VIC* were best sellers. In this book you'll find the same high quality that you have learned to expect from COMPUTE! Books.

Here are some of the exciting programs and tutorials inside:

- "SpeedScript," an all machine language word processor
- Clear explanations of just how to make custom characters
- Five complete and ready-to-run games
- Programs to help you create your own music
- "Hexmon," a machine language monitor for the unexpanded VIC
- Ways to save memory when programming
- How to program the function keys
- A disk menu utility that will list and run your programs

It doesn't matter whether you're a beginning or an advanced programmer; you'll find *COMPUTE!'s Third Book of VIC* full of useful information that will help you get more from your VIC. As with all COMPUTE! Books, you'll find the articles clearly written and easy to understand and use.

ISBN 0-942386-43-4

COMPUTE!'s THIRD BOOK OF VIC

COMPUTE! Books