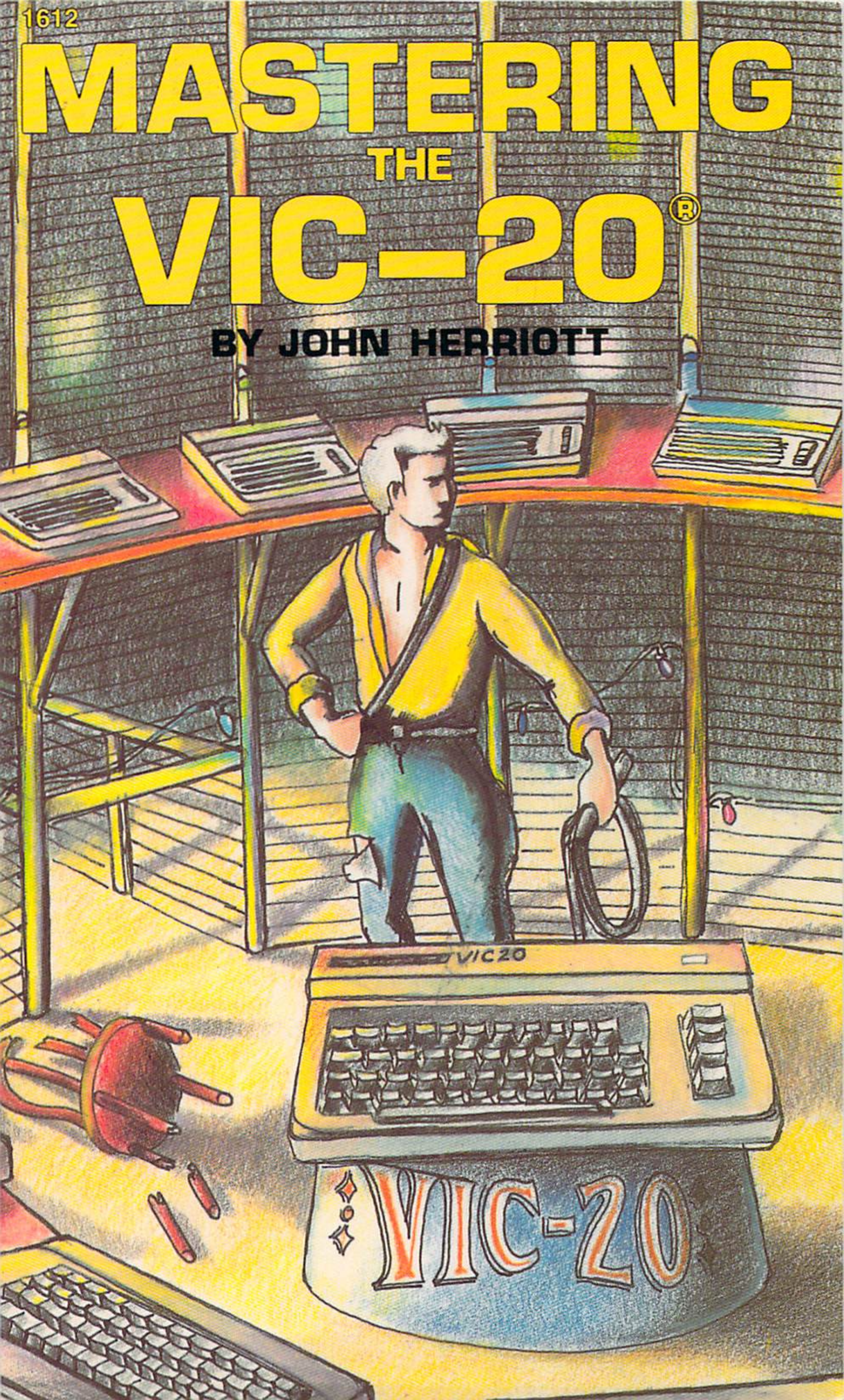
1612

MASTERING THE VIC-20®

BY JOHN HERRIOTT



MASTERING THE VIC-20®

BY JOHN HERRIOTT

TAB TAB BOOKS Inc.
BLUE RIDGE SUMMIT, PA. 17214

FIRST EDITION

FIRST PRINTING

Library of Congress Cataloging in Publication Data

Herriott, John.
    Mastering the VIC-20.

    Includes index.
    1. VIC 20 (Computer)—Programming. I. Title.
QA76.8.V5H47 1983 001.64 83-6754
ISBN 0-8306-0612-2
ISBN 0-8306-1612-8 (pbk.)

Contents

Acknowledgments

An author is indebted to many people for assistance, even when that assistance has not been obvious. For my part, I wish to thank my wife for being astonishingly forbearing; my son Michael for the "Flight of the Bumble Bee" program; my son Charles for preparing the coding data for the Russian character set; and finally, Dr. Harry Edstrom, Assistant Dean of the Faculty of Medicine, Memorial University of Newfoundland, for the Home Medicine Chest program and the loan of a typewriter when mine went berserk in the middle of the composition of this work!

Introduction

You have either bought or are about to buy, a VIC-20 computer. This book is a companion to that computer and deals only with matters concerning the machine and the things you can attach to it.

An instruction manual comes with the computer, as it does with any appliance. In addition, Commodore produces a programmer's reference guide. The aim of this book is to both complement and compliment each of those texts in much the same way that a cookbook complements the manual that comes with a microwave oven.

The VIC-20 has been around for just over two years. Some time passed before the machine was taken seriously. Now it has become a firm favorite of many users, writers, and teachers as a highly versatile piece of equipment.

Shortly before Christmas in 1982 the VIC-20 began to appear in department stores. I found myself chatting to a salesman who remarked that he had sold dozens in the space of a few days but that very few purchases had been made without a few game cartridges being purchased at the same time. Three months later, the same salesman remarked that repeat purchases still seemed to be of game cartridges rather than of expansion memory modules or the Super Expander, although purchase of the Home Calculation Package seemed to be picking up.

Clearly the VIC-20 was being bought, largely as a game machine with only a few people realizing that it was also a real, highly versatile, and very useful computer. It seems that there is a dim, but happily growing awareness on the part of the general public that a computer is a better buy than a mere game machine, even though the reason why that should be true is not yet clear in their minds.

Speaking to PTA groups about computers in education, I often hear the remark, "We did the wrong thing; we bought a game machine; we should have bought a computer." This is after I have used both the VIC-20 and another small computer to demonstrate computer assisted information. Of those who already have a VIC-20 or similar small computer, few, it seems, realize the potential or understand how to get the machine to do many of its remarkable tricks.

The increase in interest in computers over the past four or five years has been nothing short of astounding. It might seem to many people that the computer is a new device. Like the microwave oven, the computer, in one form or another, has been quietly going about its business for many years, handling utility records, income tax and bank statements, and charge accounts—and often being blamed for the errors!

The Commodore line of computers is a popular one in many schools. Perhaps this is why you are interested in the VIC-20. Certainly the VIC-20 will support much of that which your child, or you, will do in school.

As a device for the home, the VIC-20 will enhance and add concrete experience to knowledge acquired about computers, developing a versatility on the part of computer science students and broadening their knowledge of the functioning of computers, the differences between computers, the advantages of this machine over that one, the differences in forms of BASIC, and the means of tackling problems in fresh circumstances. In fact, as an educator, I am firmly convinced that the computer is a means of allowing educational systems to begin educating once more and that a computer in the home is an essential adjunct to computer science courses.

The VIC-20, but not only the VIC-20, has the advantage that it is quite inexpensive and is well-supported in both the peripherals that can be attached to it and the software that is available to run on it.

In this book I assume that you wish to know how to control all aspects of the VIC-20, and my aim, quite simply, is to provide you with the means of realizing that wish. I have tried to be as "user friendly" in this book as is the VIC-20, leading you through what might at first seem to be a maze, in a congenial fashion.

I have also assumed that you will wish to do more than merely type in ready-made programs and will wish to learn how to develop programs that are personally pertinent. Thus, while there are lots of programs in this book, each serves as a beginning of a larger program of your own.

The ultimate aim of any good teacher is, I contend, to do him or herself out of a job with respect to any individual student. There has been a frightening increase in teacher dependency in recent years. Quite bright people have been led to believe that in order to know anything one must take a course!

This is just not true. Often the only worthwhile result of taking a course is the piece of paper that says you attended and passed some exam or other. I have seen some of these people, supposedly knowledgeable about computers or word-processing, blanche when confronted by a machine that is not the same as the one on which they trained. Such courses are quite useless and serve only to provide some teachers with a salary! After working through this book, you should be able to stand on your own feet, read, understand, and make use of material you see in magazines and other books.

This book is a teaching book, but it will only teach if you work through all of the programs, seeing how to modify them, introducing them into larger schemes, and using them as springboards for further efforts. The process of merely typing in ready-made programs rarely teaches anything, and most ready-made programs do not perform the precise function needed—or do not work! When you have worked through this book, you should have most of the necessary information and skills to allow you to make other people's programs work as you wish, or better still, produce your own.

No book can treat a subject exhaustively. There are aspects of the operation of the VIC-20 that I have seen fit to omit. If you become consumed with a burning desire to know these things, then my book has done its job.

There is really no such thing as a time limit in the learning process. Do not be tempted to compare your progress with the VIC-20 with that of someone else. Take your time, digress, dip here and there if you wish, explore, experiment, and dwell as long as you wish on any point. Soon you will be an accomplished computer user.

Chapter 1

The VIC-20 is available in many places as a package: you get the computer itself, which looks like half a typewriter, and a tape-recorder (called a Datasette). The box contains also some black wiring, a large and a small black box, and a small polished metal item with a tiny lever, which slides back and forth.

SETTING UP THE VIC-20

The tiny polished metal thing is known in the knowledgeable circles you have joined as a *TV/game switch*. There is a flat lead sticking out of one side. Remove the antenna leads from your TV and attach the flat sides from the game switch to the TV VHF screws. Then attach the TV antenna leads to the screws on the game switch.

One of the black leads that comes with the computer has a plug at each end that looks vaguely like a peeled metal banana. Appropriately enough these plugs are called *banana plugs*! Place one of them in the small socket marked computer on the TV/game switch. The other end is plugged into a similar socket on the smaller of the two black boxes, which is known as an RF modulator.

Now turn the computer around so that you can see the back of it. There are two round sockets in the center. The long lead from the RF modulator is plugged into the left-most of the two round sockets. There is a *key*, or pattern of holes, that ensures that the plug will fit only one way.

If you have the Datasette, plug it into the smallest of the three flat sockets, the one immediately to the right of the two round ones.

Now concern yourself with the larger of the two black boxes. There are two leads emanating from this: one plugs into the right-hand side of the computer. There you will see two plugs, one on either side of a small switch. The lead plugs into the socket to the right, or to the rear, of the small switch. The other lead plugs into the ac power outlet in the wall.

Check over everything to make sure that all plugs are where they should be. If you have the choice make sure your TV is set to use the external antenna. Now switch on the TV and tune to either channel 3 or 4. Next switch on the computer using the small switch on the right. If you are lucky, you will see the computer picture. This says:

```
*** CBM BASIC V2 ***
3583 BYTES FREE
READY.
```

If you don't see this, do not worry! Just change to the other channel (from 3 to 4 or vice versa). Now, if all connections and switches are right, you will get the picture.

You might need to adjust the fine tuning just a little and then tinker with the color and tint controls, but for all intents and purposes, you are up and running!

USING THE KEYBOARD

The keyboard is your means of communicating with the computer. The computer communicates with you via the TV. The communication can be in three dimensions as it were: numbers and letters, color, and sound. In addition to the numbers and letters, known as alphanumeric characters, the VIC-20 can also provide a variety of symbols: hearts, clubs, spades, circles, and a plethora of straight and curved lines. There are 8 foundation colors. More can be obtained by some advanced trickery.

The sound, via your TV speaker, is rather in the nature of an impolite burp cut short in its prime, but it suffices quite well for its purpose and does come with the computer without the need to buy any further equipment.

The keyboard behaves in much the same way as a typewriter keyboard. Type the numbers 1 to 5 and you will see those numbers appear on the screen. Now press one or other of the shift keys and you will see !"#$% appear instead. Now release the shift and type the letters Q, W, E, R, and T. On the screen you should get QWERT. If you press the shift key, however, you get a ball, a circle, and three different lines. On the front side of each of the letter keys, you will see a pair of symbols. Pressing the shift key allows you to use the right-most symbols.

Now press the left-most shift key and the Commodore key, the key with G to the left of the shift key, at the same time. All the uppercase letters have become lowercase and the symbols have become uppercase. Now, the keyboard behaves just like a typewriter. To get uppercase, just press shift by itself. Experiment with that for a few moments.

The screen is probably getting quite full, so full in face that the top line disappears and is replaced by an empty space at the bottom of the screen. The image on the screen is also getting a little messy. Press the C= and shift keys once more to revert to normal and then press the shift key and the key in the upper right-hand corner marked CLR HOME. All you should have left is the little black square blinking slowly on and off in the upper left corner of the screen.

Just to check that you have all the colors (there is no possibility that you won't have them, but it's nice to be sure), try the following: press the key marked CTRL (upper left), and then press each of the numbers 1 to 8 in turn, watching the blinking square (the cursor) while you do so. The cursor will change color. Now type a few letters, press the CTRL key and a new color, type a few more letters, press the CTRL key and a new color, and so on until you tire of it or fill the screen.

Press the shift and CLR HOME keys again to return the cursor to the *home* position. It is likely that you have something other than the original color and cannot remember what the normal color is supposed to be! Don't worry: the computer knows. Merely press the key marked RUN STOP (left-hand side) and the key marked RESTORE (right-hand side), and the computer is back to normal.

There are just a few more things we can experiment with before we get on to a program. At the right-hand edge of the keyboard you will note a key with two horizontal arrows on it, which is next to the key with two vertical arrows. Press it and hold it down. The cursor moves to the right until it reaches the edge of the screen then whips down to the next line and does it all over again.

Now press shift and do it all over again. This time the cursor retraces its steps. It will even pass over the word READY and continue until it reaches the top left-hand corner.

The key to the left with the vertical arrows does the same trick, but vertically. Try it. Press it by itself for down, and the shift key for up.

There is just one more key to play with before you start real programming, and that is the INST DEL key. Start by typing in a few letters, numbers, and so forth. Now press the INST DEL key. If you hold it down, you wipe out all the characters you typed. If you are careful, you need only wipe out those you want to delete. Quite obviously, the DEL part refers to delete.

Now type a few more letters, and then press the shift key and the horizontal cursor control, which you have just been using, once. The last letter you typed appears in reverse mode each time the cursor flashes. Now, still holding down the shift key, press the INST DEL key once. A gap appears between the last two letters you typed. INST therefore refers to insert. Now move the cursor to the right, past the last letter you typed.

A few moments playing with this will repay your efforts, for there will be countless times that you will wish to change something. Complete facility with the cursor controls and the INST DEL key will help you to avoid the need to retype large lines of material.

Should you find the cursor doing anything strange or unexpected then merely press the shift and CLR HOME keys and all will be well.

In order to delete one letter from a word you must position the cursor over the following letter then press the INST DEL key without touching the shift key.

CAPSULE REVIEW

All cables and leads that come with the VIC-20 and Datasette are designed to fit one spot and one spot only. The smaller of the two black boxes the RF modulator, has a sliding switch to change the computer's transmission to the TV from channel 3 to 4.

It is wise to make all connections between computer, tape, and TV before plugging into the ac outlet and switching on the machine.

It is possible that the TV set will need some fine adjustment to tuning, color, and tint.

The computer prints uppercase (letters) on the screen in its normal state. The shift keys produce punctuation and modifier signs from the numeral keys, but graphic symbols from the letters and some other keys. The graphic symbols produced are those printed on the right/front of each key.

The Commodore (C=) and shift keys pressed together produce lowercase letters. The shift keys then operate by producing uppercase letters.

The C= pressed alone reaches the graphic symbols printed left/front of each key.

The return key produces a *carriage return* and must not be confused with the return instruction used in programs. The latter must be typed from the keyboard.

The CLR HOME key operates by moving the cursor to the top left-hand corner without erasing material from the screen when the keyboard is in normal mode. It moves the cursor and erases the screen when used with the shift key.

The six colors and black and white are reached by pressing the CTRL key and the appropriate number key.

The INST DEL key removes material from the screen from right to left and can be held down for repeat action. When the shift key and the INST DEL key are pressed at the same time, a space is created for the insertion of a character.

The two cursor control keys move the cursor down and right in the normal mode and up and left when used in conjunction with the shift key.

The cursor keys and the space bar can operate in the repeat mode when they are held down.

Chapter 2

In this chapter you are going to write a simple program, modify it, and then save it to tape. That might not seem to be much at first sight, but you will, in fact, demonstrate considerable control of the computer.

A variety show or concert offers a sequence of events that take place in a precise order. A few of the events are not announced, such as raising the curtain, dimming the lights, getting the performer to walk on stage, sit down, stand up, take a bow, and walk off. These things are obvious. They must be prepared but are taken for granted by the stage crew, performers, and audience alike.

Computers are relatively dumb. Everything you wish them to do must be prearranged. Some of the prearrangement has been made by the people who programmed the 6502 chip the VIC-20 uses. These prearrangements allow your prearrangements to be carried out successfully.

In much the same way that different sections of a show are itemized and numbered on separate lines, each activity the computer performs is placed on a numbered line. (You will meet a method of including more than one activity on one line later). It has become the habit of programmers to number program lines at intervals of ten so that if a new line needs to be inserted, there is room to do so.

WRITING YOUR FIRST PROGRAM

Type the following:

```
10 PRINT "PLEASE TYPE YOUR NAME"
```

Don't forget the quotation marks, which are obtained by pressing the shift and 2 keys simultaneously. Check it to see that it is all quite correct and then press the return key. Your line is now installed in the computer's memory and the cursor is at the beginning of the next line.

Now type this line:

```
20 INPUT Z$ ($ is a shifted 4. Remember to press the return key)
```

And now the last line:

```
30 PRINT "HELLO"; Z$ (Press return)
```

Now clear the screen (shift and CLR home) and type the word RUN and then press the return key. Do what the computer tells you to do and then press the return key.

Your ultrapolite VIC-20 has not only accepted your name but has labeled it Z$ and then printed it in place of Z$ on the screen.

The word READY indicates that the computer is waiting for you to do something else. If you type RUN once more (don't forget the return key), the computer will do it all over again. Try someone else's name, but wait a moment before you call the neighbors in to show them!

The screen might seem a little cluttered. You can do something about that by adding a couple of lines, as follows:

```
22 ? (press the shift key to get the question mark)
23 ? (Always, but always, press the return key after every line!)
24 ?
```

You have added three lines each with a question mark. Clear the screen using the shifted CLR HOME key and then type the word LIST. Enter it by pressing the return key.

There is the program with the new lines added. The question marks have disappeared and in place of each one is the word PRINT. You see, the ? is a shorthand symbol that the VIC-20 understands to mean print when it is used as a statement after a line number. Don't worry that all your question marks will come out as the word print.

Now run the program by clearing the screen and then typing RUN. You can see that the computer has left some empty lines. Add a few more numbered lines containing the ? between 24 and 30, and see more empty lines appear. The print statement by itself allows some measure of control over the way material appears on the screen.

Now you are going to change some of the program. Type the following:

```
30  IF Z$="JOHN" THEN PRINT "HELLO DAD"                 (RETURN)
40  IF Z$="JO" THEN PRINT "HELLO MOM"                   (RETURN)
50  IF Z$="RICHARD" THEN PRINT "HI DICK"                (RETURN)
60  IF Z$="CHARLES" THEN PRINT "HOWDY
CHUCK"                                                  (RETURN)
70  IF Z$="MICHAEL" THEN PRINT "SEE YOU MIKE"
                                                        (RETURN)
80  IF Z$="GRACE" THEN PRINT "HI SIS"                   (RETURN)
```

Clear the screen and then run the program. You will, of course, substitute the names of your family or any guests you might expect to show your computer to for those I have used.

Somehow the computer is able to distinguish between each of the names, but only those contained in lines 30 to 80. Any other names are ignored. Now call in the family to play with the program while you read the explanation.

The new lines contain the almost magical IF . . . THEN construction. Z$, the letter followed by the dollar sign, is called a string variable. A string in computer parlance is a collection of letters or numbers. A *variable* is a label that can have various meanings. The word *car* or *auto* is a variable in natural language because it can mean anything from a Volkswagen to a Rolls-Royce. The variable Z$ can be anything you like; you could even type 1234 in the first simple version of the program and the computer would respond with HELLO 1234, which just shows how weak-minded the computer really is!

Add the following line:

```
28  ? "HELLO "; Z$
29  ?
```

Now run the program. If you type a name that is not stored as a string, the computer will simply say HELLO. For stored strings it goes the whole hog!

While the family is busy playing with this, surreptitiously press the C= and shift keys. Be gratified by the ooh's and ah's as the VIC-20 displays the lowercase.

You will no doubt get questions like: "How do you get the colors?" "Will it talk?" "What's this button here for" The answers to these questions should be in the form of "Like this" (press the CTRL key and one of the numbers 3 to 6), "No," and "Because it would be in the way over here".

If you have worked diligently at getting to know your keyboard, you will have no difficulty dazzling onlookers with the speed with which you type RUN and press the return, runstop, and restore keys.

SAVING YOUR PROGRAM

The tape recorder should have a new tape in it. Just in case you are using an audio tape with a clear or colored leader, press the fastforward FFWD, button until you are sure there is magnetic tape in front of the head. It should take just a short stab at the button to do this.

Before you can save the program, you must give it a name. While it is possible to save a program without a name, it is unwise because the computer and you will have difficulty finding it again! So, let us call this program "NAMES". Type SAVE "NAMES". The computer responds with PRESS RECORD & PLAY ON TAPE. As soon as you do this the computer responds with:

```
OK
SAVING NAMES
```

As soon as the program is saved the computer prints the word READY. The tape recorder stops even though the record and play keys remain depressed.

The whole procedure is as simple as that. Now whenever you wish to access that program from tape you merely rewind to a point before the tape counter reading (it is wise to make a note of it together with the precise name of the program), type the words LOAD "NAMES" and press return.

The computer responds with:

```
PRESS PLAY ON TAPE
OK
SEARCHING FOR NAMES
```

As soon as it has found that program, it lets you know. When the program is loaded the familiar READY appears.

Clear the screen and then type L and shifted I. On the screen will appear an uppercase L and a quarter circle. Press the return key and the program will be listed on the screen. L and shifted I is the shorthand for list.

DOING CALCULATIONS

It will have occurred to you, no doubt, that a computer is so called because it can compute! It can handle the manipulation of numbers at high speed.

Clear the memory by typing the word NEW and, as always, pressing return! Now, without putting in line numbers, enter the following.

```
X=10  (RETURN)
Y= 5  (RETURN)
?X+Y  (RETURN)
```

The VIC-20 responds instantly with the result, 15. Now enter ?X*Z (* means multiply in computer jargon), and the computer responds with 50. Enter ?X/Y and the result is 2. You are using the computer in what is known as the *direct* mode.

You can change the values of X and Y merely by writing X= new value: Y= new value, all on the same line with a colon between. You could assign values to a large number of letters with a colon between each expression and then ask the computer to perform computations using any letters you select. However, in order to assign new values to previously used variables (letters), you must write a new expression.

A program that performs the same function could be written in such a way as to allow the user to change the value of a variable. The program is similar to the one you have just written for names except that you use line numbers.

Clear the screen of all the computations you have just performed. There is no need to type NEW because there is no stored program. Now enter the following lines:

```
10  INPUT "Value for X please"; X (note the semicolon)
20  INPUT "Value for Y please"; Y
30  ? X+ Y
```

Now run the program. No sooner have you given a value for Y than the VIC-20 prints the result. Before you can use the program again, you must type RUN.

There is a way around that problem:

```
40  GOTO 10
```

Now, each time the computer has completed the computation it returns to the beginning of the program and asks for further input.

Note the format of lines 10 and 20. You have saved a couple of lines, some effort, and some memory by putting the lines in this form. The alternative would be to type:

```
10  ?" Value for X please"
20  INPUT X
30  ?"Value for Y please"
40  INPUT Y
```

The Names program can be written in that form too:

```
10  INPUT "WHAT IS YOUR NAME";Z$
20  ?"HELLO ";Z$
```

This technique saves a lot of time. Note that a string in the form of a question following the word input does not need a question mark. The VIC-20 inserts it automatically, courtesy of the clever folk who programmed it in the first place.

Let us see if you can make the computation program a little more flexible. So far it will only add X and Y, simply because you have not told it to do anything else. You could change line 30 to read ?X*Y, and then the numbers will be multiplied. However, whatever function you ask the computer to perform by changing line 30 is fixed until you change the line once more. It is rather a nuisance to be in the middle of a long series of calculations only to have to list the program, change a line, and then try to continue.

Here is a solution; leave lines 10 and 20 and retype the following:

```
30  INPUT "+,-,*,/,";A$
40  IF A$="+"THEN 100
50  IF A$="-" THEN 200
60  IF A$="*" THEN 300
70  IF A$="/"THEN 400
100  ?X+Y
110  STOP
200  ?X-Y
210  STOP
300  ?X*Y
310  STOP
400  ?X/Y
410  STOP
```

The function of the program is obvious except for one thing, perhaps. For each operation selected, the computer is directed to a new line. It is a result of the statement on that line that the VIC-20 does its work. The program is a little long-winded to be sure, but you will have some idea of the further possibilities of the if-then statement.

Let us tidy up the program a little. For this you should make use of the cursor. List the program. You will note how the computer moves the listing up the screen for you. Now press the shift key and the vertical cursor key until the cursor is on line 40. Now, without using the shift key, move the cursor along to the first digit in 100. Without doing anything fancy like attempting to delete, just type the word PRINT. The number has completely disappeared and the word PRINT appeared in its place. Now continue with X+Y. Notice how line 50 is pushed down to accommodate the extra characters that have appeared in line 40. Press the return key and the cursor will go to a position over the 5 of line 50. Move the cursor along to 200 and type PRINT (you can use ? if you wish) X-Y. Do the same for lines 60 and 70 and then repeatedly press the return key until the cursor is down below the word READY at the bottom of the screen.

Now run the program by typing 46 for X, 67 for Y, and the * for the operator.

What's this? There are two numbers: 3082 and 113 underneath! 113 is the result of adding 46 and 67. Where did it come from? List the program once more. (Ignore the legend BREAK IN 310: it merely tells you that the program stopped at that line number).

When you pressed, the *, the computer did exactly what was expected. However, it then rolled merrily down the lines, ignoring line 70 because it did not meet the condition you laid down by entering *. It then encountered line 100 which said to print X+Y! It did just that! It then encountered the stop instruction left over from the previous form of the program. If you wish to experiment a little, just type the numbers 110, 210, 310 and 410 and press the return key after each. Now list the program. Those lines have gone! The new listing looks like this:

```
10  INPUT "VALUE FOR X PLEASE"; X
20  INPUT " VALUE FOR Y PLEASE"; Y
30  INPUT" +,-,*,/ ?";A$
40  IF A$="+" THEN 100
50  IF A$="-" THEN 200
60  IF A$="*" THEN 300
70  IF A$="/" THEN 400
100  ?X+Y
200  ?X-Y
300  ?X*Y
400  ?X/Y
```

Now run the program again. Use the numbers 46 and 67, and the same operator, *. The results look like this:

```
VALUE FOR X PLEASE?    46
VALUE FOR Y PLEASE?    67
+,-,*,/?    *
3082
113
-21
3082
.686567164
```

The first number, 3082 is the one you wanted. The next four are the result of lines 100, 200, 300 and 400.

This sort of thing happens all the time with computers if the program is not carefully assessed or *debugged*. Now you know that your incorrect utility bill is not a result (or at least, is unlikely to be) of a computer error but probably a result of poor programming or an unrectified bug.

To remove the unwanted lines merely type the line numbers and press the return key after each one. The program now finishes at line 70. In order to get the program to run again automatically, type as you did before, 80 GOTO 10

USING ROUTINES

In this chapter, just one more trick that your VIC-20 can perform will be explored.

It often happens that you need an effect to take place more than once; sometimes you need the effect many times in a given program. It can be very tiresome indeed to keep on typing in the same material at different points in the program, and so you can make use of *routines*. Routines are not something like print or if-then statements that form part of the BASIC computer language, but they are sequences of events that you build from BASIC statements and commands.

One way of getting the computer to repeat itself is to generate a loop. The vital ingredient here is the for-next statement. The computer is told to do something *for* so many times. Each time it does it, it checks to see if it has reached the required number of turns. If not then it does the *next* turn. It's rather like those games where a player is told to have three extra turns. The dice are thrown and the count is 1; the dice are thrown again and the count is now 2; again the dice are thrown and the count is now 3. If the player tries to have a fourth turn the other players say "Uh! Uh! No more!"

Here is a simple example: Type NEW then

```
10  FORI=1TO10     (in other words, ten turns called I)
20  PRINT I
30  NEXT
```

That's all the program consists of, but it does a surprising amount of work. Run it and see. Try changing the values of the two numbers in line 10. For example, change 10 TO 20.

Now change line 10 to read

```
10 FORI=1TO40 STEP 2
```

The computer merely prints the odd numbers.

Now change the first number to 2 using the cursor. Try changing each of the numbers, but don't make the last value of I too large or else you will not see what is happening.

The VIC-20 can count backwards just as well and as rapidly. Change line 10 to read:

```
10  FORI=40TO1STEP−2 (Yes . . . you can leave out the spaces!)
```

Again experiment with the numbers.

When you have grown tired of that type NEW, which erases the program from memory. Clear the screen and enter the following program:

```
10  INPUT" VALUE FOR A";A
20  INPUT" VALUE FOR B";B
30  FORI=ATOB
40  PRINT I
50  NEXT
```

This program obviously allows you to alter the numbers you put in and works on those. Change line 40 to read:

```
40  ?I:?I
```

All this does is to cause each number to be repeated.

The format of this line demonstrates another feature of the VIC-20 and that is the ability to allow you to put more than one statement on a line. The format for this is STATEMENT COLON STATEMENT. Move the cursor up to line 30 and then move it along until it is in the space after the letter B. Put a colon there and add

```
PRINT I: NEXT
```

Now erase lines 40 and 50 by simply typing the two numbers and pressing the return key after each one. If you have any problems here, it could be that you did not press the return key at the end of line 30 and then move the cursor down below the word READY.

The program is now tighter and, now that you have some expertise, is not difficult to read. In fact it is easier to follow the logic of a long program when you don't have to keep on going down to a new line too often. The whole loop is presented as one unit.

What if you wished to enter a list of names? Let's work up a program in stages. Enter the following:

```
10  INPUT "NAME PLEASE";A$
20  FORI=1TO10: ?A$:NEXT
```

This program allows you to enter a name and then prints it 10 times. It is interesting but not very useful unless you needed a whole lot of duplicates of something; a set of labels with your name and address to stick on envelopes for example. Just for fun, rerun the program but enter not only your name but also your address. The screen displays it neatly doesn't it?

If you want to enter a whole list of names, you have to make use of another statement, called DIM, which is short for dimension. The format for this statement is DIM followed by a variable (A, B, C and so on), followed by a number in parenthesis; for example, DIMA (20). This little statement says that there can be 20 items called A(1), A(2), A(3), and so on up to A(20). Each one is numbered automatically by the computer.

DIMA$(20) says the same thing but deals with strings. As the two things work in the same way, and as strings can cause confusion at first, let us deal directly with strings.

```
10  DIM A$(30)
20  INPUT"HOW MANY NAMES";N
30  FOR I=1 TO N: INPUT A$(I): NEXT I
40  FOR K =1 TO N: PRINT A$(K): NEXT K
```

When the VIC-20 asks how many names, put in a low number to start with. As each question mark comes up, enter a new name. When you have reached the limit the VIC-20 will print out the names in the same order in which you entered them. Line 10 defines the absolute limit to the number of names. If you try to enter more than thirty, the computer will display an error message after you have entered the first thirty. You can alter the limit by changing the number in parentheses, but to do this you must either enter a new line 10 or list the program and edit line 10 using the cursor, which can be bothersome.

There is another approach that you can use right now, but in order to show you, I intend to introduce a bug, so that you can see what a real bug looks like! Change line 10 to read DIMA$(N). The variable N should now be the same as the N that you enter as a result of the request on line 20.

Now run the program. The computer shows the request quite nicely, and you can enter the first name. No sooner do you press return however, than you see the following harrowing message:

```
?BAD SUBSCRIPT
ERROR IN 30
READY
```

List the program. Now you can see what is wrong. You have used the variable N as a *subscript* (that is what we call the numbers in parenthesis in a DIM statement) before you have assigned a value to it.

The solution is simple. Add line 25 and type DIMA$(N). Now type 10 and press the return key. Line 10 is now deleted and line 25 is in its place, but it now comes after you have assigned a value to N. Now the program runs well.

"All right" you say, "I have to put the names in alphabetical order for them to appear in alphabetical order"

No you don't! You can get the VIC-20 to sort the names for you—not with the program you have now, mind you, but you do have the makings of such a program. Sorting things into order is something the computer does so well and in such a variety of ways that it deserves a chapter to itself. The next chapter will do just that.

CAPSULE REVIEW

Lines are usually assigned using intervals of 10, at least in the early stages of a program. Each line consists of one or more directions to the computer.

Whatever you place between quotation marks will appear on the screen.

The input statement allows you to take part in the program activity. Whatever you type will be used by the program.

The print statement can be used alone to produce blank lines so that the screen presentation is clearer. The shorthand for print is? and for list is L shifted I. In fact most statements can be shortened by typing the first letter and shifting the second... but be careful of this as some statements cannot be shortened. A complete list of the shorthand is printed in the appendix of the VIC-20 manual.

User input can be evaluated by the computer. The simplest method is to use an if-then statement followed by some other statement.

Programs are saved to tape by entering the word SAVE. The computer prompts you with instructions and indicates when the process is complete.

A program is transferred from tape to the computer by entering the word LOAD followed, of course, with the name of the program as in the saving process. Again the VIC-20 prompts you and indicates when the process is complete.

It is wise to label programs with a short name, being careful to avoid odd spaces and punctuation marks. Make a note on the tape and on the contents card in the precise form that the name is written on the tape. Also make a note of the tape counter reading for each program. The VIC-20 will search through the entire tape and even list each program it finds on the screen, but the process can be tedious on a 60 minute tape!

Programs can be erased from memory in order to make room for new programs. The command is NEW. The new command cannot be used within the body of a program or else the program will erase itself.

The VIC-20 can be used in what is known as the *direct mode*; that is, without line numbers. However, only simple programs can be dealt with in this mode.

Computers handle numbers extremely well, in fact, although it may not be obvious, it can only handle numbers! Such things as letters, dollar signs, and graphic symbols, in fact anything you can see on the keyboard and anything you care to design is translated into a numerical equivalent by the internal program of the computer.

The computer can make distinction between different things. For example, it can distinguish between + and – and act appropriately if correctly programmed to do so. The if-then statement is a means of causing the computer to make distinctions.

The computer can be caused to repeat a given activity any number of times, even for ever! For-next statements are the means of accomplishing repeats by setting up loops. The word *for* is followed by a variable that is then given a value from one numerical point to another. Both points can be specified at that moment or can be taken from another area of the program. The points can be arranged from low to high or in reverse. The count can be made either consecutively or by steps of any declared size. Step size is indicated by the step statement and can be a positive value if the count is forward or a negative value if the count is backward; for example, 40 TO 10 STEP −2.

Strings are denoted by a dollar sign after the variable label; for example, A$, BB$. Strings can consist of a number of characters and spaces or even whole sentences. They are contained in quotation marks.

An array can be composed of strings, or it can be a numerical array. Both can be assigned to a variable name and can use the same variable name repeatedly because each item is assigned a number. The DIM statement is used to indicate the maximum number of items that can be in the array. The statement takes the form DIMA(N), where N indicates the number of different sets of digits that can be called A for numerical arrays; and the form DIMA$(N) for strings.

The number N can be declared either within the statement itself or can be called from elsewhere in the program. In the latter case the value of N must be declared before the DIM statement is used or else the program will crash.

Chapter 3

Although I have made certain decisions regarding what material I present and when, it must not be thought that there is a totally fixed heirarchical structure to the information I am presenting.

You might be anxious to get on and explore some of the other features of the VIC-20 such as color and sound. You are quite free to do so; indeed it is likely that you will have experimented with color in the short programs presented so far.

As promised, this chapter will deal with matters concerning sorting of material, as well as some other concepts! You may wish to skip it for the moment and come back to it later.

SORTING ROUTINES

Type in the following program:

```
10 DIMA$(20)
20 INPUT"HOW MANY NAMES";N
30 FORI=1TON
40 INPUTA$(I):NEXT
50 PRINTTAB(8)"NOW SORTING"
60 FORI=1TON-1
70 IFA$(I+1)>=A$(I)THEN120
80 B$=A$(I+1)
90 A$(I+1)=A$(I)
100 A$(I)=B$
110 GOTO60
120 NEXTI
130 FORI=0TON:PRINTA$(I):NEXTI
```

When you run this program for the first time, put in names that all begin with the same initial, such as John, Jerry, James, June, Jack, Jerrold, and Jervis, up to the total you indicated in response to the question. (Note the 20 in line 10.) Don't put too many names in on the first time around just in case there is an error in your typing, (which is unlikely, but you never know!) See how nicely the program sorts the names for you?

Before considering more sorting programs, you should tidy this one up a little. Begin by adding a print statement to line 130. List the program, bring the cursor up to the line, move it along until it is over the P of PRINT; then press the shift and INST DEL keys twice. Now enter a ? and a : and then move the cursor to the end of the line and press the return key twice. This brings the cursor below the word READY. Now run the program. The names are neatly separated on the screen. It would be nice if you could get rid of all that input, however, and just show the result. Add the following:

```
125  PRINT CHR$(147)
```

CHR$(147) is a code recognized by the VIC-20 to stand for CLR HOME. Run the program to see what happens.

There will be occasions when you will see the awful line

```
?SYNTAX ERROR IN LINE --
```

This will often mean that you have a semicolon in multistatement lines where there should be a colon. It is akin to a syntax error in normal writing. The expression FORI=1TON might seem like a syntax error to you, but it doesn't actually mean 1 ton; the VIC-20, or any other computer for that matter, is incapable of weighing anything!

In order to understand what the computer is doing in this program, it might help to put in a few extra lines. You can ask the computer to let you watch the magic from behind as it were, to see what is done with all those names between the time you enter them and the time they appear on the screen in alphabetical order.

```
120  PRINT A$(I)
122  NEXT I
```

When the program runs, it will print out each of its efforts to sort the information. It is best for illustrative purposes to use only four names for this exercise. Type them in and then run the program.

Did you see? Pretty darn quick, wasn't it? Try it again, but before you press the return key after the final name, poise your finger over the CTRL key. Press it as soon as you see the words NOW SORTING appear on the screen. If you remove line 125 for a run or two you can see the work and the result. Be sure to put line 125 back and allow line 120 to revert to its original form before you save the program however.

To change line 122 to 120 is very easy. Bring the cursor up to line 122. Move it over the second 2; then type 0 and press the return key. Bring the cursor back down below READY, and then type 122 and press the return key. Now you can save the program and amaze your friends by producing all sorts of lists.

How is it done? I spoke in the last chapter about how the computer can actually only deal with numbers. The letter A does not look like an A to the computer. To display an A on the screen, a series of zeros and ones, 8 columns wide and 8 rows deep, are used. A reasonable representation would be as follows

```
0 0 0 0 1 0 0 0
0 0 0 1 0 1 0 0
0 0 1 0 0 0 1 0  The zeros stand for blank bits
0 1 1 1 1 1 1 1  on the screen and the ones for
0 1 0 0 0 0 0 1  filled or set bits.
0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 1
0 0 0 0 0 0 0 0
```

Each character is assigned a number known as its ASCII (American Standard Code for Information Interchange) value: the value of B is greater than A; C is greater than B, and so on. The value of each character is listed in an appendix in the VIC-20 users' guide. That is how line 70 does its trick by assessing whether a name is greater or equal to another name. The shuffle continues until the numbers are in order.

The type of sorting method given here is adequate for small lists. Try changing line 10 to read DIM$(50) and enter 50 names. It will take some time even for the VIC-20 to sort these.

In Appendix A, you will find four other programs used for sorting. The line numbers are high so that they can be placed at the end of a program as a subroutine.

SUBROUTINES

What is a subroutine? It is a section of a program that can perform an operation repeatedly upon request. It differs from a for-next loop in so far as it performs the action only once each time it is used, whereas the loop has a counter that allows the action to be repeated a specified number of times without having to be asked each time. When the action performed by the subroutine is complete, the computer returns to the part of the program it was in before the subroutine was called.

An example is in order. Return to your first program, the one in which you put in a name and some sort of response comes out. This program will announce whether or not the name entered is the name of a family member. It is not a tremendously useful program but one that will provide some fun at a party.

```
10  INPUT"WHAT IS YOUR NAME";Z$
20  ?  "THANK YOU ";Z$
30  IFZ$="JOHN" THEN GOSUB 500
40  IFZ$="JO"THEN GOSUB 600
50  IFZ$="RICHARD"THEN GOSUB 700
90  ?  "DO I KNOW YOU?";Z$
100  STOP
500  ?"YOU ARE FAMILY"
510  ?"GET BACK TO THE TYPEWRITER";Z$
520  RETURN
600  ?"YOU ARE FAMILY"
610  ?" WOULD YOU LIKE SOME COFFEE?";Z$
620  RETURN
700  ? "YOU ARE FAMILY"
710  ? "HAVE YOU TAKEN THE TRASH OUT"; Z$
720  RETURN
```

This trite little program, which is incomplete, demonstrates how to send the computer to a new line according to some specific response on the part of the user. What then?
Add the following line:

```
100  INPUT "HOW OLD ARE YOU";Z$
```

Notice how the return statement causes the program to be picked up at the line following the appropriate if-then line. This is the whole point of a GOSUB. The program skips to another point and then returns to the point from which it left.

I don't recommend saving this program. Perhaps you would like to try something a little more interesting?

```
10  "Hello. My name is Victor. What's yours?"
```

```
20  INPUT Z$
30  ?:?"I'M VERY PLEASED TO MEET YOU";Z$
40  ? "WOULD YOU LIKE TO KNOW WHAT I CAN DO?"
45  INPUT "PRESS Y OR N AND THEN RETURN";A$
50  IFA$="N"THEN 5000
55  ? CHR$(147)
60  ? "I'M PLEASED ABOUT THAT"
80  ?:?:?"HERE ARE SOME OF THE THINGS I CAN DO"
90  ?:?:?"1.CALCULATE. 2.TEACH"
100  ?:?:?"3.KEEP FILES.4.SORT"
110  ?:?:?"PLEASE PRESS THE NUMBER THAT CORRESPONDS
     TO WHAT YOU WOULD LIKE TO SEE"
115  INPUT A
120  ON-(A=1)-2*(A=2)-3*(A=3)-4*(A=4) GOTO 1000,2000,3000,4000
1000  ?CHR$(147)
1010  ?"CALCULATION ROUTINE"
1990  STOP
2000  ?CHR$(147)
2010  ?"TEACH ROUTINE"
2990  STOP
3000  ?CHR$(147)
3010  ?FILE ROUTINE"
3990  STOP
4000  ?CHR$(147)
4010  ?"SORT ROUTINE"
4990  STOP
5000  ?CHR$(147):?"NICE TO HAVE MET YOU: Z$:GOTO 10
```

The program will be finished a little later. A complete version of this program can be found in Appendix N, program 43. For the moment let us see what you have. The VIC-20 introduces itself and calls for responses from the user. A menu of programs is listed and the user is encouraged to select one of the items by entering the appropriate number. This is a common way of doing this. You will find menus, however, that call different sections by means of code abbreviations. When you write a more complex program than this you might wish to try that.

The new statement is ON-GOTO. Quite simply, you can avoid the use of a great list of if-then statements by use of ON-GOTO. Where you send the computer to is up to you. I have selected line numbers in the thousands so that there will be plenty of room for the subroutines. For the moment you have only the title of the routines. Note the stop instruction, which prevents the program from running on to the next part. If you wish you could replace each stop in lines 1990,2990,3990 and 4990 with GOTO 10. This will automatically return the program to its beginning. However, it would be best to leave that for a little while or else something curious will occur.

The first thing you could put in this program is the sort routine discussed earlier. You must bear in mind, however, that the user may not be very familiar with the machine or any computer for that matter. You should provide some instructions.

```
4020  ?"YOU MAY ENTER UP TO 20 NAMES."
4030  ?:?"MEMBERS OF YOUR FAMILY; FRIENDS OR ANY OTHER
      NAMES."
4040  ?:?"WAIT FOR THE ? BEFORE TYPING, AND PRESS THE RETURN
      KEY AFTER EACH NAME."
4050  ?"YOU WILL HAVE TO TELL THE COMPUTER HOW MANY NAMES."
4060  FOR L=1 TO 10000
4070  NEXT L
4080  ?:?:? "PRESS ANY KEY TO CONTINUE"
4085  GET A$:IF A$="" THEN 4085
4090  ?CHR$(147)
4095  ?"GOOD."
```

Look at lines 4060 and 4070. All these lines do is tell the VIC-20 to hold its tongue and count silently up to 10000. It doesn't take long! The material is held on the screen while the user reads it. You may feel that 10000 is not long enough. Don't make it too long however, or the user will think something is wrong.

Now, continue with the sort program.

```
4100  DIMA$(20)
4105  ?"HOW MANY NAMES?"
4110  INPUT N
4115  FOR K=1TON
4120  INPUT A$(K)
4125  NEXT K
4130  ? TAB(8)"NOW SORTING-PLEASE WAIT"
4135  FOR K=1 TO N-1
4140  IFA$(K+1)>=A$(K) THEN 4165
4145  B$=A$(K+1)
4150  A$(K+1)=A$(K)
4155  A$(K)=B$
4160  GOTO 4135
4165  NEXT K
4170  FOR I=1 TO N
4175  ?A$(I)
4180  NEXT I
4185  FORI=1TO500
4195  NEXT I
4200  ?:?:?"HOW'S THAT!?"
4210  FORI = 1 TO 1000
4215  NEXT I
4220  ?"WOULD YOU LIKE TO TRY THAT AGAIN? Y/N"
4225  INPUT A$
4230  IF A$="Y"THEN 4105
4240  ?"WOULD YOU LIKE TO TRY SOMETHING ELSE? Y/N"
4245  INPUT X$
4250  IFX$="Y" THEN PRINT CHR$(147) : GOTO 90
4255  IFX$="N" THEN 5000
4990  STOP
```

Now you have something that looks like a real program. You can play with this quite merrily. You could even show it to someone else—as long as you made it quite clear that the teach, calculate and file sections were not finished yet.

As I remarked in the introduction to this book, my aim is to allow you to stand on your own feet. I'm going to put that into practice by suggesting that you write the calculation section. Use any tricks you can, including the ON-GOTO to select the multiplication, division, addition, and subtraction operations. Keep your program within the 1000 to 1990 line area, but otherwise be as fancy as you know how. It would be foolish of me to expect that this is the only book you will have read on computers. It is possible that you have already entered programs from magazines or from books. Use any features that you understand. Some features of these programs from alternate sources may differ from those in the programs described in this book. This problem is dealt with in Appendix B.

To conserve memory, you may wish to clean up portions of this program by producing multistatement lines. Be very careful of GOTOs in this case. To check on how much memory you have left, list the program and then, in direct mode, type ?FRE(0). The result should be 1925. This will probably not be enough to include a long calculation program, let alone a teaching or a file program. Don't worry, for there are still a few tricks up the VIC-20's sleeve!

When working on a long program it is wise to save as much as you have done. Save this one under the name DEMO.

The form is SAVE"DEMO" followed by the pressing of the return key. You are then instructed to press the RECORD and PLAY buttons on the Datasette. The little red light comes on to tell you that the program is being saved and the computer tells you so on the screen. The word READY tells you that it is done.

Can you be sure the program has been saved accurately? Yes indeed, for the VIC-20 allows you to verify the save with the verify command. Wind back the tape a way. Then type VERIFY"DEMO". You are instructed by a prompt to press the play button. The VIC-20 tells you when it is searching for DEMO and also when it has found it. It informs you that it is verifying the match between the program as it stands in the VIC-20 memory and on the tape. It also tells you whether or not everything is okay. A similar procedure is followed when you load a program. Now, if you can drag yourself away from the machine, you can rest easy in the knowledge that all your work is safe.

CAPSULE REVIEW

An array may be regarded as a sequence of things that have a label, or variable name, in common. The word *car* or *auto*, for example describes a type of passenger vehicle, which can exist in a variety of forms. The word *car* therefore can be regarded as a natural language variable.

A BASIC variable usually consists of either one or two letters or a letter and a numeral. A variable can refer, of course, to either a single item or an array of items. VIC BASIC allows for three types of variables: floating point variables, represented by one or two characters; integer variables, represented by one or two characters followed by the % sign as an indentifier; and string variables, denoted by one or two characters followed by the $ sign as identifier. In each case, the first character in a variable name must be a letter.

VIC BASIC allows arrays of up to eleven elements to be used (0-10) written in the form A(N),AA(N), B1(N), where N may be any number from 1 to 10. (There is usually no point in writing A(0).) An array that requires more than eleven elements must first of all be *declared* by using the DIM statement, which dimensions the array. Thus DIMA(20) allows twenty floating point items or elements; DIMA%(20) allows twenty integer elements; and DIMA$(20) allows twenty string elements. The number in parenthesis refers only to the number of elements and not to the number of characters in each element. Thus DIMA$(20) can refer to twenty different phrases or sentences, all of different lengths as long as the total number of characters in any single string does not exceed 255. Thus, DIMA$(20) can mean a total of 5100 characters, including spaces, of course, as long as you have enough memory or RAM.

The computer can actually only recognize numbers, and those only in the form of zeros and ones. Programming a computer by direct manipulation of this *binary code* is a long and tedious business and so other, more convenient ways have been devised, BASIC, a high-level *language* or communication medium (the letters of the name stand for Beginners All purpose Symbolic Instruction Code), has sets of instructions that stand for a whole collection, sometimes thousands of activities performed by the computer. Each instruction is rather like the word *run*, which you can use to stand for a great number of complex muscle movements.

Every character is assigned a value called its ASCII value. A list of these values is printed in an appendix in the VIC-20 users' guide. These values can be used to enable the computer to sort items into numerical or alphabetical order, both of which amount to the same thing as far as the computer is concerned.

Not only does each character have a value, but so do all the other keys on the computer keyboard. Many of them can be called into action during the course of a program by using the CHR$ feature in the form ?CHR$(N). Thus a direction to ?CHR$(42) will cause the * to appear on the screen. Likewise ?CHR$ (42);CHR$(63) will cause *? to appear. ?CHR$(42); CHR$(32);CHR$(42);CHR$(13);CHR$(42) will cause the * to be followed by a space (CHR$(32)), followed by another *, followed by a carriage return to the following line where yet another * will appear.

It is possible to call certain features into play during the course of a program; for example CHR$(147) will cause the screen to clear so that material can be presented in readily readable fashion.

Subroutines are rather like miniprograms. The computer can be directed to run or avoid subroutines according to certain conditions specified by the programmer—you!

GOSUB-RETURN is a combination statement each part of which may not exist without the other. A GOTO statement, on the other hand, can be regarded as a single entity, directing the program to move to a different point within itself and remain there until another GOTO is met. The GOSUB-RETURN cluster requires that the main body of the program is rejoined at the precise place at which the branch to the subroutine occurred. You cannot cause the computer to return to any other point, in fact.

ON-GOTO allows for the avoidance of large groups of IF-THEN statements. The simple form is as follows:

ON A GOTO (line no.),(line no.), (line no.)

If A is equal to 1, program control will go to the first line number. If A is equal to 2, control will go to the second line number, and so on for as many different values as you like or have memory to accommodate. The statement is highly useful in *menu* driven programs; the ones in which the user is allowed to select a particular feature or aspect of the program. At the end of this type of subroutine a GOTO returns the user to the original point in the program.

The for-next loop can be used to control the timing of the appearance of material on the screen so that material can be presented in a clear form and at a nonfrightening rate. The form of such a loop can be FORI=1 TO N: NEXT I; where N is a number proportional to the amount of time you wish the computer to wait before proceeding to the next line.

The form of the input statement that you will use in any program is conditional upon the type of variable required. INPUT A will wait until a number has been entered. INPUT A$ requires a string. However, because a numeral is also a string, it is possible to respond to the question WHAT IS YOUR NAME PLEASE? by entering 135748 . . . in which case you must tolerate being called by that label for the rest of the program. It is possible to use this feature if identities of users are required to be secure.

Unlike input, the get statement allows for the entering of one character at a time without need of pressing the return key. It is particularly useful in situations in which the material on the screen requires paced presentation. For this the GET is placed in a for-next loop.

A program uses up memory space roughly in proportion to the number of characters used, although DIM statements have a nasty habit of using space just by being there! You can check on the amount of memory left by typing ?FRE(0). The number that appears is the number of bytes remaining for use.

After saving either parts or the whole of a program on tape, it is possible, even advisable, to verify that the program has been saved correctly. The procedure involves winding the tape back and then typing the command VERIFY "   ", the name of the program being placed inside the quotation marks. The computer will carry out the process and report the result.

Chapter 4

This chapter will begin with a program that is apparently quite complex. The complexity arises not from any really new programming format—for-next loops should be quite familiar by now—but from the use of the poke statement.

USING THE POKE AND PEEK STATEMENTS

What exactly is poking? It is rather like my bookshelves. I tend to keep books and orchestral scores in the same place, not that they are necessarily in a specific order, so that I can either lay my hands on a particular volume in the dark or give precise directions to someone else. A computer does this too. Putting something in a specific spot in memory is called *poking*, and looking to see whatever might be lurking in a specific spot is called *peeking*.

Certain areas of memory are set aside for precise purposes. One of these purposes is the generation of modulated noise; we call it music. Now, to be frank, the sounds the VIC-20 make are not all that musical. A strangulated beep or even burp might be the best description, but that is better than no sound at all and, if you keep the sound at reasonable levels, it can be tolerated quite well.

The program given at this point is not really a musical one but rather one that produces sound effects: the noises of some small-engined means of transport starting up, going through the gears, and mercifully disappearing into the distance. It is possible that you might have to exercise your imagination a lot in order to hear this. The program is written in long form.

Whatever program has been in your computer up to now should be saved and then the word NEW typed and the return key pressed. Then type in the following program

```
10  POKE 36878, 15
20  FORI = 128TO165
30  POKE 36874, I POKE 36875, I POKE 36876, I
40  FORD=1TO100: NEXTD:NEXT I
50  POKE36874,0POKE36875,0:POKE36876,0
60  FOR I=128TO140
70  POKE36877, I
80  FORD=1TO20:NEXTD:NEXTI
90  POKE36877,0:POKE36878 8
100  FOR I=140TO195
110  POKE36874,I:POKE36875,I:POKE36876,I
120  FORD=1TO75
130  NEXTD:NEXTI
140  POKE36874 0 POKE36875, 0:POKE36876,0
150  POKE36878,5
160  FORI=175TO200
170  POKE36874,I :POKE36875, I:POKE36876,I
180  FORD=1TO200:NEXTD:NEXTI
190  POKE36878,2
200  FORI=195TO205
210  FOR G=145-155
220  POKE36874,I
230  NEXTG
240  FORY=168TO178
250  POKE36875,I
260  NEXTY    0
270  POKE36876,I
280  FORD=1TO300
290  NEXT D:NEXTI
300  POKE36878,0
```

Before running the program, make sure that all lines have been entered correctly; that is, as they are printed here. You can improve on this as you wish later.

Fundamentally the program consists of a series of nested for-next loops, each of which is quite self-contained. Note the order of appearance of the variables in the for part and in the next part.

There are five addresses or bookshelf slots that deal with sound in the program. Each address is a five-digit number, the first four digits of which are 3687. There are four *voices* on the VIC-20. Each of these voices is denoted by the fifth digit: 4 for voice 1, 5 for voice 2, 6 for voice 3, and 7 for voice 4. Voice 4 generates noise for sound effects and so on. The fifth address deals with volume, and the fifth digit is 8. Volume can be adjusted within the range 0 to 15, where 0 equals no sound. Thus the first line of the program sets the volume (36878) at maximum.

The second line deals with pitch, that is, how high or low the note is. When a violin string is set in motion by a bow it vibrates. If it vibrates quickly, the resulting sound will be high. If the vibrations are slow, or low frequency, the sound is low. Bear in mind that this has nothing to do with volume. How loud or quiet a sound is has to do with how far the string moves from one side to the other as it vibrates. Large movements give loud sounds and small movements, quiet sounds.

The frequency and thus the pitch of the sounds you can pursuade the VIC-20 to produce are in the range 128 to 255. The lowest notes produced by each voice are therefore indicated by the number 128 and the highest by 255. You can set things lower if you wish, in which case you might hear something rattling, or you might set the pitch higher than 255, in which case you will see the chilling message ? ILLEGAL QUANTITY ERROR! This is really a great pity for the simple reason that I have read that ultra-high pitched sound is an effective vermin and insect repellant.

All of this information brings us to line 20 of the program. This line allows the pitch to range from 128 to 165; not a great range, but it is plenty for our purposes.

Line 30 calls, or pokes, each of the musical voices 1, 2, and 3 to range through the pitches represented in I of line 20.

Line 40 introduces another for-next loop nested inside the first one. Loop number one is labeled with the variable I, and the second uses D for duration. Note the form on line 40. Loop D must be closed off before I can be closed off.

The next line turns the sound from the three voices right off. What we are doing here is saying "Don't vibrate!"

At line 70 the noise voice is set to each value in the range of frequencies established in line 60. It is switched off on line 90. On the same line the volume is set at a lower value and off we go again in a new gear.

I remarked at an earlier point in this chapter that the program was in long form. Now, while some of the lines are in multistatement format, which helps to reduce typing time, there is still a great deal of time spent typing in the same sets of numbers over and over again. This can become tedious and takes away from the time needed to think about the program.

Wouldn't it be a lot better if there were some means of shortening the process? Yes, it would, and there is a way. (Why else mention it?)

You can merely assign each voice and volume to a variable. In assigning variable names, the letters you use to denote the things you wish to vary, it is a good plan to use a letter that makes quite clear what it is you are varying.

Thus, S, which can stand for sound or speaker can be used for the voices, and V can stand for volume. Assign the variables as follows:

S1 = 36874; S2 = 36875; S3 = 36876; S4 = 36877 V = 36878.

Put these expressions in lines at the beginning of the program. Then, when you wish to call voice one, you can type **POKE S1, N** where N is either a specific number representing a pitch or a variable assigned a value in a for-next loop. The beginning of our bike program now looks like this:

```
10  POKE V,15           (volume set at maximum)
20  FOR I= 128TO165
30  POKE S1,I:POKE S2,I:POKE S3,I     (voices set for values of I)
40  FOR D=1TO100:NEXT D:NEXT I        (duration loop)
```

Three lines and a considerable amount of perspiration have been saved. The setting of S1, S2, and so on could take place on line 5 by making it a multiple statement line. Note that you must type the word **POKE** for each statement. One poke will not do for all of them; each must have its own poke.

The bike program is not really a musical one, although you can hear the individual notes near the end. This is because the frequency range is less for the final loop, and the duration, D, is greater. You could send the frequency up higher, but then the bike would sound as if it were going into orbit!

Music does not normally have great slides between one note and the next. This is an effect that singers, violinists, and wind instrument players use sparingly so that the effect seems the greater. Electronically generated music, as from synthesizers, the crystal trombone, or the theremin, uses the effect much more frequently.

The values for I in the program run from a low to a high number. You could easily reverse that. In addition, you can reduce the duration. Try this little program I call Bomb.

```
5  S1=36874:S2=36875:S3=36876:S4=36877:V=36878
10  POKEV,15
20  FORI=225TO195STEP-1
```

```
30  POKES1,I:POKE S2,I+4:POKE S3,I+20:POKE S4,I+25
40  FORD=1TO10:NEXT D:NEXTI
```

Play with line 40 varying the duration. If the neighborhood kids don't come running to see what game you are playing, you don't have it quite right! If said children do turn up, they will want to know what it is that is making the noise. Let's give them something!

```
45  POKE36879,8
50  FORI=7680TO8185STEP 23
60  POKE I,81
70  FORD=1 TO42:NEXTD
75  POKE I,32
80  NEXT I
85  POKES1,0:POKES2,0:POKE S3,0:POKE S4,0
90  POKE 36879,27
```

And then add:

```
8 ?CHR$(147)
```

(That's right, line 8. When you list the program you will find that the lines have been put in the right order)

CHR$(147) clears the screen; POKE 36879,8 turns the screen black; and POKE 36879,27 returns it to normal.

Now run the program. It follows the usual format of "You hear it before you see it!" Something could come of this you know! If only you could blend the sound and the image together, you might have something! Begin by moving line 45 to an earlier point in the program. Bring the cursor up to 45. Now type 9 and a space, and then run the cursor to the end of the line and press return. Bring the cursor down below READY and then type 45 and press the return key. When you list the program you will see the changes. When you run the program the change is dramatic. All we have done is to darken the screen before the noise begins.

Changing the value of the step to 18 brings a whole group of beasties across the screen from right to left. Increasing the value to 28, moves them from left to right. Experiment with hordes for a little while and then revert to the original value which was 23.

Each spot on the screen has a code. The codes start at number 7680, which is the top left-most spot, and increase to 7701 as you go across the screen to the right. You have reached the other side of the screen but are still on the top line. The next number, 7702, is the code for the left-most spot on the second line down.

If you were to change the value of the step in line 50 to 22, the little objects would come straight down the left-hand side of the screen in most uninteresting fashion. Now you can see why you must increment at the rate of 23. Line 50 directs the computer to print a ball at spot 7680, skip the next 22 spots and print the ball at spot 7703, skip the next 22 and print at spot 7726, and so on, until spot 8185 is reached.

As you have just discovered, changing the value of the step alters the spot at which the ball next appears. In other words, the placement of graphics on the screen is manipulated in much the same way as sound is manipulated. Sound is generated by picking spots on a frequency grid, and graphics are generated by picking spots on a visual grid, the screen.

Before the graphics are explored any further, let's bring your object to the ground in a more telling manner. I think it should explode—and explode it will.

```
100  POKES4,220      (S4 is the white noise generator)
105  FORC=1TO11
110  FOR L=15-(C*2)TO 8-(C*2) STEP -0.125    (reduce volume by in-
     crements of one eighth)
120  POKEV,L
130  FORD=1TO10:NEXTD
140  NEXTL:NEXTC
160  POKEV,0
200  POKE 36879,27
```

There are some gaps left in the line numbers just in case you want to alter or add anything later on. The variables used are as follows: C counts the number of explosions; L decrements the volume for each explosion; and D controls the duration of each.

You could experiment here by causing the explosions to be of varying intensity and duration but to do that will mean using some method other than loops. You can stash that thought away for the time being.

The explosion sounds fairly convincing on its own. However a visual effect to go with it will clinch the matter. First change line 105 to FORC=1TO4. (Simply move the cursor up to line 105; move it across until it is over the first digit in 11; type 3; and then press the space bar.)
Now add lines

```
125  FORX=8TO255STEP43
126  POKE 36879,X:NEXTX
```

and delete line 130. **WARNING!** Do not run this program if either you or anyone who might be watching is subject to epileptic seizures. The effect you have produced creates a series of flashes on the screen. Flashing devices, whether computer generated or not, can cause seizures. This warning is not confined to the VIC-20—any machine will produce the effect!

Line 126 pokes address 36879, which controls the colors of the screen and its border. The number after the comma will dictate the screen and border combinations. There are 128 of them, which are presented in Appendix E in the manual that comes with the VIC-20. Not all of them are useful for text as the contrast between the two shades is not very great. However, such combinations can be very pleasing under other types of circumstances.

In our example here, line 125 gives values for X from 8 to 255. Step 43 was chosen, not as a result of any great thought, but at random. Try other step numbers and watch the difference. Line 126 pokes the screen/border combination address for each of the values indicated in line 125. Line 90 could be removed as the return to normal takes place on line 200.

Now look at your revised line 105. This line is nothing more than a counter. It gives you just four rounds of the loop FOR C to NEXT C. The next line, line 110, is the one that gives the decay of each burst of the explosion. It says that the first explosion will begin at full volume (15) minus two-times the value of C from the previous line (15 minus 1 times 2 or 13) and goes down in volume by steps of 1/8 (0.125 is the decimal equivalent of 1/8) until it reaches a value equal to 8 minus 1 times 2, or 6. The next time around, C is equal to 2; the start figure for the volume is 11, and the final value is 4. The fourth and final round starts the volume at 7 and finishes at 0. To make the explosions shorter, change the value of 0.125 on line 110 to 0.5. This will also reduce the number of flashes. Now you can see what power there is in three statements in lines 125 and 126. A complete version of the Bomb program, with line 110 slightly altered, is found in Appendix N, Program 16.

Screen and border combinations do not have to flash in order to be effective. Consider our humble menu program. At the beginning of each subroutine, a simple poke to 36879 and a suitable screen and border combination will allow you to color code and thus highlight each section in a remarkable way. As with any effect, however, don't be too enthusiastic and change color like a chameleon or else you will merely unsettle the user. Be subtle!

So far the program has only dealt with sound effects. The VIC-20 can also play music if you tell it how. In addition, it will allow you to play tunes, using the numbers 1 to 8 or even the letters A to G.

USING DATA STATEMENTS

In order to produce the next program, which will not only play a tune for you but also let you play a tune to the VIC-20, you are going to learn about an extraordinary feature of the BASIC language. So far you have put odd bits of information that you needed to use in the body of the program. The new method allows you to organize the program so that the information you need to draw upon is put at the end of the program or sequence in what are called *data statements*. The data is used by *read statements*. The read statement comes fairly near the beginning of the program. The computer then scans the data line and acts upon it.

We'll have a little fun at the same time!

```
15 PRINT:PRINT"DO YOU KNOW THE TUNE  'BELIEVE ME IF
   ALL    THOSE ENDEARING YOUNG CHARMS
30 PRINT"I CAN PLAY IT";
40 PRINT:PRINT"YOU CAN PLAY IT TOO    USING
   THE NUMBERS  1 TO 8"
45 FORI=1TO5000:NEXTI  (gives you time to read it all)
60 PRINTCHR$(147)   (clears the screen)
100 PRINT"THIS IS HOW IT GOES"
110 PRINT:PRINT"THE NUMBERS ARE PLAYED
    IN THIS ORDER:"
115 FORI =1TO 1500:NEXTI
120 PRINT" 3 2 1 2 1 1 3 5 4 6 8 8"
200 S2=36875:V=36878
210 POKEV,12 (not too loud)
215 READP  (read statement)
220 IFP=-1THEN290
230 READD (another read statement)
240 POKES2,P  (turns on voice S2 for whatever
               note line 215 finds)
250 FORN=1TOD:NEXTN  (this measures the duration of the
                      note for whatever value line 230
                      finds)
260 POKES2,0   (turns S2 off)
270 FORN=1TO20:NEXTN  (small silent gap between the
                       notes)
280 GOTO215  (returns to get the next bit of data)
290 POKEV,0  (turns off the volume)
300 DATA207,200,201,200,195,600,
    201,200,195,400
310 DATA195,400,207,400,215,400,
    209,400,219,400
320 DATA225,400,225,800,-1
```

Run this portion of the program to see what happens. In this program you have a large loop that runs from line 215 to line 280.

You have two smaller loops within this large one at lines 250 and 270. Line 215 tells the computer to read the first number in the data statement in line 300. It finds the number 207 and goes on to read the next number, which is 200. The voice is then turned on to the pitch represented by 207 and lasts for the time it takes the computer to cycle 200 times. The sound is then turned off by line 260; there is a very short silent pause, and then the VIC-20 returns to pick up the next note.

If you look along the data lines which could have been condensed into one line in this case, you will note (sorry!) that each pitch value is followed by a duration value in even hundreds. Most of the duration figures are 400, which represents a fundamental beat; let us say that 400 hundred produces the equivalent of quarter notes. Eighth notes are half that long and so are represented by 200. Music notation is a little odd in that you have notes that can last for 1½ beats. You therefore add 400 and 200 to arrive at 600 for the 1½ beat notes.

The −1 at the end tells the computer that the tune is over. Each time the loop from line 215 to 280 runs, the program checks to see if there is a −1. If there is not, the program continues running. As soon as a −1 is found, however, the program branches to 290, which turns the whole thing off.

So far so good. When you have finished playing the tune carry on by entering the following:

```
330 PRINT:PRINT:PRINT"OK--YOU TRY IT. START ON
    NUMBER 3"
400 POKEV,8
405 POKES2,0
410 FORY=1TO8
420 READA(Y)
430 NEXTY
440 DATA195,201,207,209,215,219,223,225
450 GETA$:IFA$=""THEN450
460 Y=VAL(A$)
470 IFY=8THEN600
480 POKES2,0
490 FORT=1TO25:NEXTT
500 POKES2,A(Y)
510 GOTO450
520 POKES2,0
```

Check to be sure that you have typed everything in correctly and then continue typing this part, which should look familiar!

```
600 S4=36877:PRINTCHR$(147)
610 POKES4,220
```

```
620 FORC=1TO4
630 FORL=15-(C*2)TO8-(C*2)STEP-0.5
640 POKEV,L
650 FORX=9TO255STEP45
660 POKE36879,X:NEXTX
670 NEXTL:NEXTC
680 POKEV,0
685 POKES4,0
690 POKE36879,27
695 FORI=1TO2000:NEXTI
700 PRINT"GOT YOU!!!!!"
710 RUN
```

Now run the program on some unsuspecting party! The program from lines 400 to 520 should be fairly simple to understand. The volume is set and the voice is set at zero. Y counts from 1 to 8. READ A assigns the 8 values in the data statement to A(1) through A(8). The data statement gives the approximate pitch values for the scale of C major.

Line 450 uses a get statement, which merely scans the keyboard to see if a key is pressed. If a key is pressed, the numerical value of the key is transferred to Y, and that value, which is between 195 and 225, is poked to S2, and the note sounds.

Line 470 checks to see if number 8 is pressed. As soon as it is, the program branches to line 600, which produces the explosion and the flashes!

There is an extended version of this program in Appendix N, program 31, which leads the user on a little more by demonstrating and asking for other tunes with no disastrous effects whatsoever. The program could be a lot of laughs at parties!

CAPSULE REVIEW

If you wish to call your dentist you first look up his number and then dial. Only your dentist will be at that number and, presumably, you need his assistance with respect to your teeth. Your attorney or your plumber will have different telephone numbers. It is the same with the different functions your computer can perform. The functions can be called up for specific action by poking, rather than dialing, the appropriate number. Some of the numbers, which are called addresses, refer to the ability of the VIC-20 to generate sound via the speaker on your TV. These addresses are: 36874, 36875, and 36876 for the musical voices, 36877 for the white noise generator, and 36878 for volume.

After each number there is a comma followed by another number. For the voices, the numbers may range from 128 to 255 for audible sound. The numbers may also range on either side of those extremes, but the result will be inaudible for the lower limit and an error message for the upper.

Address 36878, which controls volume, requires a value between 0 (no sound) at 15 (full volume). You can, of course, adjust the volume at your set.

Although each of the three voices can only be set to pitches represented by the numbers 128 to 255, the same number will give different pitches for each of the three voices.

To test this type:

```
POKE  36878,5      (sets volume)
POKE  36874, 225     (sets the pitch for C)
POKE  36875, 225     (sets C one octave above the first voice)
POKE  36876,225     (sets C one octave above the second voice)
```

Each voice can sound over a three octave range. The total range that can be covered by the three voices is five octaves.

The frequency numbers poked after the comma bear no relation whatsoever to the natural frequency of a string or wind instrument. Middle C (on the piano) is approximately 256 cps. The C an octave below that is exactly half that figure or 128. A C on the VIC-20 is set by numbers 135,195,225 and 240.

Voices can sound together to produce harmony.

```
POKE  36878,5
POKE  36874,225
POKE  36875,207
POKE  36876,175
```

This produces a triad or chord of C major. Voice 36874 produces the lowest note of the chord even though it has the highest figure or modifier after the comma!

Using voice 36875,204 instead of 203 produces a triad or chord of C minor.

Frequencies can be made an integral part of a for-next loop and can be made to range from one pitch to another in a glide up or down. The step size between frequency numbers can be controlled by the step instruction in a for-next loop.

Volume can also be adjusted to either fixed values or constantly changing values during a program.

There are a number of ways of producing the glide. The following programs are examples:

```
10  POKE 36878,5
20  FORI=195 TO225
30  POKE36875,I
35  FORX=1TO 30 :NEXTX
40  NEXTI
```

```
10  POKE 36878,5
15  INPUT Q
20  FOR I=195 TO 225 STEPQ
30  POKE 36875,I
35  FOR X=1 TO 20:NEXTX
40  NEXTI
```

```
10  POKE 36878,5
15  INPUT Q
20  FOR I =195 TO 225 STEPQ
30  POKE 36875,I
33  POKE 36874,I+7
35  FORX=1 TOI/2:NEXTX
40  NEXT I
```

```
10  POKE 36878,5
15  INPUT Q
20  FOR I=225 TO 195 STEP-Q
30  POKE 36875,I
33  POKE 36874,I+7
35  FORX=1TOI/2:NEXTX
40  NEXT I
```

A generalized program might read

```
10  V=36878
15  INPUT Q
20  FOR I=(no) TO (no) STEP ±Q
30  POKE V, I ± (no)
40  POKE 36875, I
50  FOR X= 1 TO (no):NEXTX
60  NEXT I
```

The two no's in line 20 represent frequency numbers of your choice. The ± indicates that you can choose STEP Q or STEP−Q. In line 30 you can add or subtract a number of your choice. In line 50, you can control the duration of each tone by inserting a number of your choice where the (no) is.

Experiment with various sound effects and save the good ones on a sound effects tape. Experiment with the voices through 36877.

In the same way that sounds are provided by poking the appropriate address, so the graphics and colors of the screen and the border can be brought under your direct control. Color control is accessed by poking address 36879 with a suitable number indicating the color combinations after the usual comma.

Any of the graphic symbols can be placed at any point on the screen that you wish. They can be moved around the screen by changing the screen code. Screen codes range from 7680 to 8185 moving in lines across the screen.

In addition, graphic symbols can be manipulated in various colors by poking the color memory map addresses. These correspond to the screen codes but use the numbers from 38400 to 38905 followed by a numeral from 0 to 7 to indicate the required color.

The presentation of material can be greatly enhanced by the use of various border and screen color combinations.

It is possible, courtesy of BASIC, to isolate certain information from the main body of a program by means of data statements, which are then read from the program.

For-next loops can be nested inside one another as long as the NEXT order is the exact reverse of the FOR order.

Loops can be used as timers, allowing material to be presented on the screen a portion at a time, giving the user time to read each portion. `FORD=1 TO 1000:NEXT` gives a duration of approximately one second. Such timing loops can be used as the second item of data in read-data combinations to time the occurrence of sounds, providing for a rhythmic presentation and thus a musical result.

Chapter 5

At the beginning of this book, you played with the keyboard. Since then you have practiced with enough programs for you to become quite proficient at finding various things on the VIC-20. You have discovered that each character has a code and that this code can be used to produce the character at will.

CONTROLLING THE DISPLAY

For example, you have used CHR$(147), which to the VIC-20 is as valid a character as, say, the letter G, although 147 is the code for clearing the screen.

You can find out the code for any character by typing ASC("x"). Type ?ASC("G") and the number 71 should appear. Type ?ASC("X") and the number 88 appears.

Now type ?CHR$(88). The letter X appears. Try one or two more, all in direct mode, and then type ?CHR$(14). Everything on the screen, except for the numbers and the parenthesis, has turned into lowercase characters ?CHR$(142) puts the whole thing right again.

Clear the screen and type the following still in direct mode:

```
?"
```

After the quotation mark, press the down cursor key three times. The result on the screen is an inverse Q. Now press the right cursor key three times. The result this time is an inverse square bracket. Now type a short message such as HELLO. Now do the same thing using the two cursor keys, and then type another message. Close the quotes and press return. The version I have on my screen looks like this:

```
HELLO
          JOHN
```

Now you might think that it is possible, in the direct mode, to print part of the message in uppercase, type CHR$(14) and then finish the message. Not so, I'm afraid. The whole thing comes out in lowercase.

You will recall that if you press the C= key and shift at the same time you put everything into lowercase. Uppercase letters are achieved by pressing shift as with an ordinary typewriter. Press the C= and shift key. You are now in what is called the *text mode*. Now type the following:

```
10  ?"What is this?"
20  ?" (unshifted [CLR HOME] [cursor down]× 3) What is this?"
```

Now you will see that the unshifted CLR HOME key has produced a lowercase inverse s, and the cursor down key, three inverse letter Q's. Run the program and you will see:

```
What is this?
What is this?
```

There is a clear line between the two questions.

Line 20 could be altered to include a couple of cursor right instructions to move the second question over a little.

```
What is this?
  What is this?
```

With clever and cunning use of these keys, material can be displayed at any point on the screen, thereby enhancing the presentation, making things easier to read, and even highlighting portions of the text.

When you are exploring possibilities, do bear in mind that if you overdo things in a program, the result often detracts from the overall effect. Don't do things just because they are possible. This is a hideous crime known as *conspicuous computing*, a term coined by

Prof. Ronald Ragsdale of the Ontario Institute for Studies in Education. Keep things simple.

Store the information you have just acquired in some neat corner of your mind for use later. Now you are going to do something really clever.

PRODUCING YOUR OWN CHARACTER SETS

The character set on the VIC-20 is quite neat and tidy in both upper- and lowercase. and you can use upon 62 other symbols beyond those on the shifted numerals. However, it is possible that you may wish to have a different set of symbols or even another character set to play with.

In order to do this you must poke some more and even do a little peeking. So far you have poked something into a specific address. Peeking is the process whereby you can look at a specific address to see what is there. Your purpose is often to move something from one address to another.

Your first task is to make sure that the new address will not be disturbed and will be clear for use. You can do this with the following lines:

```
1  POKE 52,28:POKE 56,28: CLR
2  FOR I=7168 TO 7696:POKE I,PEEK(I+25600):NEXT
3  POKE 36869,255
```

Now you are ready! You have prepared 512 bytes of memory. If you type ?FRE(0), the number 3071 appears. The VIC-20 thinks that it has only that amount of memory to play with! The other 512 bytes are ours to do with what we will!

If you look very carefully at a character on the screen you will notice that it is made up of a series of dots, rather like those "newspapers-in-lights" one see in cities. Each character is made up of dots in a square which is 8 dots by 8 dots. Figure 5-1 shows the matrix of squares. By placing dots in the squares, a pattern that corresponds to the letter, numeral, or other symbol you wish to produce can be formed. To produce the letter I, dots are placed in squares 3 through 7 on the top row, in square 5 for the next 5 rows, and in squares 3 through 7 for the 7th row. Columns 1, 2, 7, and 8, and line 8 are left free so that there is some space left around the letter.

The matrix is rather like a box of little lights. Those lights that correspond to the dots that form the character are lit, and the others where there are no dots.

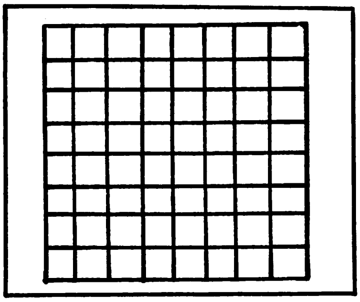


Fig. 5-1. The 8×8 matrix within which characters are created.

The dots correspond to bits which are either off or on. Binary code is used to indicate whether or not bits are set (turned on). 0 is used for off and 1 for on,

The matrix for the letter I you have produced looks like this in binary code.

```
0 0 1 1 1 1 1 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 1 1 1 1 1 0
0 0 0 0 0 0 0 0
```

This is a highly graphic representation of the letter I!
A letter C would look like this:

```
0 0 0 1 1 1 0 0
0 0 1 0 0 0 1 0
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 1 0 0 0 1 0
0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0
```

Now, you cannot just transformer these binary numbers directly into the computer. You must first of all translate them into decimal equivalents.

"But," you will say, "don't computers understand binary better than decimal?"

You are so right, but high-level languages like BASIC go to the trouble of doing the translation so that mere humans do not have to think very hard. This is called "being user-friendly." Thus, when you want to use binary numbers you have to think very hard in order to translate them into decimal so that the BASIC can translate them all back again into binary. Don't worry about it. This section will give you a chance to get to know binary code, if you don't already, that is.

Binary code is read most conveniently from right to left. Each digit is worth twice the value of its neighbor to the right. Thus the value of each of the eight digits is as follows:

128, 64, 32, 16, 8, 4, 2, 1

But this is true only when the digit is a 1! If there is a zero at a point, that point is worth nothing.

Examples: 00000001 = one, plus nothing for all of the others.
00000101 = one, plus nothing for the next place, plus 4 for the next and nothing for the rest: 1+0+4+0+0+0+0+0 = 5

The decimal numbers used to form the I are determined like this:

```
00111110 = 0+2+4+8+16+32+0+0 = 62
00001000 = 0+0+0+8+0+0+0+0 =    8
00001000 =                      8
00001000 =                      8
00001000 =                      8
00001000 =                      8
00111110 =                     62
00000000 =                      0
```

The decimal equivalents of each of the lines in the letter C, reading from top to bottom, would be: 28,34,64,64,64,34,28,0. This is exactly how you would enter the information into the computer, using a data statement to do so. Thus, I would be written as

```
DATA 62,8,8,8,8,8,62,0
```

and C would be written as

```
DATA 28,34,64,64,64,34,28,0
```

These are the vital statistics, as it were, of each of those letters.

So far, so good! You are now going to change one of the letters in the VIC-20 character set into something else. You will not disturb the VIC-20 by doing this, and you will not lose the character set that the VIC-20 uses. You are only borrowing it!

Type in the following line:

```
10  FOR A1=7176 TO 7183: READ A: POKE A1,A: NEXT
```

Then this line:

```
20  DATA 28,20,34,62,65,65,65,0
```

Now type RUN and watch the A's change shape.

A much more dramatic display can be observed if you retype line 20 as follows:

```
20  DATA 98,35,20,8,20,98,99,0
```

This time you have produced an X in place of each A. The word DATA thus becomes DXTX and READ becomes REXD. Don't worry! The VIC-20 still thinks it sees A's!

Try this one:

```
20  DATA 10,28,38,47,49,81,113,28
```

This produces an old German-style letter H. In order to get it all in the matrix, you had to use the bottom line.

Try just one more and then you can get down to work:

```
20  DATA 8 12,79,9,9,23,98,0
```

There you are! Your VIC-20 can write in Chinese! A sample of more Chinese characters is given in the following very short program.

```
1  POKE52,28:POKE56,28:CLR
3  FORI=7168TO7696:POKEI,PEEK(I+25600):NEXT
5  POKE36869,255
10  FORA1=7176TO7183:READA:POKEA1,A:NEXT
20  DATA4,9,127,20,56,2,97,0
30  FORB1=7184TO7191:READB:POKEB1,B:NEXT
40  DATA62,4,9,127,8,24,8,0
50  FORC1=7192TO7199:READC:POKEC1,C:NEXT
60  DATA28,20,20,20,20,38,67,0
70  FORD1=7200TO7207:READD:POKED1,D:NEXT
80  DATA8,60,42,43,72,8,24,0
90  FORE1=7200TO7215:READE:POKEE1,E:NEXT
100  DATA8,9,127,8,8,10,62,0
```

The Chinese language in written form poses a problem in as much as there are around 48000 characters! Rather more than one can fit into the VIC-20's memory or accommodate on the keyboard. However, Chinese characters are made up of a variety of combinations of strokes. Indeed, a dictionary lists the characters in order by the number of strokes. It would be possible to program the fifteen major strokes and build up the character in the same fashion in which they would be formed by brush or pen. The characters would be very large but that is not a drawback. Why not try it?

Program 13 in Appendix N produces a complete set of Hebrew characters. The problem here is that the computer writes from left to right . . . and Hebrew, like Arabic and Persian, is written from right to left.

It might be a good plan to save lines 1 to 10 in this program as a short utility. This will save you from having to hunt through all your books each time you want to use it. For each character you wish to change, there must be a different variable. Thus for the first character use A1 and A as in the sample line 10. The next character will use B1 and B.

The Hebrew character set is written with each data line following the for - next loop containing the read statement. This is done for ease of reading the program and keeping track of what is happening at each point. It is, of course, usual to place data lines at the end of a program.

If you have the Chinese character program in the VIC-20, run the program to change the A's, clear the screen, and then type a few A's. Now press C= and SHIFT as if to produce lowercase. What you see on the screen is garbage. It is not even the character in reverse. Just don't try typing lowercase Chinese!

Each character takes up eight memory locations. You are storing your new characters starting at location 7168. The first character is therefore residing at locations 7168 to 7175. The next character starts at the following number, 7176, and ends at 7183. The rule is simple: each character is location 7168 plus the character code number times 8. Thus the left square bracket start location is 7168 + 27*8 = 7384. Its end location is the start location +7.

With this information, you can find the start location of any symbol available from the keyboard and use the corresponding key as the trigger for the new character or symbol. You can even use @& $ % and the English pound sign, but, whatever new shape is produced the computer will still think it is the original.

For example, the $ (dollar sign) is used as the modifier for strings as in A$, B$, and so on. You might indeed set up the following lines:

```
20  FORI2=7392to7392+7:READ I: POKEI2,I :NEXT
25  DATA28,54,124,120,124,62,28,0
```

Now whenever you press the pound sign key you get a very familiar figure, to be seen stalking arcades and even homes!

If however, you change line 20 to read:
20 FOR I2=7456 TO 7456+7 etc., you find that it is now the $ that has changed. A printout of a program that showed DIMA@would look exceedingly odd, but the computer would read it as plain old DIMA$

The Hebrew program, program 13 in Appendix N, uses A1, B1, C1, and so on in each for-next loop merely to make it clear which letter is being used for each character. Typing that key will cause the appropriate Hebrew character to appear.

Now that might be just a little awkward if you wish to use Latin script as well. However, you do have a total of 64 symbols and letters available to you. You could confine your new script or character set to the locations for those characters with codes from 27 to 47 and from 58 to 64, giving a total of 26. In this way, you could have your normal Latin script, the new font (an Italic or Gothic?) and the numerals. It would be necessary to ink the new font on small pieces of tape which are then stuck to the keys in order to spare your memory!

In addition to the Hebrew character set you will find the Russian character set in program 14 in Appendix N.

The data printed below is for the Greek character set. There are two ways of going about using it. You can load the Hebrew program, and then change each data line to those you see above in the same order. Then save the new program. Alternatively, you may, of course, set up a data file on tape or disk and read each line in; however, this might prove to be more trouble than it is worth.

```
10  REM GREEK CHARACTER SET
12  REM IN ALPHABETICAL ORDER
1000  DATA0,0,24,37,66,70,105,0
1010  DATA28,34,34,62,34,33,124,0
1020  DATA49,73,6,8,16,32,0
1030  DATA28,18,8,52,66,66,60,0
1040  DATA24,32,64,120,64,32,24,0
1050  DATA10,4,4,8,14,2,6,0
1060  DATA44,82,18,18,18,18,2,0
1070  DATA28,34,65,127,65,34,28,0
1080  DATA0,16,16,16,16,20,8,0
1090  DATA0,38,40,48,40,36,34,0
1100  DATA16,8,20,20,18,33,33,0
1110  DATA0,0,36,36,36,54,40,32
```

```
1120 DATA0,0,36,36,36,40,48,0
1130 DATA19,28,32,28,32,60,2,12
1140 DATA0,24,36,66,66,36,24,0
1150 DATA0,0,63,82,18,18,18,0
1160 DATA0,24,36,36,60,32,32,32
1170 DATA0,63,68,66,66,66,60
1180 DATA0,0,62,72,8,8,8,0
1190 DATA0,0,70,34,34,34,28,0
1200 DATA16,16,56,84,84,84,56,16
1210 DATA0,66,34,20,8,20,98
1220 DATA8,8,73,73,73,73,62
1230 DATA0,0,42,73,73,73,54,0
```

Maybe you are not in the habit of writing in either Russian or Greek everyday. I have included these fonts just to show that the VIC-20 is not confined to English.

ENHANCING THE SCREEN DISPLAY

Graphics can make text much more appealing to look at. Certain sections can be highlighted by use of the methods just described; for example, double-sized letters can be created by generating a portion of the letter in each 8×8 matrix and then assembling the matrices like a jigsaw.

However, highlighting can be achieved in a much easier fashion. Try the following. Do not type the parentheses . . . just follow the instructions inside them. (First press C= and SHIFT)

```
10  ?"(C= and F × 13)"
20  ?:?"(C= and F once, two spaces, then SHIFT HELLO, three spaces, C=
    and F once and then ")
30  ?:?" (C= and F × 13)
```

Clear the screen and then type RUN. The word HELLO will appear inside a frame of small black squares. Try the program with other graphic symbols.

For Valentines' Day, the following might be effective!

```
10  ?"PRESS ANY KEY TO FIND THE SECRET MESSAGE"
20  GETA$: IFA$="" THEN 20
30  ?" ([graphic S] the heart! 22 times)"
40  ?:?" (3 spaces) I LOVE YOU---------";
41  FORI=1TO2000: NEXT
45  ?" (22 hearts!)"
50  ?:?:?"YOUR FRIEND-----"
55  FORI=1TO 5000:NEXT
58  FOR I=1TO 10
60  ?:?:?CHR$(147)" ([cursor down]×4, [cursor right]×4)V ([cursor
    down],[cursor right]) I ([cursor down], [cursor right]) C"
65  FORX=1TO500: NEXT X
66  FORY=1TO20:?CHR$(147):NEXTY:NEXTI
```

Your own imagination will allow you to decorate the outline in totally stunning fashion, but might I advise that for a quiet life you substitute your own name for VIC's?

Don't forget that you can place things exactly where you want on the screen by poking the address. Symbols, reverse characters, and special characters you design yourself can all be combined on the screen with great effect.

CREATING BAR CHARTS

Advertisements in magazines often show a brightly colored screen with thick bars, running either vertically or horizontally. Such graphic displays are not difficult to produce and seem to impress office managers out of all proportion to either their actual worth or their difficulty of production.

Let us take the following example:

```
40  Z$=CHR$(166)
50  INPUTX
100  FORI=1TOX:?Z$: NEXTI
```

Clear the screen and then type **RUN**. Enter a value less than 20 after the question mark. The checkerboard symbol (CHR$(166)) runs down the left-hand side of the screen.

Now change line 100 by adding a semicolon (;) after Z$, and before the colon. Now the CHR$(166) runs horizonally.

Next change the semicolon into a comma. This time you have two distinct bars. Change the comma back to a semicolon and then type in a number greater than 20, let us say 35. Now you can see that the whole of one line is filled and part of the next. Try other values for X. 500 will fill the screen.

Now add the following:

```
30  A$=CHR$(113)
```

and change line 100 as follows:

```
100  ?CHR$(18):FORI=1TOX:?Z$;:?A$;:NEXTI
```

Don't worry about CHR$(18) for the moment, just run the program. Now you have a pattern of alternate symbols.

If you change the value of CHR$(18) to, say 144, you can turn the whole screen black!

Change the program to read:

```
100  ?CHR$(18):FORI=1TOX:?A$;:NEXT I
110  ?CHR$(144):FORI=1TOX:?Z$;:NEXTI
```

You could add a couple of lines to produce another symbol in another color, or bring the last printout into normal color balance (CHR$ (31)).

So far you don't have much in the way of a bar chart. That is why you have such strange line numbers: you will add some lines to allow greater freedom of expression.

```
20  Y$=CHR$(115)
50  INPUT"HOW MANY LEMONS";X
60  INPUT "HOW MANY ORANGES";Y
70  INPUT "HOW MANY NUTS";Z
100  ?CHR$(144):FORI=1TOX:?A$;:NEXTI
110  ?CHR$(31):FORI=1TOY:?Z$;:NEXTI
120  ?CHR$(156):FORI=1TOZ:?Y$;:NEXTI
```

Note that we already have lines 30 and 40 which assign CHR$'s to variables.

The following variation on the program could exist:

```
20  X$=CHR$(18)+CHR$(160)
30  Y$=CHR$(18)+CHR$(160)
40  Z$=CHR$(18)+CHR$(160)
```

Lines 50 to 70 remain as before, and then line 80 is inserted:

```
80  ?CHR$(144)
```

In lines 100 to 120, change the CHR$ to 18 and add these lines:

```
105  ?CHR$(28)
115  ?CHR$(31)
```

Now run the program.

As long as you can remember that the lines represent lemons, oranges, and nuts, in that order, things are fine. If you can't, however, things becomes a little rough. You need some means of printing at the end of each bar exactly what it represents.

```
4  DIMA$(20)
5  A$(1)="LEMONS"
6  A$(2)="ORANGES"
7  A$(3)="NUTS"
```

Now watch carefully: on line 80 insert :?A$(1); after ?X$; and before the NEXT I. Make sure you have all the colons and semicolons right. Then insert :?A$(2); in the same spot on line 110; and :?A$(3); in the same spot on line 120.

Now when you run the program, inserting whatever values you like, you are left in no doubt about what is what!

The alternative is to insert three more lines

```
102  ?A$(1)
112  ?A$(2)
122  A$(3)
```

Instead of placing the dimensioned arrays within lines 100-120. Now the name appears only at the end of the line.

Quite clearly, what you put in the program and what you make each bar represent is entirely up to you. The bars could be monthly temperatures, widget, twirdlegroinge, and grundget sales, the ratings you give to pinups and movie stars, or whatever takes your fancy. The program will be approximately the same to produce any kind of horizontal bar chart display.

To produce one of those fancy vertical ones, you must go back just a little to your first efforts. You'll change it just a trifle to make use of the CHR$ + CHR$ trick. This process is called *concatenation.*

```
40  Z$=CHR$(18)+CHR$(160)
50  INPUT X
55  ?CHR$(144)
60  FORI=1TOX:?Z$:NEXTI
```

These lines produce a black bar down the left-hand side. Now add a line:

```
10  X$=CHR$(18)+CHR$(166)
```

Then in line 60, add a semicolon after ?Z$ and ?X$ before NEXT I. Don't forget to leave the colon before ?X$!

```
60  FORI=1TOX:?Z$;:?X$:NEXTI
```

Things are looking better because you have two different patterns right alongside each other. It is the semicolon that does the trick, of course. A comma will separate the columns by half a screen width. A colon will pile the columns on top of one another.

It is at this point however, that you reach somewhat of an impasse, for no matter how hard you try, you will not be able to make each of the columns a different length. There can be as many columns as you can fit on the screen, but they will all be exactly the same length. You could explore the possibilities of nested for-next loops, but let me warn you that it would be tedious and frustrating. It would also get you nowhere.

I am going to digress a few moments and show you how it is done on other computers. This will be useful to you because it is very likely that you will want to translate a program written for another computer to operate on the VIC-20.

On many computers (or rather, in the variety of BASIC they use) there is a plot statement. On the Apple, for example, you can designate points on the screen by plotting the coordinates of a point with two numbers, the first indicating the column and the second the row. PLOT 20,24 would place a point in the center of the screen. On the ZX81, you can use X and Y; X for the row and Y for the column. Plot X,13; Plot Y,11 produces a spot in the center of the screen. If you wished to produce a bar in the center of the screen from the top down 13 rows, you would write a for-next loop that would continue to print squares one under the other until the limit is reached. The loop would read FOR I= 1 TO 13.

Because most computers print from the top line downwards (that is, when the screen is full, the top line is pushed up to make more room, so that even though new material is appearing at the bottom it is really appearing at the top of the remaining space!), the whole set of instructions is very simple:

```
10  FORI=1 TO 13
20  PLOT X+I
```

Then you could plot the column you wish to use:

```
30  PLOT Y,11
40  NEXT I
```

The line or bar is drawn in the customary rapid fashion.

What if you want to go up? You start, of course, at the bottom, line 25 and subtract I for each stage of the loop.

```
20  PLOT X, 25–I
```

Taking things a stage further, you can draw diagonal lines by incrementing or decrementing not only X, but also Y.

```
10  FOR I=1 to 13               10  FOR I=1 TO 13
20  PLOT X,I                    20  PLOTX,25–I
```

```
30  PLOT Y,I                          30  PLOT Y, I
40  NEXT I                            40  NEXT I
```

and so forth. The whole thing works just like a graph.

Now to return to the VIC-20. You will recall that each spot on the VIC screen has its own number. 7680 is the top left corner and 7910 is the center.

You will also no doubt recall the Bomb program in which you made the flying object go diagonally across the screen by incrementing 7680 by 23. Incrementing by 22 would bring the object straight down the left-hand side. The problem here is to bring more than one symbol down the screen and for differing distances. In addition you want each symbol to occur alongside its neighbor. That is where the problem lies. You must find some way to emulate the plot statement found on other machines.

Consider the following program, which does the job with the lemons, oranges and nuts, starting from the top.

```
10  ?"(shift [CLR/HOME])"
12  POKE 36879,8
15  INPUT"HOW MANY LEMONS";X
20  P=7724
30  FOR I=1 TO X: POKE P,1: P=P+22: NEXT I
40  INPUT" HOW MANY ORANGES";Y
43  P=7725
50  FORJ=1 TO Y: POKE P,2:P= P+22: NEXT J
60  INPUT "HOW MANY NUTS";Z
63  P=7726
70  FORQ=1 TO Z: POKE P,3: P=P+22: NEXT Q
550  ?"([cursor down])" which produces an inverse Q times 20
```

The screen goes dark and for each item a letter of the alphabet appears.

All well and good so far. What you really need now is to make the columns come up! Not a difficult job at all really. To leave a couple of blank lines at the bottom, use the address 8120 and enter that on line 20 as the value of P. You can change lines 43 and 63 to read 8121 and 8122 respectively. Next we change the plus sign in lines 30, 50 and 70 to minus signs so that P=P−22.

Now run the program. You have the columns running in the right direction, and you might be satisfied to have the items labeled with letter names. If you wish to produce those pretty colored bars, however, you must add one or two things and change something. Add lines:

```
21  C=38840
44  C=38841
64  C=38842
```

Now, using your cursor and delete keys very carefully, add the following to line 30: POKE C,2:C=C−22 and change POKE P,1 to POKE P,224. Do the same for lines 50 and 70 but use POKE C, 7 for line 50 and POKE C, 5 for line 70.

38840, 38841, and 38842 are the color codes which match the screen codes. The number 224 for POKE P is the inverse of space. Now the program runs fairly sweetly.

It is important to ensure that there is sufficient contrast between adjacent colors or else that the bar chart will not be as clear as you would like. It is up to you what items you ask for, of course, and how many. The program will become a little long-winded if you have a lot of items, and it will need to be reorganized to make it convenient to use the whole screen. That will be tackled in the next chapter.

CAPSULE REVIEW

Each character has an ASCII code value. The character can be directly entered from the keyboard or can be called by its code number. The character is accessed by the CHR$ function in the form: CHR$(X). Thus CHR$(82) is the letter R, and CHR$(36) is the $ sign. The instruction ?CHR$(82) will cause the letter R to appear on the screen.

In addition to letters, numbers, operators, and modifiers, any color, function, or command can be produced by calling the CHR$ appropriate to the requirements. Thus, ?CHR$(29) will cause the cursor to move to the right one space.

The code value of X can be divined by commanding the computer to ?ASC(x) where x is the actual character. Thus ?ASC("R") will produce the number 82.

CHR$'s can be concatenated in the form CHR$(X)+CHR$(X). Functions can be combined with characters in this form, thus ?CHR$(82)+CHR$(14) will produce a lowercase r. If this command is issued in the direct mode, everything else on the screen will be governed and affected by the CHR$(14) code.

Screen editing commands can be incorporated in a program. The form is PRINT (or ?) followed by quotation marks, then the action (cursor down, up, left, or right, clear screen, and so forth). The result on the screen will be in the form of text governed by those functions included after the quotation marks. These functions performed appear as inverse characters in the program listing. Thus "cursor down" appears as an inverse letter Q. Thus material appearing on the screen can be presented in the most favorable fashion according to the programmer's taste.

In addition to the character sets(letters, numbers, symbols) offered by the VIC-20, you are free to design others according to need and taste.

Characters are made up of dots arranged in an 8 by 8 matrix. Each dot is represented by a binary digit in the form 0 or 1. Each line of the 8 by 8 matrix is therefore translated into binary code using 1 for a set dot and 0 for an unset dot. That line is then translated or transformed into decimal numbers. Eight decimal numbers are needed to describe the rows of dots of a normal size character.

In order to produce alternate, self-programmed images, space must be reserved in memory to accommodate them. This is done by poking specific addresses with the appropriate values. The area of RAM so reserved will remain undisturbed by the rest of the program for the duration of its use.

The decimal numbers are accessed and used by the program via read data statements. The decimal values are contained in the data lines, which can be accessed repeatedly by for-next loops. Prolonged use of a different font may require some form of reminder placed on the key tops.

Screen display can be enhanced either by the use of user-generated graphics or by the use of graphics drawn from the sets provided by the VIC-20.

Horizontal bar charts can be generated with considerable ease. Vertical bar charts require more complex programming to be effective. They are done by poking values into screen addresses and, for color, into color addresses.

Chapter 6

The book has already made some reference to subroutines and a passing hint about the GOSUB statement. Our bar chart program is a superb candidate for subroutines as there is a good deal of repetition of one particular feature in a long version of the program. If you were to repeat the contents of line 30 for each and every time you wished to add an item, not only would the programming become tedious, but it would also fail to make use of the main advantage of any computer, convenience! Besides, you would use up a great deal of precious RAM leaving very little for other purposes.

USING SUBROUTINES

The problem in putting the repeated material in a subroutine is in incrementing values. One thing the computer does very well is to keep count! The procedure is very simple. First a variable, let us say XX, is assigned the value 0. At some convenient point in our subroutine, you can increment the counter by 1: XX=XX+1. The counter could serve to limit the number of items you can display information for on the screen. As there are 23 columns on the VIC-20 and as it would look neater not to use the entire screen width, you could limit the value for XX to a total of approximately 15. This will leave room for some text telling what the bar chart is supposed to represent or some figures indicating months, years, temperature, total dollar sales, or what-have-you.

The counter can do some other work. Each time you wish to move the colored bar over one place, you have to enter a new value for the screen and color codes. As you can see from the previous program, the incrementation is by 1 each time. After you have incremented XX by 1, you can put in a line that says P=P+XX and C=C+XX. This will do the incrementing for you!

CREATING A FLOWCHART

In developing a program, it is a good plan to produce a flowchart. It need not be tremendously detailed, but it should be sufficient to allow you to compartmentalize the various components of the program. Clearly you need two types of sections for the bar chart program: one section that asks for input and another section that processes the information and presents it in readily understandable fashion.

A flowchart commonly consists of a lot of fancy boxes with lines drawn between them in various ways. This is fine, even essential, for a long and involved program, but for our current purposes, a simple flowchart will do. It will consist of just two parts:

```
INPUT MODULE    ----    PROCESSING AND PRINTING MODULE
```

As it stands the flowchart moves in one direction only and does not indicate that there are repeated returns to the input module. Nonetheless each module, can be worked upon separately without getting in the way of the other one.

It seems reasonable to begin with the input module, which begins in much the same fashion as our previous program did.

```
5  PRINT "([CLR HOME])"
10 P=8120
15 C=38840
20 INPUT" HOW MANY ITEMS"; A
40 INPUT "ITEM 1: NAME"; A$
45 INPUT" QUANTITY";N
50 GOSUB 5000
```

This is quite a good start. You have cleared the screen and put the cursor in its place. You have set values for the spot you wish the display to start at. You have two sets of input: one that delineates the total number of items and one that allows you to name the item type and its quantity. You also have the important GOSUB statement..

Lines 40, 45, and 50 need to be repeated and, if you will bear with me for the moment, you'll see how to do it the long way. Bring the cursor back up to line 40 and just type 60 over the top of the line number. Move the cursor along and change the 1 to 2. Press the return key. Now type the number 65 over the top of the 45 and press the return key. Then type 70 over the top of 60. List the program, and you will see that you have the three extra lines with the minimum of typing. Just to give the program a little more meat, repeat what you have just done to produce lines 80 to 110 and items 3 and 4.

Line 5000 is the point at which the subroutine begins, and you must make sure that it is isolated from the rest of the program except when called by the GOSUB line. Enter this line: 4999 END

Now you can type in the subroutine:

```
5000  XX=XX+1
5100  FORI=1TO N: POKEP,224: POKEC,2+XX:P=P-22:C=C-22:NEXT I
5120  QQ=QQ+1:DD=DD+1
5130  P=8120+QQ:C=38840+DD
5140  IFXX>=ATHEN 5600
5200  RETURN
5500  PRINT"(Cursor down 20 times)"
5600  END
```

The variable XX is used to increment the color used on the screen. Actually this only allows you to use colors 3 to 8. In order to run through the colors again, it will be necessary to reset XX. Note that you do not allow XX to produce color number 2. It is extraordinarily difficult to see a white bar on a white background!

For your purposes the program runs fairly sweetly. The variables QQ and DD allow you to push the screen and color addresses over one column to the right for each item. You could use just QQ which would be much neater and tighter programming, but the use of two variable names might make things easier to follow for the moment. You could also make each of them increase by two rather than one, producing neat gaps between the colored columns.

Line 5140 makes sure that you do not attempt to enter more items than you said you would. In this regard, you must consider how much information can be absorbed by the viewer. If there is too much information presented at one time, the user will suffer from information overload. Therefore, confine the total number of bars to say 13. This would give you the possibility of displaying, say, monthly figures with an average for the year.

It will also clear up another little problem that could arise when the input question marks clutter up the bars. To avoid this I have set the screen and color codes at 8128 and 38848 for the subroutine in the following, much tighter version.

You will have realized that not only can you repeat the display portion of the program but you can also repeat the input module, thereby saving a lot of time! Here is the shortened version of our program:

```
5  PRINT"([CLR HOME])"
10  P=8128
15  C=38848
20  PRINT "HOW MANY ITEMS DO YOU WISH TO DISPLAY?"
30  INPUT A
40  PRINT "([CLR HOME])"
50  PRINT "ITEM NAME        QUANTITY" (Space according to your
                                       needs)
60  FOR W=1TOA
70  INPUT A$
80  INPUT N
5000  XX=XX+1
5100  FORI=1TON: POKEP,224: POKEC,2+XX:P=P-22:C=C-22:NEXT I
5120  P=8128+XX:C=38848+XX
5200  NEXT W
5600  END
```

This looks fine, and yet there is a small bug! If you attempt to deal with more than the screen can handle the chilling message REDO FROM START pops on the screen. You see, as soon as the question marks reach the same line as the top of the highest bar, the VIC-20 thinks it has nowhere else to go.

You will have noticed, of course, that the GOSUB has disappeared from our program and has been replaced by a for-next loop.

You may feel justified in being very annoyed at being led up a blind alley. This often happens in programming, and the would-be programmer must be prepared to try a number of different methods before coming up with one that is totally satisfactory.

DEALING WITH PROGRAMMING PROBLEMS

There are times when programming can be utterly frustrating. The program just will not run, and any efforts to make it do so are fraught with disaster and disappointment. One time I was working on a program and suddenly the computer seemed to go mad. I had been adding improvements to a short section of a routine, constantly listing and running the section. Suddenly I was no longer able to run the program, all I could get was a listing! I listed small sections of the routine, examining each part very carefully, and could see no problem. My program began at line 10, and so I would command the computer to list from that line number. Still no cause was obvious! In exasperation I listed once more, meanwhile keeping my finger on the CTRL key. Then I saw it! There at the beginning of the program was a line number 1! It read: 1 LIST! Somehow in my haste I had touched key 1 before instructing the computer to list. It had thus been entered as part of my program. Each time I ran the program the computer would follow the instruction to the letter!

This sort of thing happens when you are tired, and the best thing to do is stop the session, first saving the work you have done, and come back to it when you are fresh.

In the present instance, you have accomplished something: you have learned how to manage a bar chart program that can accommodate a few items, and you are on the way to producing a type of program that can cost a lot of money if bought commercially, the data base program!

It is a little annoying, however, that with your present resources, you cannot clean up the act so that you can use the whole screen. Obviously there must be some way, computers can do anything!

At the beginning of this book, I mentioned that the keyboard and TV were devices with specific functions. Clearly, the computer must know which is which and be able to distinguish between information presented to it by the keyboard and that which it already has inside its head. Is there some way we can access this ability to distinguish between devices?

UTILIZING DEVICES

Yes, there is! You will probably already be aware of the fact that various devices can be attached to the computer. Two of them you already have: the TV and the Datasette. In addition you can attach a printer and a disk drive, the latter being a recording device that operates faster than the Datasette.

It is not sufficient to attach these devices. One must also tell the computer which device is being "talked to". For this, you need to know the device number. The printer and disk drive will be dealt with later, but what you are going to do now will serve well for that time when it comes. Each device that can be attached has a number that is recognized by the VIC-20. Before you can use this number, however, you must open a channel of communication. It is rather like lifting the telephone before you dial or switching on the radio before tuning it to the station you wish to hear. The statement to be used is simple: OPEN followed by a file number and a device number.

In this case, you wish to read the keyboard, storing the information that the user enters and, more importantly, using it without disturbing what is about to appear on the screen. The keyboard device number is 0. The statement will thus be in the form OPEN 1,0.

The format of the input will be a little different. Perhaps you might have already tried to use a shorthand for INPUT by typing I then SHIFT N. If you have, you will know that it will not work. There is no shorthand for INPUT, however, I SHIFT N does give you INPUT# !

This is the very thing you need! Type INPUT#1,N your numbers will appear on the screen, but there will be no ? prompt, and the numbers will be running across the top of the screen well out of the way. Look at a cleaned up version of the program with INPUT#.

```
5  "([CLR HOME])"
10  P=8122:C=38842      (You can move your bars to the left now!)
20  ?"HOW MANY ITEMS DO YOU WISH TO DISPLAY?"
30  INPUT A      (You have not yet opened a file so you use INPUT)
35  ?"([CLR HOME])"
40  OPEN1,0      (Your file is open and the keyboard, 0, is being called)
50  XX=0
60  FORW=1TOA      (The loop counts up to the total number of items)
70  INPUT#1,N      (Here is our new instruction)
100  XX=XX+1      (This is our counter variable)
110  FORI=1TON:POKEP,224:POKEC,2+XX:P=P-22_C=C-22:NEXTI
120  P=8122+XX:C=38842+XX
130  NEXTW
140  END
```

Now the program runs sweetly. You can give N any value up to 19 without running into the top of the screen or the numbers you are entering. The variable A can be given any value up to 11 before funny things begin to happen. Try entering 13 items and watch how the screen behaves!

However well the program seems to work, there is still a small feature that is somewhat annoying. Notice that the seventh bar seems to be missing. It isn't! You cannot see it for the simple reason that it is white on a white background!

The variable XX is not only used to increment the screen location on line 120, it is also used to increment the color code. The first color to appear is cyan. The colors are incrementd by one until they reach yellow and then recycle back to black (which isn't actually a color). The next color is white.

Now, either you remain happy with the fact that the seventh item has no visible bar or find some way to avoid the problem. One way to avoid white is very simple: 100 XX=XX+2. Now you can enter more than seven items without one of them being subject to a white-out! But something else happens! The bars are now spaced! If you wish, you could start the columns at addresses 8120 and 38840, thus allowing single digit numbers to appear above the columns they represent. Double digit numbers will get things out of kilter, however. The number of columns or bars that can appear on the screen is also reduced.

Just for practice, change the bar chart program to produce horizontal bars using INPUT#1,0. Finally, add line 135 to close the channel: 135 CLOSE 1.

The information you have learned thus far can be used in creating a data base program. Before you tackle such a large job however, there are a number of other tricks you must teach your VIC. One of them you have already encountered but not discussed at any length: DIMA$(X), the dimensioned array. While you are dealing with a facet of DIM, you might as well deal with so-called artificial intelligence, just so that you can see how dumb computers really are.

USING ARRAYS

The problem you are going to give to the computer is reminiscent of the old contention that if you give a 100 monkeys a typewriter each and enough time, they will eventually come up with the entire works of Shakespeare!

```
5  ?"([CLR HOME])"
10  DIMA$(20)
30  GOSUB 501
35  FORD = 1TO 1000:NEXTD
40  ?A$(1)+A$(3)+A$(10)+A$(1)+A$(4)
45  C=0
50  FORI=1 TO5
70  J=INT(RND(1)*20)+1
80  Z$=A$(J)
90  ?Z$;
100  NEXT I
102  ?
105  C=C+1
110  FORD=1 TO 2000 : NEXTD
115  IFC>=5THEN END
120  GOTO50
501  A$(1)="THE"
502  A$(2)="AND"
503  A$(3)="CAT"
```

```
504  A$(4)="DOG"
505  A$(5)="RAIN"
506  A$(6)="ROTTEN"
507  A$(7)="RED"
508  A$(8)="GREEN"
509  A$(9)="BROWN"
510  A$(10)="BITES"
511  A$(11)="GROWLS"
512  A$(12)="MEOWS'
513  A$(13)="AT"
514  A$(14)="RUNS"
515  A$(15)="BARKS"
516  A$(16)="AROUND"
517  A$(17)="SCRATCHES"
518  A$(18)="EATS"
519  A$(19)="FROM"
520  A$(20)="FALLS"
600  RETURN
```

The program begins by declaring a dimension for A$, allowing up to 20 versions of A$ to exist. The Gosub reads the arrays from line 501 onwards and then returns to print out a concatenated group which we have determined beforehand. This shows how it could be done.

Next we have something we have not yet dealt with: RND. Line 70 asks the VIC-20 to select at random from the entire list of A$(X). It does this by first selecting a number between 1 and 20. It then identifies this number by the variable J. J is then used as the subscript to A$(X) and the result is called Z$. Z$ is then printed. The VIC is instructed to do this in groups of five by means of a for-next loop, and then to leave a little gap and do it all over again. The total number of times this is done is limited by counting up to five. As soon as five is reached the program stops.

When you run the program you will see that the computer prints mostly garbage. The computer cannot of itself recognize the difference between parts of speech. You need to add a few things to allow it to do this. (For your typing convenience, you will find the complete listing in Appendix N, program 41.)

```
15  DIMB$(20)
20  DIMC$(20)
25  DIMD$(20)
```

Delete line 50 and line 100, lines 115 and 120, and lines 80 and 90. Then:

```
125  IF C>=25THENEND
130  Z$=A$(J):PRINT Z$;
```

```
410 PRINTA$(9);" CHEAP "
420 PRINT"THE "  A$(4);"'S ";A$(10);
430 PRINT" BRIGHTENED, HIS "A$(11);
440 PRINT" WIDENED AND HE ";A$(13);
450 FORD=1TO9000:NEXTD
460 PRINT" I AM TRULY AMAZED, SAID THE ";A$(4);
470 PRINT" I MUST GO AND GET SOME ";
480 PRINTA$(14);" THEY WILL HELP MY "
490 PRINTA$(15);" "A$(16);
500 PRINT" I WILL COME WITH YOU AND SEE IF "
510 PRINT"THEY FIT, SAID THE "A$(1);
520 FOR D=1TO9000:NEXTD
530 PRINT"♥"
540 PRINT"AS THEY ";A$(17);
550 PRINT" INTO THE "A$(18);
560 FORD=1TO2000:NEXTD
570 PRINT:PRINT:PRINT"THEY "
580 PRINT:PRINT:PRINT A$(19);"!"
590 FORD=1TO2000:NEXTD
600 PRINT:PRINT:PRINT:PRINT "THE END!"
```

This program demonstrates the value of dimensioned arrays and the ability to print them wherever you wish in the program. In a sense it is an extension of the very first program you wrote, which asked for names and printed them out.

You can write as many stories as you like to fit the required words, perhaps even using exactly the same words that the user types fitted into a variety of stories. You could, for example, have ten stories all of which call for the same parts of speech in the same order. The input section would be required only once in your program. The stories could each begin at 500, 600, 700, and so on, and the VIC-20 would then randomly select the story to which the words would be fitted. For a party there would be sufficient variety to keep people laughing and groaning for quite some time.

Taking things just a stage further, it would be possible to build a whole series of string arrays, each labeled with a variable according to whether it is a verb, a noun, or an adjective, and so on. The computer would then select from each of these lists at random and insert the selected word into the stories itself. Such a program could fascinate folk who are not familiar with computers for hours.

The program would run in reverse to the one you have just written: you would input the story but indicate by code what part of speech you wish to use. The computer then prints your story with the randomly selected parts of speech. This looks very much like artificial intelligence at first sight. It is nothing of the kind, of course. Artificial intelligence requires much more sophisticated programming techniques than those which you have at your current disposal. Such programs also take lots of time to write!

For the moment, you can introduce a very simple program to get some practice in the use of DIM and RND. At the same time you can practice the process of *top-down* program development.

Many texts on programming urge the beginner to draw a map or flowchart of the program. For long and complex programs, this is very good advice. However, it is also a good plan to practice the art of program development at the computer. Any program you write will consist of component parts that perform specific functions. Each part can be singled out for development at the keyboard.

The first thing to do is to define the problem. The program to be created will allow guests at a party to enter their names. The names will be sorted and ultimately retrieved from storage for whatever purpose you wish. It could be that you wish to offer a door prize at some function or other, or maybe you would just like to have the first electronic visitors-book on the block. Maybe you wish to keep track of attendance at society meetings or keep account of donations or dues.

Obviously the first part of your program will allow guests to enter their names. Inasmuch as there is no need to view the file before everyone has registered, there is no need to allow for this. But you can leave space for such a feature in case you wish to add it later.

The second part of the program will allow you to view the file by printing it on the screen very slowly. You could include a menu to allow you to select the printout when you are ready.

The first part of your program is already done! You have the very thing in the names program on page 19. You should increase the value of the line numbers however, just to give yourself some room to work.

```
10  ?"[CLR HOME]"
20  ?" TO ENTER NAMES PRESS A"
30  ?TO DISPLAY NAMES PRESS B"
40  INPUTA$
50  IFA$="B"THEN250
100  INPUT"HOW MANY NAMES";N
105  ?"[CLR HOME]"
110  DIMA$(N)
120  FORI=1TON:INPUTA$(I)
130  ?"[CLR HOME]"
140  NEXTI
150  ?"([cursor down] times 7, [cursor right] times 4)NOW SORTING"
160  FORI=1TON-1
170  IFA$(I+1)>=A$(I)THEN200
180  B$=B$(I+1):A$(I+1):A$(I)
190  A$(I)=B$:GOTO160
```

```
200  NEXT I
205  GOTO 10
250  ?"[CLR HOME]"
255  ?"WHAT IS THE PASSWORD?"
260  INPUTX$
270  IFX$<>"JIM"THEN 400
300  ?"[CLR HOME]"
305  FORI=0TON+1:?A$(I):?
310  FORD=1TO2000:NEXTD:NEXTI
400  ?"WRONG PASSWORD":GOTO255
```

This program is essentially the same as the previous sorting program. However, you have the choice as to whether you wish to enter or list the names. In case there is some need for security a very simple password device is included. In this case the name JIM allows you to see the list. The password can be anything you like, of course. You could use a telephone number or birthday listed as a set of numbers, as in 741938, personal vital statistics, such as 362236, or even a two part password; one part being consisting of letters and the other of numbers.

There is, of course, the danger that someone who is familiar with computers will know how to list the program and thus find out the password. You could try a few simple tricks such as shifting the letters J I M as you enter them on line 270. This means that if you just type JIM, you will get the message on line 400. You could also put spaces between the letters. An experienced individual will crack whatever code you care to think of, but most folk will not. And by the time most folk do know what to do, you will be a thoroughly experienced programmer yourself and will have mastered methods of keeping programs secure.

For the moment, however, you have a problem yourself! Let's say the party is over, the guests have gone home, and you have saved the information on tape. The next day you load the program and run it. You choose the B mode to print out the names, and nothing happens!

Now you know that the information is stored on tape or should be, but no matter how many times you run the program, the names you went to such trouble to collect do not appear on the screen!

The problem lies with the command run. You could try entering RUN30, but the run command will still clear out all the variables . . . and the names that were entered are just that, variables!

The solution is to remember to use the GOTO command followed by the line number *after* the one in which the array is dimensioned, in this case, line 30. You will have to load the program again and then type GOTO30. Now your names will appear, assuming you have not forgotten the password!

Programs like this are touted as being useful to hold any information that you would normally put on little cards: recipes, record and book titles, and so forth. I must confess that I don't use such a program for any of those things because I know exactly where each book and record is on the shelves, and because I don't cook! I do use such a program to keep track of computer tapes and disks, and to produce indices for books and bibliographies for books and papers. A larger file program is printed in Appendix F.

For your amusement I have included the bare bones of a program called "Talk" in Appendix G. It makes use of the DIM statement with subscripts, GOSUB and GOTO, but is not intended to be taken seriously! I provided only the bare bones beginning so that, if you wish, you can extend it in any way you wish. As it stands, it will fit into the 3K RAM of the VIC-20. Further suggestions as to expansion are included after the program listing.

CAPSULE REVIEW

When material is recycled there is often a need to know how many times the repetition takes place. In addition, certain values are often required to be incremented for various purposes. A value, usually 0, is assigned to a variable. A counter is then placed within the body of repeated material. If there is to be a limit on the total numbers of times the routine is to be used, a limiting statement, usually in the form IF X = N THEN -----, is used.

Screen pokes can be incremented using either the same variable or another one. The same rules apply.

When developing programs care must be taken that no BASIC commands, such as list, appear by accident. The result can be very exasperating.

The input statement requires that a question mark appear on the screen and that all information or data which is entered appear on the screen too. This can result not only in a cluttered screen but also in interference with any graphic output that results from running the program. One solution would be to clear the screen for each entry by means of the CLR HOME statement embedded in the loop.

A better solution is to use the INPUT# statement. This requires that a file be opened by using the open command. Both open and INPUT# belong to the category known as I/O, which stands for input/output and refers to the accessing of the various devices that either form the basic computing equipment (the VIC-20, the TV) or the peripheral or surrounding equipment (the Datasette, disk, or printer).

The keyboard can be read by means of the open command. The format for this command is : OPEN 1,0; where the first numeral states the file to be opened and the second the device being called.

The open command may not appear within a loop unless the close command also appears within that same loop.

Following the opening of a file and device, the statement INPUT# is used instead of the common INPUT. The format is : INPUT# 1, followed by the variables that are appropriate. The first numeral after the INPUT# must be the same as that following the open command, for it refers to the same file.

String arrays may be concatenated or linked and the result displayed on the screen or other device. Concatenation may be the result of a fixed statement: A$(1)+A$(3) and so forth; or may be the result of random selection of a value for X.

Random selection is performed by using the RND statement in the form: X=INT(RND(1)*Y)+1, where X is the number to be generated and Y, for example, can be the total number of string arrays from which you wish to make the selection. The dimensioned array totaled 20, you may assign any value up to and including 20 for Y.

The RND statement causes the computer to select a number between 0 and 1. Quite obviously this number is a decimal fraction. This number is then multiplied by Y, and the INT (integer) statement is used to produce a whole number up to and including Y. (A more detailed explanation would be quite lengthy and, for the purposes of this book, quite unnecessary. It works, and that is the important thing!).

There is a great possibility that the computer will select the same number twice or even three times in a row. This may or may not be inconvenient depending on your program.

The run command erases any variables that have been stored. Thus any data or input material will be lost. To avoid this, use the GOTO command, followed by a line number after that in which variables have been assigned.

Simple security measures against unauthorized access to stored information may be taken. However, any security measures written in BASIC can be circumvented with relative ease by a knowledgeable user.

Program development is best accomplished by preparing a plan beforehand, even to the extent of producing a flowchart.

Chapter 7

Sooner or later the VIC-20 owner is gripped by the uncontrollable urge to expand the equipment. Insofar as it is likely that the initial use of the computer will involve games, the first peripheral that will be purchased will be the joystick. (You could, of course, have bought a lot of other things instead, but I have to make some assumptions!)

THE JOYSTICK

The name of the device is a result of the quirky and naturally pornographic turn of mind of the early fliers. Nowadays, even small aircraft have a curious half steering-wheel set upon a column. You push on the wheel and down you go; pull on it and up you go. Turning it left and right causes corresponding changes of direction. In ancient times (the early part of this century!) there was no wheel but rather a vertical tiller. To turn left or right, you pushed the whole stick left or right. The last aircraft that I flew in with such a device was a J-3 Cub.

Now I fly a VIC-20! You can too but first you must learn something about the operation of the joystick. Like most everything on the VIC-20, you must poke the joystick into action. ''Ah, but wait a moment,'' you might say. ''Isn't the joystick a device?''

Well, yes it is.

"Then why use pokes when devices use open, INPUT#, and so forth?"

This is a very good question. The joystick, while being a device, is not what you would call a stand alone device. When you used the keyboard as a peripheral you had a specific purpose in mind. Normally the keyboard is read automatically. The joystick takes over part of the keyboard for its own purposes, when you tell it to, that is. Thus you can use pokes to tell the computer that part of the keyboard is now shut down leaving the joystick in control. If the keyboard is used while the joystick is in control, some strange things happen.

To verify this, type POKE 37154,127 and then type the numbers in turn. Only the odd numbers appear! You can type any of the letters and all of the operators save for the minus sign. In a sense POKE37154,127 is the equivalent of OPEN Joystick. However, the joystick is only partly operational. Push the stick to the right and then repeat the exercise with the numeral keys.

See what I mean?

The pokes you have performed deal only with the right position of the joystick. At least, pushing the joystick to the right has an effect on even numbers.

If you wish to write a program that has no need of the keyboard, you begin with POKE 37154,127. To restore the keyboard at the end of the program you can use POKE 37154,255.

Now you only need to do this poke if you wish to check that the joystick has been pushed to the right. For this you also need another poke. (The first poke only *allows* you to do it, it doesn't do it for you!) The poke is actually the reverse of a poke! It is a *peek*.

You see, you use a poke when you want to actually place something in a specific spot in memory. What you want here is the means of checking a specific spot to see what is there!

The peek does this, and so you can use PEEK(37152) AND 128 The other directions also use peeks as follows: left: PEEK(37151)AND16, down: PEEK(37151)AND8, and up: PEEK(37151)AND4. There is one more peek and that concerns the fire button: PEEK(37151)AND32.

You will recall that in our sound programs the pokes were assigned to variables to save typing lots of numerals. You can do the same with peeks. Thus you can assign variables to the numbers and then use those variables within the appropriate expression to produce new variables.

Things can get pretty complicated right about now if you are not careful. You could find yourselves with a very long program that does very little. So, if you will pardon me, I will take a short cut and give you what appears to be a riduculous example of double-negative gobbledegook, but it will get you using your joystick tout-suite.

The formula you are going to use saves you from having to think upside down or inside out. Usually when you set a bit, which is short for binary digit, you set it to 1. The inverse happens with the joystick. Instead of the bits being turned on when the stick is pushed (whatever the direction) they are turned off. This is what I mean by "thinking inside out". The formula to use is as follows: X=(NOT PEEK(37151))AND 60–((PEEK(37152)AND128)–0). Don't worry about it! Just keep it in a safe place until you need it! Your value for X will now be easily incorporated into further expressions with the computer doing all the work for you, and surely that is the main reason you have a computer, isn't it?

This is what the directions look like with X:

X AND 16 --Left
X AND 8 --- Down
X AND 4 --- Up
X AND 1 --- Right

There is one more for the fire button:

X AND 32

In the following program, you can make use of the fabulous formula plus two more that check for motion left-right and up-down. The new formulae are:

```
H=SGN(XAND 1)–SGN(X AND 16)
V=SGN(XAND 8)–SGN(X AND 4)
```

The result of all this, in the program that follows places a small dot on the screen (actually it is our friend the period CHR$(46), which you can then move all about the screen as you wish. By pressing the fire button you can change the color of the period. The colors are indicated in the data line at the beginning of the program. To set yourself a challenge, first draw a few patterns on the screen, and then press the fire button until the period is erased. Then see if you can erase all the other periods by passing over them. It isn't easy!

Here is the program:

```
10  ?"[CLR HOME]"
20  DATA5,24,28,30,128,148,150,157
30  DIMC(7):
40  FORJ=0TO7:READC(J) :NEXTJ
50  S=1:?CHR$(C(S)) ;
60  POKE37154,127
70  (NOTPEEK(37151))AND60–((PEEK(37152)AND128)=0)
```

```
    (That's the one!)
80  POKE37154,255
90  IF (XAND32)=0GOTO140      (The fire button)
100  IFX=0GOTO60
110  B=1:S=S+1:IFS>7THENS=0
120  ?CHR$(C(S));
130  GOTO60
140  B=0
150  H=SGN(XAND16)-SGN(XAND1)      (You can see how useful X is!)
160  V=SGN(XAND4)-SIGN(XAND8)
170  ?CHR$(46);CHR$(29);CHR$(17);CHR$(17) ;
180  FORJ=0TOH+-:?CHR$(157) ;:NEXTJ
190  FORJ=0TOV+1;?CHR$(145) ;:NEXTJ
200  GOTO60
```

You can experiment with line 20 to produce new colors, and you are quite free to change CHR$(46) to something else.

One word of caution! Make sure you include all the parentheses in all the right places or else the program will not work.

Let's combine something you have done before with what you have just learned. It is much more fun to go around the screen gobbling up dots with some strange device. For the moment, you will keep your device simple and nondirectional. Here is the program:

```
5/POKE52,28:poke56,28:CLR
8?  "([CLR HOME])"
9  GOSUB 210
10  X=7680:XX=38400:Y=22
15  FORI=1TO80
18  Z=INT(RND(1)*22)+1
19  Q=Z+Y
20  POKEX+Q,46:POKEXX+Q,2
22  Y=Y+5
25  NEXTI
40  POKE37154,127
50  S=(NOTPEEK(37151))AND60-((PEEK(37152)AND128)=0)
60  POKE37152,255
80  H=SGN(SAND16)-SGN(SAND1)
90  V=SGN(SAND4)-SGN(SAND8)
100  ?CHR$(64) ;CHR$(17) ;CHR$(17) ;CHR$(29) ;
110  FORJ=0TOH+1:?CHR$(157) ;:NEXTJ
120  FORJ=0TOV+1:?CHR$(145) ;:NEXTJ
130  GOTO50
199  END
210  FORI=7168TO7175:POKEI,PEEK(I+25600) :NEXTI
220  POKE36869,255
230  FORA1=7168 TO7175: READA:pokeA1,A:NEXT
240  DATA8,8,62,73,62,28,0,0
250  RETURN
```

This relatively simple program merely scatters dots over the screen (the number can be changed by altering the value in line 15:

the smaller the number the fewer dots, merely because the scattering process does not allow X and XX, which are the screen and color codes respectively, to reach the highest values), which are then overwritten by the curious craft generated by the data in line 240.

This game can be quite annoying because the alien gradually fills the screen with itself. This results in a situation where you can get quite lost. In addition, there is no screen limit, which means that you can run right off the edge of the screen at the right or left and appear immediately on the other side. This can be useful, of course. However, if you are trying to gobble up more spots, it can become a problem if you overwrite yourself. There are thus times when you can be moving the joystick and nothing seems to be happening. It is, the alien is merely moving through himself. If you direct the alien downwards you will find that the whole screen will scroll upwards, thereby getting rid of all the dots in very short order—but that's cheating!

If you wish to erase the alien you may poke 32 into the address just left by the image. You can try this problem for yourself. The solution will be found in other programs!

In Appendix H you will find a program called Amazement with which you can have some fun.

Here is a crunched version of the bouncing ball program found on page 64 of your VIC-20 users' manual:

```
10  ?"[CLR HOME]"
20  POKE36879,9:POKE36878,15
25  S=7680
30  X=1:Y=1:DX=1:DY=1
32  FORL=1TO10:POKE7680+INT(RND(1)*506),102
33  NEXT
40  POKES+X+22*Y,81       (This is the ball)
50  FORT=1TO10:NEXT
60  POKES+X+22*Y,32       (This erases the ball!)
70  X=X+DX
80  IFX=0ORX=21THEN DX=-DX:POKE36876,220
    (detects right and left edges)
90  Y=Y+DY
100  IFY=0ORY= 22THENDY=-DY:POKE36786,230
105  IF PEEK(S+X+22*Y)=102THENDX=-DX:DY=-DY:
     POKE36876,180:GOTO70
110  POKE36876,0:GOTO40
```

This is a neat little program that does nothing all by itself. You can sit and watch it for hours. A neater trick would be to change the color combinations every so often, preferably at random. The program shows how to animate, add sound, and detect presence and react accordingly.

Quite by accident (you will eventually get to know how many unusual effects are discovered by accident), I added line 105 in the wrong place. I listed it as line 120. Instead of the ball being bounced back when it reached a gray square, the square was gobbled up. The first few squares vanished fairly quickly, but it was a very long time indeed before the last four went! Coupled with a color change sequence and a routine that throws in more squares or other shapes at random from time to time , you could produce a baby-soother or perhaps even use the program in a doctor's or dentist's waiting room. The sound could be improved to produce a swish as the ball travels, and different notes at random. My own preference would be to turn the sound down for such a program however.

THE GAME PADDLES

The game paddles are used by following procedures, similar to those used when dealing with the joystick, except that there are two of them. This might seem to promise greater problems, more confusion, and greater likelihood of error, but in fact things are not nearly as bad as a first glance would suggest.

Page 249 of the *VIC-20 Programmer's Reference Guide* prints the following program:

```
10  POKE37139,0:DD=37154:PA=37137:PB=37152
20  PX=36872:PY=36873
```

These variables are the addresses of, respectively: the data direction registers for ports A and B; the output registers for paddles X and Y; and the digitized values of paddles X and Y. Part A refers to paddle X. Part B refers to paddle Y.

```
30  GOSUB 9000
32  ?PEEK(PX) ;PEEK(PY) ;X;Y
8990  GOTO 30
9000  POKEDD,127:Y=-((PEEK)(PB)AND128)=0) :POKEDD,255
9010  X=-((PEEK(PA)AND16)=0) : RETURN
```

When you run this program with the game paddles installed, of course, all you will see is four columns of numbers that seem to be static. Turn the control on one of the paddles and you will see one of the columns of three digit numbers change as you do so. Of course, it is possible that you do not have two columns of three digit numbers; it depends on how the paddle control is set when you run the program. You will also see two columns of single digits; Ø . Press the fire button and you will see a 1 appear. The left column of each pair is the X paddle and the right column the Y. Play with it for a while observing the effect of each of the paddles one at a time.

Then add lines as follows:

```
25  S=7680:T=38400
50  POKE(S+255)-(PEEK(PX)) ,81:POKE(T+255)-(PEEK(PX)),2
60  POKE(S+510)-(PEEK(PY)) ,83:POKE(T+510)-(PEEK(PY)),6
```

And erase line 32.

All this program does is to put a ball and a heart in various places on the screen as you operate each of the paddles. The symbols are not erased but merely overwrite themselves. Notice how carefully you must turn the controls in order to produce consecutive symbols. Play with this until you have a thorough understanding of what is taking place. Pressing the run/stop and restore keys will clear the screen. You can; of course, put in a line at the beginning to clear the screen from scratch.

The two symbols have been separated on the screen to make their activities obvious. If you change lines 50 and 60 to read as follows:

```
50  POKE(S+22)+(PEEK(PX)) ,81:POKE(T+22)+
    (PEEK(PX)) ,3
60  POKE(S+22)+(PEEK(PY)) ,83:POKE(T+22)
    +(PEEK(PY)),6
```

the two symbols share the same area. Turning the paddle controls very, very slowly or, watch how the symbols take each other's place and change color. Also notice how the controls seem to be working in reverse. Try to set both paddles to the same value; in other words, place both symbols in the same spot. Then fill the screen (actually only half of it) with one symbol and very, very slowly wipe the other symbol over the top. Observe what happens closely.

Next, remove the return statement from line 9010 and put it in line 9040; then add:

```
9020  IFX=1THENA=A+1:IFY=1THENA=A+1
```

or if you prefer:

```
9020  IFX=1ORY=1 THEN A=A+1
```

Also add:

```
22  A=0
```

and change the final digits in lines 50 and 60 to A.

These lines deal with the fire button on each paddle. All they do in this program is change the color. White makes the symbols invisible and thus serves as a means of deleting the symbols.

The use of two separate statements, each giving a different value for the color (and thus a different variable) for each paddle, can create a clever game in which each player tries to wipe out the other's symbol and at the same time, puts as many of his or her own symbols on the screen as possible. The game would end when the players tire.

There can be as many games using the paddles as your imagination will allow. Symbols can be those drawn from the VIC character set or those that you design yourself.

CAPSULE REVIEW

The joystick and the paddles are, like the keyboard, input devices. In the same way that the keyboard is scanned by the computer to see what has been entered, so the joystick contacts are read to determine the same information.

A portion of the keyboard is taken over by the joystick. Command is passed to the joystick with POKE37154,127 and returned to the keyboard using POKE37154,255.
The direction the stick has been pushed towards is read by the PEEKS:

(PEEK(37152)AND128)	Right
(PEEK(37151)AND16)	Left
(PEEK(37151)AND 8)	Down
(PEEK(37151)AND 4)	Up

The fire button is read by the following PEEK:
(PEEK(37151)AND 32) By assigning the various addresses to variables, the expressions can be shortened to more managable form.

Because the position of the joystick is read in reverse, it is useful to introduce a further expression to rectify the matter; In one form the expression is as follows:
X=(NOTEPEEK(37151))AND 60– ((PEEK(37152)AND128)–0) allowing us to reduce the directional expression to (XAND1) -Right; (XAND16)--Left, and so on.

The paddles operate in similar fashion, but they are somewhat simpler to use. Although six parameters must be set, variable means can be used to deal with the necessary expressions, thereby shortening the programming process considerably.

The addresses are as follows:

37139 is the data direction register (DDR) for paddle X
37154 is the DDR for paddle Y
36872 is the digitized value of paddle X
36873 is the digitized value of paddle Y
37137 is the output register for paddle X
37152 is the output register for paddle Y

You will note the similarity between addresses for both paddles and joystick.

Chapter 8

This chapter will discuss some more attributes of the VIC-20 that exist as result of the nature of the BASIC language. There will come a time, if it has not already come, when you will look through and even study other books on the BASIC language.There will be some instructions that you will not recognize simply because they do not form part of the vocabulary understood by the VIC-20. This should be no cause for concern. There is hardly a computer on the market that will understand every word that is possible in the language called BASIC. The situation is rather like the difference between varieties of English or German. Certain words which are common in England are never used in North America, and vice versa. Some words used in northern Germany are quite unused in Austria and so on.

You will have realized by now that each instruction in BASIC stands for a large number of operations. Sometimes a group of BASIC operations are clustered together and brought into play by yet another BASIC statement. Such instructions are often found as a superset of BASIC. A number of them will be discussed in the chapter which deals with the Super Expander.

STRING OPERATIONS

For the moment let us take a look at some of the string handling abilities. A string, as we already know, is a collection of characters, all of which belong together in some way and are thus presented on the screen in a particular order that is specified by the programmer.

You might write a line such as:

```
10  A$="What you doing?"
```

Whenever you wish those words to appear on the screen, you merely direct the computer to PRINTA$. There is no likelihood that the computer will print "Doing you what?" or "What? doing you" or any other combination—unless you wish it to, that is!

There are three statements which allow us to single out certain portions of a string for special treatment. They are LEFT$,MID$ and RIGHT$. Although the intent of these statements is fairly obvious, the way to use them may not be.

Let's begin with a short program, a game, in fact, that will demonstrate the use of the MID$ function.

```
10  ?"[CLR HOME]"
20  ?"GUESS MY LETTER"
22  A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
25  Y=0
30  X=INT(RND(1)*26)+1
35  X$=MID$(A$,X,1):?
40  ?I'M THINKING OF A LETTER"
50  ?"CAN YOU GUESS IT?"
55  FORD=1TO2000: NEXTD
56  ?"[CLR HOME]"
60  INPUT"ENTER YOUR GUESS";Y$
70  IFY$<X$THEN?"TOO LOW":GOTO55
80  IFY$>X$THEN ?"TOOHIGH":GOTO55
90  IFY$=X$THEN?"THAT'S IT!!"
92  IFY$=X$THEN Y=Y+1
98  IFY=10THEN120
100  GOTO30
120  ?"YOU SCORED TEN POINTS!": GOTO30
```

Lines 30 and 35 do the work in this program. Line 30 is the now familiar random number generator, and line 35 contains the string function MID$. A$ is the entire alphabet; X determines at which character, between 1 and 26, selection begins, and the 1 indicates that the selection ends 1 character later. Change this last digit to a 2 or 3 and see what happens.

The string functions LEFT$ and RIGHT$ operate in the same way; the first dealing with the leftmost characters of a string up to and including the number specified in the expression; and the second dealing with the rightmost characters.

The obvious use of these functions is to extract certain portions of information without disturbing the rest. A data base program can use these functions. Let's begin with a simple program that extracts a certain class of information.

```
10  ?"What initial do you seek?"
20  INPUT X$
30  READ N$, T$:
40  IF LEFT$(N$,1)= X$ THEN ? N$, T$
50  GOTO 30
100  DATA JOHN,555-1212
110  DATA HARRY,555-1213
120  DATA FRED, 555-3121
130  DATA ARTHUR, 555-2121
```

This program picks the initial letter required and then prints out the name and the telephone number.

Add:

```
22  IFX$="NONE"THENPRINT"WHAT NUMBER":INPUTA$
45  IFRIGHT$(T$,4)=A$THENPRINTN$
```

This alteration produces a name for the last four digits entered. The program gives you a choice between name and number. You could easily alter the digit delineating T$ to deal with just one, two or three numbers.

If the exchange codes were different, a line seeking LEFT$(T$,A) could tell you the names of those listed under that exchange. N$ could also allow you to enter a full name; then, with appropriate LEFT$, MID$ and RIGHT$ statements you could select whichever portion of the name you wish.

Expanding even further, you could store name, address, telephone number, business, profession, vital statistics, and a rating from 1 to 10! Each of these items would be regarded as a *field* that can be accessed and used to locate all the other information.

The data base program mentioned earlier is nearly within our grasp. By combining the alphabetizing program with a search facility, you have the means to produce whatever information you wish, just when you want it.

Note the display format: the name is neatly separated from the number. It is the comma between the N$ and the T$ in line 40 which does this. A couple of print statements inserted before N$ will bring the display farther down the screen and make things a little more legible. A CLR HOME statement before the print statement will get rid of the other clutter.

Let us add a few more lines and see what happens:

```
140  DATA JIM,555-1214
150  DATA JERRY, 555-1312
```

Then change the number in line 45 from a 4 to a 2. Now follow the instructions very carefully:- At the first prompt type in J. The result is:-

```
JOHN     555-1212
JIM      555-1214
JERRY    555-1312
```

The names are not in alphabetical order for the simple reason that the read statement goes through the names as they stand. Now run the program again, this time typing NONE after the first prompt and then typing the number 12 after the second. The result this time is:-

```
555-1212    JOHN
555-1312    JERRY
```

The computer has accepted the last two digits in the two phone numbers.

To accept the first two digits you would need to search for MID$ in the form MID$(T$,4,2). Now you have the makings of a telephone directory!

A curious function, or pair of functions, involve time. Type the following program:

```
10  TI$="000000"
20  FORI=1TO10000:NEXT
30  PRINTTI$
40  PRINTTI/60"SECONDS FROM RUN"
```

When you run this you will get the eerie feeling that the VIC-20 has gone to Florida; then suddenly up will pop the result, which should be 000010 for TI$ and a slightly higher number for the SECONDS FROM RUN. Whence the extra time? The computer, fast as it may seem, does take just a little time to perform all those fancy tricks. For example, let us see how long it takes to count from 1 to 1000. Change line 20 as follows:

```
20  FORI=1TO1000:PRINTI:NEXT
```

As it runs, this program will display each of the numbers from 1 to 1000, much, very much, faster than you can count. At the end, you will see the total number of seconds. On my machine the result was 28, with total elapsed time being 28.1333334 seconds from run. I ran the program five times with the result being the same each time. When the figure was changed to 2000, it took the machine 55.7666667 seconds from run, which is slightly less than twice as long.

Clear the screen and type ?TI$. I cannot say what figure you will get on your screen, but whatever it is divide it by 60 in the direct mode (i.e. ?N/60). The answer will be in seconds and it will represent the time elapsed since you ran the program you have just been playing with.

Now type ?TI. You will get quite a large number. Now type ?(TI/100)/60. This will give the number of minutes elapsed since you started playing with the TI$ function.

The uses to which the time functions can be put are again limited only by your imagination. Games can be played against the clock and computer assisted instruction programs can be worked against the clock. In the latter it would be possible to count the time taken to answer questions and give credit accordingly. In a sense you have already made quite extensive use of the clock ticking away inside the VIC-20 when you used those FORD-NEXTD timing loops.

Some of the functions available on your VIC-20 may not seem to have much purpose. Many of them are indeed rather specialized and would be used only under certain circumstances. A couple of examples of fairly specialized functions are STR$ and VAL.

STR$(X) is an odd one for it allows for the printing of a numeral as if it were an alphabetical string. An example is in order: As you will have already found out, the simple variable X used in an input statement requires that a numeral be entered. The variable X$ would allow one or more characters of any type to be entered. However, any number that was entered into X$ could not be used in mathematical functions.

What if you had a dimensioned array and you wished to deal with letters and numbers that will be operated on mathematically?

When you want to deal with numbers in a nonmathematical way STR$ will help out.

```
10  A$=STR$(235.47)
20  ? A$
```

So far so good. Now try this:

```
10  A= 235.47
20  A$=STR$(A)
30  ?A:?:?:?A$
```

VAL is the reverse of STR$ inasmuch as it returns the string material as a numerical value. For example:

```
10  X=VAL("235.47")
20  ?X
```

In long, involved programs which perform immense calculations, it is often more convenient to cause the computer to print out the numerical result alone, without all the verbal trappings. You might, for example, enter a large number of figures that relate to specific items. Instead of printing out the item's names, the VIC-20 (or any other computer with the VAL function for that matter—it is fairly standard) will just print out the result of the calculation that refers to those items.

VAL and STR$ often go together.

LEN(X$) returns the number of characters in a string, X$, for example:

```
10  X$="THE VIC-20 FIRESIDE COMPANION"
20  ? LEN(X$)
```

The result is 29. Note, however, that all spaces commas, colons, and the like will be counted too. That is why the result of the little program is 29 when there are actually only 26 printed characters.

OTHER FUNCTIONS

SPC(x) is a useful function. It inserts spaces prior to printing a string on the screen. It would seem on the surface to do the same as tab, however, there is one very important difference. The following short sequence makes the point:

```
10  ?TAB(10)"DOES THIS"
15  ?:?:
20  ?"WHILE SPC(10)"SPC(10)"DOES THIS"
```

Tab will only read from the left edge of the screen. SPC reads from the current cursor position. This allows for much greater flexibility in the placement of material on the screen. It might well appear to you to be a tiresome detail at this stage, but you can always clean up a program and make it much more readable when the foundation has been laid.

In a later chapter you will be dealing with the astonishing results obtained by the use of formula. Now it can be a tremendous nuisance to have to write out and enter a specific formula each time you wish it to appear in a program. Of course, you can put the formula in a subroutine and call it up when you need it. There is, however, a much, much simpler way. The statement DEF FN will do it for you. I know that DEF FN is not a function itself, but I consider it appropriate to this section because it is a specific type of shorthand, and that is what most functions are.

Here is a fairly simple formula: Y=100+100*SIN(X/20). Could you imagine having to enter it four or five times and then to check through a program for a bug?

Instead, you can simply write:

```
DEF FNY(X)= 100+100*SIN(X/20)
```

Then, when you wish to use the formula, you simply call it by entering the appropriate statement, PRINT or whatever, followed by FNY(X). The value of X can, of course, be altered. The form would then be: FNY(6) or whatever value you wish. Maybe you don't want to put a value in there, but make it subject to an input statement elsewhere. In this case the DEF FN statement must use some character in the parenthesis which does not appear in the formula, thus you would write DEF FNY(B)=100+100*SIN(X/20).

Take careful note of the fact that there are no defined functions for string variables such as A$.

Other functions are mathematical in nature, are fairly obvious if you are skilled in math, occur quite frequently in graphics programs, and need no explanation at this point.

Chapter 9

There will come a time when you feel that the three and a half kilobytes of RAM (Random Access Memory) that are available on the standard VIC-20 are a trifle confining. You will wish to expand and add more memory. RAM is the working area available to you. The more RAM the longer and more complex your programs can be. RAM size is measured in kilobytes, or thousands of bytes.

MEMORY EXPANSION

Long ago (over twenty years ago, which according to my children is the "olden days!") those people who were working with the old-fashioned computers developed a shorthand method of describing what they were doing. They were, of course, programming computers by using binary notation, which you met in an earlier chapter. Binary notation requires that one use binary digits. These soon became known as *bits*, which is short for BInary digiT.

In the same way that we call a collection of one hundred pennies a dollar, a collection of bits was given a name. You have already discovered that a character on the screen is made up of 8 groups of 8 bits. A group of 8 bits is called a byte. To complete the food analogy, computer buffs call half a byte, or four bits, a *nibble*.

Quite obviously a large collection of bytes required a convenient name. Although the name *kilo-byte* seems to refer to one thousand bytes, there are actually 1024 bytes in a kilobyte.

If you multiply 1024 by 3½ you will get 3584. When you switch on the VIC-20, you will notice that it tells you that there are 3583 bytes free. Don't worry about the missing 1½ bytes! You have not noticed the loss up to now, have you?

"But," you will say, "My VIC-20 is supposed to have 5K?!!"

Well yes, it does have 5K. Some of it is used, however, by the system itself, looking after the screen and making sure that everything goes in the right place. Some manufacturers count the read only memory as well as the random access memory when advertising their wares. ROM is the area reserved by the computer to recognize all the BASIC commands and instructions you give to the computer. It is the workhorse of the computer that does most of the work for you. Otherwise you would be coding your programs in bits every time you wanted to use the computer and probably going hairless doing so! The one-and-a-half K you seem to be missing are well used! Some manufacturers of larger computers indicate how much ROM and how much RAM there is.

Now in adding memory to a VIC-20, you can either add RAM alone in configurations of 3K, 8K, and 16K, or you can add 3K plus a superset of instructions. The extra memory comes in the same format as the games you have no doubt bought. The cartridge is slipped into the black slot just behind the power-on light. You must switch computer off while you do this of course.

THE SUPER EXPANDER

For the convenience of VIC-20 owners, Commodore has produced a cartridge called the Super Expander. Not only does this cartridge supply you with 3K extra RAM, but is also makes some extra instructions made available to you to make your programming move a lot faster. As soon as you have slipped the Super Expander into the cartridge slot, switch on the VIC-20. On the screen you will see that there are now 6519 bytes free. No doubt you will do a quick calculation, either on your fingers, your calculator or your computer, and discover that 3583 and 3072 should be equal to 6656! Well, yes, they still are. There seems to be 137 bytes missing! The missing bytes are used by the new instructions that we now have at our service.

Quite a lot can be done with the 6.5K bytes that you now have. The first thing is to discuss a program that makes use of the Super Expander. This program is actually printed in the small manual that comes with the cartridge. The typesetter made one or two errors however which means that the program will not run. Here is the correct version.

```
10  GRAPHIC2:COLOR11,6,6,6:X=170:Y=170
20  J=RJOY(0)
30  x=X+((JAND4)=4)-((JAND8)=8)
40  Y=Y+((JAND1)=1)-((JAND2)=2)
50  POINT3,X*3,Y*3:IFJ=128THEN:SCNCLR
60  GOTO20
```

The first error corrected is the syntax on line 10. There should be no comma after the final '6'; and the second error is the missing '1' in line 40. An alternative version of the program is shown in Appendix J.

You will not be able to resist running the program . . . so go ahead! When you have had enough of that and have also wrested the VIC-20 away from your children, look at the explanation of the program.

The first thing you have done on line 10 is set up the graphic mode by typing GRAPHIC2. This sets the output to the screen in the high resolution mode. You could change the 2 to 1 and see that there are patches all over the screen. Enjoy youself discovering where all the patches are. You will also realize that the vertical lines are twice as wide as when graphic mode 2 is set.

The second thing that is set is the color, actually 4 sets of color: screen, border, character, and auxiliary colors, in that order. You can see from the program that the screen color is set at 11 (in a range of 0-15); the border color is set at 6 (in a range of 0-7); the character color is set at 6 (in a range of 0-7 for the normal mode, 8-15 for the multicolor mode) and the auxiliary color is also set at 6 (in a range of 0-15).

Change the character color to something else in the range of 0-7 and see what happens. Then you can try changing the values of the other parameters in the COLOR statement and see what you get.

Next add line:

```
1 INPUT A
```

and then substitute one of the color values with the variable A. Now you can decide what color you would like the border, screen, character, and auxiliary to be from outside the program itself. If you provide for four lots of inputs, you can control all of the parameters; and if you also provide the fifth, you can set the graphic mode. For example:

```
1  INPUTA :INPUTB:INPUTC:INPUTD:INPUTE    (A is the graphic
   mode,B-E are color values)
2  INPUTX:INPUTY
```

Then remove the X and Y statements at the end of line 10. As each ? comes up on the screen you will enter values for graphic, screen, border, character, and auxiliary colors; followed by two more sets of digits that set the point at which the spot appears on the screen. For example, the current point is in the center of the screen. If you change the values of X and Y to 10, the spot will appear at a point near the top left corner. If you set the X and Y values at 1, the point will be right in the top left corner, just barely visible.

Lines 30 and 40 are our familiar joystick reading expressions. You are probably quite glad that our long NOTPEEK expression has disappeared.

Line 50 might be a bit of a puzzle at first. Although the parameters you have been changing so merrily have affected the color and placement of the dot on the screen, it is line 50 that does the actual work. The point statement tells the computer to make a point. The IF-THEN checks to see if the line goes off the left and right edges of the screen. You will already know what happens if it does! What, then, do the two statements X*, and Y* do?

Change the value from 3 to 1. Now the line is much finer and there are none of those funny blips on diagonal lines. Now change the value up to 6, but be sure to give the same value for each for the time being. Experiment with other values. Keep the value 3 for POINT.

Now try giving different values for X and Y as in: X*7, Y*20 and see what happens. The first thing you will notice is that the starting place for the dot has changed. You will already have noted that you can draw the lines much more quickly with higher values. This can be the basis of a neat little game which could teach a small child to learn how to control the joystick.

You might think that there is no value in a separate program just to teach joystick control. There is a very great value. If a small child tries to learn how to control the stick during a game he or she is likely to become very frustrated because the aim, winning the game or achieving a high score, is not attained. In a program that teaches how to control the stick with no other purpose in mind, all the attention is focused on that aspect. As soon as the line goes off the screen, the program stops. A line that reads the fire button (XAND32) could easily be inserted to start the game again, and the parameters could all be set at random by one of those nice INT(RND(1) expressions! An essential truth can be learned from all this: the best games are simple!

The final thing to note about this program is the statement

SCNCLR which clears the graphic screen. Make a special note of the form of the IF-THEN statement when followed by one of the Super Expander commands. There must be a colon after the word then.

Now for something completely different! Actually it is a blend of previous material but made much simpler by use of the Super Expander. I would like to take a moment to point out to you that you have covered most of what is to be learned about BASIC; at least, you have used most of the instructions. Everything that follows will therefore be familiar to some extent, there being very few commands left, save for those that deal with other peripherals. Those will be dealt with in due course.

Enter the following program:

```
10  ?"[CLR HOME]"
15  ?"THE NAME OF THE PRESIDENT OF THE USA IS....?"
20  ?:?:?"CHOOSE ONLY ONE NUMBER AT A TIME.."
30  ?"1 MARGARET THATCHER"
35  ?" 2 PIERRE TRUDEAU
40  ?"3 RONALD CORBETT"
45  ?"4 RONALD REAGAN"
50  ?"5 HAILIE SELASSIE
60  INPUT A
70  IFA=4 THEN GOTO 200
```

So far nothing happens, yet you can see that it is a simple quiz and that if the correct answer is given the program will move to a subroutine.

Now for the clever bit! You will recall that earlier in this book you went to a great deal of trouble to produce music. The Super Expander can do wonders with sound too!

Type the following line very carefully

```
100  ?"([CTRL]and [reverse arrow], which is at the top left of
     the keyboard)T8V9S2032CRRS101GG#GRGRRRBRRS202CRRRRR"

               ([CTRL]and [reverse arrow] give a reverse F)
110  GOTO10
```

Then type the following line equally carefully:

```
200  ?"([CTRL]and[REV.ARROW])T3V9S303GRRERCRRERRGRRBRRRR
     R
210  GOTO10
300  ?" ?[CTRL]and[REV.ARROW])T3V9S302CRRRCRRCRCRR$
     ERRDRDRRCRCRRCRCRRRRR"
310  GOTO10
```

Then add line 80 as follows:

```
80 IFA=5THEN 300
```

For your convenience, the complete listing is included in Appendix N, program 17.

This program, by itself, can become utterly boring after just a few turns, but it does demonstrate the writing of music in a much easier fashion, if you know something of music, that is. The necessary information must cover knowledge of names of notes and some idea of the duration of notes. The means of access to the music mode is the [CTRL]and]REVERSE ARROW] which must follow a print statement and be inside quotation marks. Then various parameters are set in the following order:
Tempo (T); Volume (V); Speaker (S) followed by number indicating which speaker); Octave (O).

The ranges allowed for each are as follows:

T : 0 to 9 ; S : 1-4 ; O : 1-3 ; V : 0-9.

Having set these parameters you are free to indicate which notes you want sounded by putting in the ordinary letter names by which musical notes are known. As with normal music writing, a sharp # is placed before the note to be modified. The same is true for a flattened note, except that the symbol for flat is the $. Rests, or soundless periods between notes, are indicated by the letter R, repeated for as long as necessary.

Concerning tempo, it is important to note that the higher the figure selected the slower will be the tempo. Try changing the number after the T in each of the statements to produce different tempi. The complete program is listed in Appendix I.

The subject of music on the VIC-20 (and its big brother the Commodore 64) is worthy of a book on its own. There are a couple of programs in Appendix J that make good use of their music propensities, with and without the Super Expander.

Let's explore something else; the draw instruction. Type the following lines:

```
10 GRAPHIC2:COLOR11,6,6,6
15 DRAW 2,26,26TO200,200
```

This produces a diagonal line.

```
20 DRAW2,200,200 TO 200,400
```

This draws a line from the coordinates x=200, y=200 to the new spot x=200 (in other words the same column as before), y=400 or the 400th line down from the top.

```
30  DRAW2,26,226TO200,400
40  DRAW2,26,26TO26,226
```

There you are. You have a side of a box: not very remarkable, certainly, but it is a start.

A range of mountains might be drawn as follows:

```
50  DRAW2,0,1023TO100,820
60  DRAW2,100,820TO110,840
70  DRAW2,110,840TO190,623
80  DRAW2,190,623TO300,790
90  DRAW2,300,790TO460,510
100  DRAW2,460,510TO825,900
110  DRAW2,825,900TO1023,1023
```

These could be the Rockies or the Snowy Mountains or Snowdonia! Now add the following lines:

```
65  COLOR11,5,5,5
85  COLOR11,6,1,6
105  COLOR11,5,5,5
```

Now add the sum in the right hand corner:

```
120  CIRCLE 15,890,100,20,20
```

Followed by rays of the sun shining brightly on the mountain:

```
130  DRAW7,460,510TO870,120
140  DRAW7,825,900 TO910,80
```

And then, to top things off:

```
150  CIRCLE1,500,100,70,30
160  CIRCLE1,600,120,95,30
170  PAINT1,500,100
180  PAINT4,600,120
```

Now, I do not think for one moment that this effort is likely to end up in any art collection. That thing in the left corner looks a little odd to say the least. But you can state one undeniable truth about computers: they will save a considerable amount of work and yet leave much to the process of creativity. You probably won't call computer pictures art, and yet there is no doubt that the entire procedure of putting a complex program, including sound, graphics and so on, together quite a way beyond a mere science.

No picture should be without a title, and so:

```
190  CHAR3,8,"THE HILL"
```

And to complete the whole thing, add appropriate music, and erase that funny object in the corner!

```
200  ?"[inverse F]T7V3S202CCEG"
210  ?"([inverse E])S302CCC
220  ?"([inverse F])S202AA"
```

And to heighten the effect somewhat:

```
195  FORD=1TO2000:NEXTD
```

Finally add a pause before the darkness steals over the sky leaving our mountain range bathed in moonlight.....

```
222  FORD=1TO400:NEXT
225  PAINT6,0,100
```

I know! The whole thing is quite awful! And yet it might cause a chuckle or two.

Line 225 is somewhat of a fake. It doesn't matter what you give as the parameters, you will end up with everything dark over the mountain, with just one cloud and the moon. The only difference is the way in which the darkness wipes on. You might experiment with this. Again, the complete program is listed in Appendix N, program 18.

The sound instruction allows you to set all four speakers at one time in the order S1 to S4, followed by a volume setting between 0 (silence) and 15.

This saves a lot of poking about and allows for simple but effective three part harmony writing. The tempo would be fixed in much the same way as in the data statements you saw in an earlier part of this book. However, the number relating to the duration between sounds must be set in a FOR-NEXT loop. Alternatively, the frequency for each voice, the duration, and the volume can be placed in data statements. Keeping track of it all can be rather complex however.

So far you have laboriously typed in each of the statements you desired. There is a quicker way. On the right of the keyboard there are four light-brown keys. Type the number 10 and then press the key marked f1. The result is : 10 GRAPHIC . You can now add the appropriate digit, follow it with a colon, and then press shift and the f1 key which has f2 marked on the front. This gives you access to f2 and the result is COLOR ! Easy isn't it?

Each of the programmable function keys has been preset to

provide seven of the new commands available to you in the Super Expander. However, some corrections must be made to the statements in the manual that comes with the Super Expander. You will no doubt have discovered them for yourself in a very short space of time, but here they are in any case.
f1=Graphic, f2=Color, f3=Draw, f4=Sound, f5=Circle, f6=Point (not Run+Return), f7=Paint (not Point), f8=List+Return.

Use of these programmable, or in this case, preset keys saves a lot of time. If you wish to check at any time the exact value of the function keys merely type, in direct mode, the word KEY and press the return key.

The function keys are even more useful than that! They can be programmed from within a program to output whatever you wish. Normally this would be done by a line such as:

```
10  IF A$=CHR$(133) THEN A$="EXACTLY"
```

CHR$(133) is function key 1, or f1.
With the Super Expander one need only type:

```
10  KEY1,"Exactly"
```

and thereafter, until you switch the computer off, that key will contain only those contents. If you wish to use CHAR or SCNCLR instead of LIST+RETURN, you would merely change the contents of key 8 to conform with your wishes. Simply type the word KEY alone to find out what those contents are, just in case you have forgotten!

In normal use, you would perhaps begin a program in normal text mode either going to a graphic mode as the program progresses or using subroutine. It is important that the return to text mode be indicated by use of the instruction GRAPHIC4. Take great care that you keep track of the mode you are in for if you are already in text mode and you attempt to use GRAPHIC4, you will crash the system and lose your program. This is only likely to happen while you are developing a program, however, so it is then that you should be most careful.

Quite clearly the Super Expander is a highly useful peripheral allowing you to provide a large degree of polish to your programs with the least amount of effort.

CAPSULE REVIEW

The working space in a computer is known as RAM, which stands for random access memory. The VIC-20 comes with about 3½ thousand bytes of working area, with which you can do some remarkable things, but sooner or later there will come a time when you will need more RAM. A very convenient way of both adding RAM and adding a superset of the BASIC language, allowing for more tricks without a lot of hard work, is to buy the Super Expander.

The name of the device must reflect the fact that not only is there more RAM, actually 3 more kilobytes, but also an extra set of instructions that deal with graphics, sound, joysticks, and paddles. Ten instructions that avoid the use of extensive pokes and peeks are now available. They are:

GRAPHIC	sets up graphic display
SCNCLR	clears a graphic screen
COLOR	sets up screen, border, character, and auxiliary colors
REGION	selects a character color
DRAW	plots lines between points
POINT	plots single points
CIRCLE	draws circular and related figures
PAINT	allows colored areas
CHAR	sets text on a screen full of graphics
SOUND	sets all four speakers at one time

In addition, the Super Expander makes a system of writing music, which allows the use of note names, available. This system is closer to normal musical practices than the assignment of numerical values that bear no relation to frequency or pitch.

Seven functions are also made available, the most useful of which are RJOY(x) and RPOT(x), which deal with the joystick and the game paddle respectively. The game paddle process is to some extent simpler than the joystick to manage and understand. As with everything else, practice makes perfect!

There are four graphic modes, 0 to 3, and a fifth, labeled 4! which returns the program to text mode. The first mode, 0, is the normal text mode, however, setting the mode to zero after some other mode has been in use will not work. The mode 4 must not be used if the computer is already in text mode or else the system will crash, and resetting by turning the VIC-20 off and then back on, is required—which is most upsetting.

COLOR controls four registers, each defined by a number and placed in the order screen, border, character, and auxiliary. The settings for each are limited as follows:
screen — 0 to 15, border — 0 to 7, character — 0 to 7 in Graphics 0 mode and 8 to 15 in multicolor mode, auxiliary — 0 to 15. The fifteen colors and shades available are as follows:

0	black	8	orange
1	white	9	light orange
2	red	10	pink
3	cyan	11	light cyan
4	purple	12	light purple
5	green	13	light green
6	blue	14	light blue
7	yellow	15	light yellow

It would appear from this that there are 15 hues, shades, or colors available for the characters. This is not so because the numbers 8 to 15 are used in place of 0 to 7 in multicolor mode.

This may best be illustrated by the following short program.

```
10  GRAPHIC 1:COLOR0,0,2,6              Sets screen to
                                        black, border
                                        to black, char-
                                        acter to red, and
                                        aux to blue
20  CHAR2,2,"THIS IS IT"                Text appears in
                                        row 2 starting
                                        in column 2
30  REGION7                             Affects charac-
                                        ter color, which
                                        is yellow here
40  CHAR4,3, "THIS ISN'T"
50  REGION 5
60  CHAR 6,2,"THIS COULD BE IT"         Character blue
70  REGION3
80  CHAR8,4,"BUT IT ISN'T"              Character now
                                        cyan
90  REGION5
100  CHAR10,3,"IS IT?"
110  REGION 10
120  CHAR12,5, "WELL NOW!!!"
```

Changing the GRAPHIC value will demonstrate various effects. Be warned that you may see slightly different shades resulting not only from your set but also from the conditions under which you view the screen. Some sets will show a kind of shimmer, an effect, which under certain circumstances (as in the formulaic images in the next chapter, for example), can be quite charming. Both green and amber screens can be simulated by use of the appropriate graphic mode and color set.

The screen is divided into pixels 1024 across and 1024 down for the purpose of addressing points. This nomenclature is translated by the VIC-20 into the actual resolution. Thus the center of the screen is 512,512. The first number refers to the horizontal axis and the second to the vertical. This point can be either a single spot, when the point instruction is used, or the center of a circle, ellipse or arc.

Lines can be drawn from point to point by using the draw instruction. In each case the first number given is the horizontal and the second the vertical. Thus DRAW1,200,200 TO300,300TO500, 400TO600,300TO200,200 will produce an odd shape, yet include the appropriate points.

The paint statement fills enclosed areas with the chosen color, yet care must be exercised or else the entire screen will be painted whether you want it to be or not! The mountain scene in this chapter is just such an example.

Seven of the graphics instructions are preset like organ stops. They are accessed by pressing the light brown keys, known as function keys, to the right of the main keyboard. The values of these keys may be discovered by typing the word KEY and pressing the return key. Key 8 allows for rapid listing of any program, whether a graphic program or not.

The values of the function keys can be altered to suit your needs. An often used instruction or function, such as RJOY, can be assigned to one of the keys for an entire programming session. The value will revert to that of the original setting when the computer is next switched on. The function keys can, of course, be used in the normal fashion, which will be covered in another chapter.

Using the Super Expander the sound function is very similar in operation to the graphics function. Volume is set by assigning a value to V, and tempo, by assigning a value to T. The speakers are labeled S1, S2, S3, and S4, when the Super Expander system is used. The Super Expander system is reached by typing PRINT followed by the CTRL and left arrow keys (not left cursor!). T, V, and S are then set, in that order followed by the octave selection O (the letter, not the number!). Notes are named as in conventional music: C, D, E, and so forth. Accidentals (sharps and flats) are placed before the note, exactly as in musical notation. Note that only seven notes of the diatonic scale are available. The octave must be changed in order to reach notes lower than C or higher than B.

Chapter 10

Magazines contain lots of advertisements for computers and lots of those advertisements show a screen with clever patterns on them. Those clever patterns are well within your reach with the Super Expander.

USING FORMULAS WITH THE SUPER EXPANDER

Type in the following program:

```
10  GRAPHIC2 :COLOR2,3,4,5
20  FORX=1TO1023:Y=100+50*SIN (X/20)
30  POINT5,X,Y
40  NEXT X
```

When you have run this program, try changing the values in the expression Y=; first change the 50 to 20. The wave is shallower, that is, the amplitude is less. Now increase the value to 100. You are in danger of going over the top of the screen, therefore change Y to =100+100*SIN (X/20), thereby bringing the wave down to the middle of the screen.

Now add line 15:

```
15  Z=500
```

and then substitute the first number in Y for Z. Your line should now read:

```
Y=Z+100*SIN (X/20)
```

Then add:

```
50  Z=Z+100
60  GOTO20
```

Now you have two waves, 100 points apart. At least, you will have only two waves if you stop the program at the end of the second one. The GOTO 20 ensures that the computer will continue to draw waves forever!

What about something a little more decorative? Change the Y expression in line 20 to read

```
Y=Z+SIN (X/30)*COS (X/15)*50
```

Doesn't that look just a little like the hat shape you see in those ads?

Now change X to read FORX=200 TO 450. I know it looks a little peculiar to start a FOR-NEXT loop at a point other than 1, but don't forget that X is a starting point on the screen. When you have seen what that does make the following alterations:

```
16  A=1
19  FORI=1TO10
18  V=30 :H=15   (I know that is out of order, but it makes the following a little
                 clearer.
```

In line 20 change Y to read:

```
20  Y=Z+SIN (X/V)*COS (X/H)*50
30  POINT A,X,Y
50  X=Z+20
60  V=V-1 :H=H+1
65  A=A+1
70  NEXTI
```

This time the patterns change very slightly and the color also changes. The complete listing is included in Appendix N, program 19.

The process is a trifle slow because we have to use the point command. The draw command produces straight lines.

Let's explore the formula process a little more, for that is what does all the fancy work for us. Try this one:

```
10  GRAPHIC2: COLOR1,3,5,7
20  R=.8:PI=pi/30:P2=2*pi/3   (pi is that magic number 3.142 . . . found just
                              next to the restore key. Press the shift key and
                              this key.)
30  FORT=0TO4.08 STEP PI      (you can change the 4.08, but make it no
                              larger than about 4.2)
```

```
40  R=R*1.17557
50  X1=COS(T)*R+500:Y1=SIN(T)*R+500
60  A=T*P2
70  X2=COS(A)*R+500:Y2=SIN(A)*R+500
80  DRAW1,X1,Y1TOX2,Y2
90  B=T+2*P2
100  X1=COS(B)*R+500:Y1=SIN(B)*R+500
110  DRAW1,X1,Y1TOX2,Y2
120  X2=COS(T)*R+500:Y2=SIN(T)*R+500
130  DRAW1,X1,Y1TOX2,Y2
140  NEXT
```

Make absolutely sure that you have entered every line exactly. You might think that such an admonishment is unnecessary, yet you would be surprised how often a simple typographical error can cause a great deal of grief and frustration. To give you an example: in using the function keys on the VIC-20 Super Expander, I somehow got into the habit of placing a comma after the graphic command. I would type in my program and lean back with the greatest of satisfaction as I typed run and then pressed the return key. You can imagine my annoyance when the program would crash and the horrid message SYNTAX ERROR in 10 would appear on the screen. I'm a quick learner, though: now I know straight away what the problem is when the message appears!

When you first run this program, you might think that nothing happens. It begins very slowly with a dot, and then another and then a few more and then—well, you can find out for yourself.

Here's another:

```
10  GRAPHIC2:COLOR1,3,6,7
20  A=100:B=180:C=90
30  AN=360/A
40  FORN=1TOASTEP4:T=N*AN
50  X=B*COS(T)+200:Y=C*SIN(T)+150
60  DRAW1,800,800TOX,Y
70  NEXT
```

You can have a lot of fun with this one by changing the variables A, B, and C. Some of the numbers you choose will produce the ILLEGAL QUANTITY ERROR message. Don't worry about it, just change them to something else. Better still, put a GOSUB at 20 leading you to, say, 100. Write a few lines allowing you to input values for the variables. Don't forget the return statement. At 90 you could put a short duration loop followed by a screen clear and a GOTO sending control to the beginning again. You or anyone else could have fun for hours with this one.

The next program is not very long, and yet the product takes an astonishing amount of time to appear on the screen. I must confess I have not seen the complete outcome myself. I think I know what the result should look like. I do know what the first ten minutes look like. I suggest that you let it run overnight, or sometime when you are away for the weekend. You could give it to your VIC-20 when you are cross with it and want revenge!

```
10 GRAPHIC2:COLOR0,5,7,7
20 P=490:Q=450
30 XP=300:XR=2.2*π
40 YP=100:YR=1:ZP=75
50 XF=XR/XP:YF=YP/YR:ZF=XR/ZP
60 FORZI=-QTOQ-1
70 IFZI<-ZPORZI>ZPGOTO150
80 ZT=ZI*XP/ZP:ZZ=ZI
90 XL=INT(.5+SQR(XP*XP-ZT*ZT))
100 FORXI=-XLTOXL
110 XT=SQR(XI*XI+ZT*ZT)*XF:XX=XI
120 YY=(SIN(XT)+.4*SIN(3*XT))*YF
130 GOSUB170
140 NEXTXI
150 NEXTZI
160 STOP
170 X1=XX+ZZ+P
180 Y1=YY-ZZ+Q
190 POINT2,X1,Y1
200 IFY1=0GOTO220
210 POINT2,X1,Y1-1,X1,0
220 RETURN
```

Try variations on the variables in the first two or three lines to see if you can produce different varieties of the object. Can you make it taller or wider, or redesign it altogether? With astute alteration of variables it is possible to change the size of any of the results of the programs in this chapter. You can put more than one pattern on the screen at once.

Another trick you might try with this program is to alter the color so that the object is shaded. Incidentally, they say that the best type of screen is one which produces an amber image on a dark background. Here you have it!

Finding a use for the programs beyond a means of generating one or two pretty patterns might seem difficult. However, consider that the meat of the program, the thing that makes the program work, is the formula that drives the points all over the screen.

Although the topic of computer assisted instruction is a large one worthy of separate treatment (a book will appear on this subject for the VIC-20 and Commodore 64), consider for the moment a teaching program that demonstrates, in visual form, the results of particular formula.

Perhaps you might like to try your hand at the production of a program that is menu driven. The student could select each part of the program either by choosing the result: "A fan-shaped figure"; or by choosing the formula. By allowing the students to enter values for the variables with the input statement, various effects could be noted and taught. Warnings about values that produce incongruous results or which cannot be handled could be given so that the student does not find error messages appearing on the screen.

Start with a simple program that allows the student to produce rectangles, circles, ellipses, parabolas, and so on, and move on to boxes, kites, and spheres. Again, it is your imagination that counts.

Just before we leave the graphics, consider the following program:

```
10  GRAPHICS3: COLOR0,0,1,7
20  POINT1,20,20,225,50,100,275,150,250,200,225,30,500,275,450
30  POINT1,150,300,150,320,150,340,151,341,149,321,150,322
40  CHAR18,2,"ORION"
50  POINT1,300,360,200,280,180,10,181,14,179,12
```

It is a time-consuming task to plot each of the stars, and yet one could produce a fine catalogue of constellations to look at on those rainy, stormy, or foggy days. You will notice that the image even twinkles a little! A simplified version of the program is found in Appendix N, number 25.

CAPSULE REVIEW

The most extraordinarily beautiful images can be created on the VIC-20. Whether or not we are entitled to give the accolade of ART to the results of the settings of pixels is a moot point, and one which will be hotly debated for some considerable time to come. If art is that which provides some form of visual entertainment via the stimulation of the emotions, then certainly one can describe many of the images as art.

An artist is also a scientist (the converse is also sometimes true) in as much as he must contend with the reaction of materials to the visible light spectrum and balance and contrast his materials to produce the effect he desires. So far as the computer is concerned the designer makes some assumptions, acts upon them, and then judges the result. Quite fetching effects can be produced upon the screen by the astute use of the discoveries of that branch of science known as Mathematics.

Now, math might be to you as secret an art as it is to me, yet simple application of mathematical formula can produce a magic quite fascinating. An expression consists of two fundamental parts, one on each side of an equals sign. For our purposes the expressions are quite simple, being the reduction of a complex formula to a single variable that is then used to manipulate the setting of dots on the screen.

The artist may not know the precise frequency that his hues vibrate at, and yet he can use the result with stunning effect. So with formula: you might now know the difference between a sine and a cosine, and yet the simple application of them can entertain you for hours and perhaps teach you, as they have taught me, something about their mysteries.

A simple expression will actually produce only one point on the screen. The trick to shape production is to progress or advance the point by altering some variable in the expression. Some alterations will produce error messages on the screen, either before or during a program run. Do not let this disturb you, but rather let it cause you to persevere in your efforts.

Fortunately most forms of BASIC include what amounts to shorthand versions of important mathematical functions. The computer was, after all, developed as a computational device to aid the process of mathematics. Mercifully, ordinary folk soon saw that the computer could be put to higher use than of mere number crunching and so were able to turn the tables in the favor of true creativity.

Many images take a considerable amount of time to draw, and it might prove rather frustrating to have to go without the use of your computer, as it were, while it does its party tricks.

The images, and indeed the type of images, the computer can draw are legion. With a large amount of patience, it is possible to produce a fair representation of almost anything, from a car to a picture of a beautiful girl. Your efforts will be of great interest to me and could possibly be published in an edited collection. You may correspond with me through the publisher.

Chapter 11

Man is a highly competitive creature. In some forms he is also a highly incurious creature. I'll wager that the majority of folk who have bought a computer did so because the children did some persuading!

Now, children, when they are given the chance, will absorb vast quantities of information very rapidly. Then they like to indulge in some form of recreation that still causes them to pit wits against one another either directly or via some device.

COMPUTER GAMES

Computer games are the in-thing. You, like me, might become totally bored after the first three minutes of any computer game, and yet you will have demands made upon you to either buy such games, in which case you had better go out and buy another VIC-20, or have great fun with your computer writing them yourself.

Computer games fall into two main categories: those that are verbal and those that are graphic. Verbal games are more of a challenge, perhaps, to computer buffs because they cause the player to think, access, make judgments, and act upon then.

Graphic games rely more upon physical reaction to situations. In recent months, there has been a tendency to introduce a great deal of graphics in those games that are fundamentally verbal in order to enhance the sense of realism. Monopoly® and Risk® are games that could be just as easily played without a board, and yet there is some great feeling of satisfaction in picking up your top hat or motor car and counting off the spaces during a turn that lands you on your own property! And how sweet the delight of the other players when you land on theirs!

You are going to develop a game from the top down. The game is a version of one of those games in which you are required to input a certain piece of information based upon information presented to you on the screen. Your information causes a certain situation to arise, and there is a result of some sort. The situation you begin with is that you are about to land upon some alien planet. You are a certain height above ground, and you are required to land the device yourself. You mostly crash! The game is well-known and goes under the name of Lunar Lander or some such. I call this version Jupiter Rendezvous. You will work up the base program and add a few graphic images to smarten the thing up a little.

```
  5 PRINT"⊐"
  6 PRINT:PRINT:PRINT:PRINT:
 10 PRINTTAB(2)"JUPITER RENDEVOUS"
 15 FORD=1TO2000:NEXTD
 16 GOTO2000
 17 COLOR1,3,6,0
 19 PRINT"⊐"
 20 PRINT"WOULD YOU LIKE INSTRUCTIONS?"
 25 INPUTA$:IFA$="YES"THEN GOSUB1000
 26 INPUT"NAME PLEASE";N$
 30 PRINT"⊐":PRINT"GOOD......LUCK!!!"
 35 FORD=1TO2000:NEXTD
 40 PRINT:
 45 TI$="000000":H=500:V=50:F=120
 50 IFB=0THEN65
 55 PRINTTI$  ;TAB(4);H;TAB(12);
 57 PRINTV;TAB(4);F;TAB(12);
 58 PRINT"I";TAB(H/4+18);"**"
 60 GOTO70
 63 PRINTTAB(4)"TIME";TAB(12)"HEIGHT"
 65 PRINTTI$  ;TAB(4);H;TAB(12);
 66 PRINTTAB(4)"SPEED"
 67 PRINTV;TAB(4);F;TAB(12);
 68 PRINT"I"TAB(H/4+18);"**"
 70 INPUTB
 75 IFB<0ORB>39THEN150
 80 IFB>30THEN150
 85 IFB>FTHEN95
 90 GOTO100
 95 B=F
100 V1=V-B+5
105 F=F-B
110 H=H-.5*(V+V1)
115 IFH<=0THEN160
125 V=V1
130 IFF>0THEN50
```

```
 135 IFB=0THEN145
 140 PRINT"⊔⊔⊔OUT OF FUEL⊥⊔⊔"
 145 PRINTTI/60;TAB(4);H;TAB(12);
 147 PRINTV;TAB(4);F;TAB(12);
 148 PRINT"I";TAB(H/4+18);"***"
 150 B=0
 152 FORD=1TO3000:NEXTD
 160 PRINT"**WELCOME TO JUPITER**"
 165 FORD=1TO3000:NEXTD
 170 H=H+.5*(V+V1)
 175 IFB=5THEN190
 180 D=(-V+SQR(V*V+H*(10-2*B)))/(5-B)
 185 GOTO195
 190 D=H/V
 195 V1=V+(5-B)*D
 200 PRINT"JUPITER AT";T+D;"SECONDS"
 205 PRINT"LANDING SPEED=";V1;"FT/SEC"
 210 PRINTF;"UNITS OF FUEL LEFT"
 212 FORD=1TO3000:NEXTD
 215 IFV1<>0THEN2500
 220 PRINT"GOOD SHOW...."
 225 PRINT"HAVE YOU THOUGHT HOW YOU WILL GET BACK?"
 227 FORD=1TO3000:NEXTD
 230 IFABS(V1)<5THEN245
 235 PRINT"JUPITER PILOT ";N$;" HAD A NASTY ACCIDENT
     TODAY"
 236 PRINT"WHILE FAILING TO LAND SAFELY ON JUPITER"
 240 PRINT"HE IS SURVIVED BY......."
 245 PRINT"WOULD YOU LIKE ANOTHER MISSION?"
 250 INPUT A$
 255 IFA$="YES"THENPRINT"GLUTTON FOR PUNISHMENT, AREN'T
     YOU?"
 256 FORD=1TO3500:NEXTD
 257 IFA$="YES"GOTO30
 260 PRINT:
 265 PRINT"SEE YOU"
 270 PRINT:
 275 END
1000 PRINT"   YOU HAVE APPROACHED THE PLANET JUPITER
     AND ARE READY TO LAND"
1001 FORD=1TO2000:NEXTD
1002 PRINT:PRINT:PRINT"    UNFORTUNATELY YOUR
     COMPUTER IS ONLY     PARTIALLY FUNCTIONAL"
1003 PRINT"DO YOU UNDERSTAND?"
1004 INPUTA$:IFA$="NO"THENGOSUB1500
1005 PRINT"⊐"
1006 PRINT"THE COMPUTER CAN GIVE YOU READINGS ON RATE
     OF DESCENT, FUEL ";
1007 PRINT"     REMAINING AND OTHER     MESSAGES"
1008 FORD=1TO3000:NEXTD
1009 PRINT"⊐"
1010 PRINT"THIS MEANS...............YOU MUST
     LAND THE    CRAFT....YOURSELF"
1011 PRINT:PRINT:PRINT"AFTER  EACH SECOND OF
     FLIGHT YOU WILL BE    TOLD YOUR HEIGHT,";
1012 PRINT" YOUR        SPEED AND FUEL"
1013 FORD=1TO2000:NEXTD
```

```
1014 PRINT"A ? WILL APPEAR."
1015 FORD=1TO4000:NEXTD
1016 PRINT"□"
1017 PRINT:PRINT:PRINT"   YOU MUST ENTER
     THE NUMBER OF UNITS OF    FUEL";
1018 PRINT"YOU WISH TO BURN"
1019 FORD=1TO2500:NEXTD
1020 PRINT"SHOULD YOU CRASH THE    APPROPRIATE
     OBITUARY WILL APPEAR"
1025 RETURN
1500 PRINT"□"
1502 PRINT:PRINT:PRINT"THAT MEANS:...........
     THE COMPUTER WORKS.....BUT ONLY JUST!!!"
1503 FORD=1TO3000:NEXTD
1505 RETURN
2000 GRAPHIC3:COLOR0,0,5,2
2005 CIRCLE3,500,500,200,300
2007 CIRCLE2,575,575,20,18
2010 PAINT2,500,500
2030 CHAR2,5,"JUPITER"
2035 CHAR19,4,"RENDEVOUS"
2040 FORD=1TO3000:NEXTD
2045 GRAPHIC4
2050 GOTO17
2500 GRAPHIC3:COLOR6,5,1,7
2505 CIRCLE1,500,1023,500,50
2510 DRAW1,450,973TO592,930TO603,940
2520 DRAW1,603,940TO641,980TO682,960TO702,973
2530 CHAR4,2,"C R A S H"
2540 POINT7,420,920,405,902,590,955
2550 CIRCLE7,700,700,5,5
2560 FORD=1TO5000:NEXTD
2570 GRAPHIC4:COLOR1,3,6,0:GOTO230
```

That is just about the longest listing you have seen so far, but do not let that deter you from entering it in using the keyboard. As you do so examine each line making sure that you understand what is happening.

The program is designed so that you can alter it to suit your own needs and preferences. Sound would be a good addition to this program. Consider writing a sequence that is somehow tied in to the speed, falling in pitch as the craft falls and rising if the craft begins to rise. You could have some magnificent sound effects for the crash (have you noticed how most of the programs in this book have an alarming tendency to crash?) coupled with some animation. The anthem of your choice could play if you are successful!

The program puts the instructions out of the way at line 1000. In this way the answer NO allows the program to drop through normally. This is a useful trick and saves a whole line. Quite simply, you check only for the positive response; the negative response requires no action. The trick saves time and some memory too.

You will note the use of some of the functions you dealt with in the last chapter:TI$, for example, both in the initialization form and the reproductive form.

You will have noticed in earlier programs how some variables are given labels that are either two letters or a combination of a letter and a number. This is an extremely useful ploy for it allows you to keep precise track of each variable with a mnemonic to help you. H obviously means height and V obviously means velocity. F, of course, means Fuel.

ARRANGING TEXT ON THE SCREEN

In developing such a program it is useful to be able to compartmentalize each portion, running it on its own to make sure that everything functions and is reasonable. One problem you might have to deal with is that of putting text on the screen so that words do not become split between lines. As you enter the text on the screen you will notice that a long sentence will go on to the next line. For example:

```
10  ?:?" This is a rather
long sentence"
```

On the screen, the word LONG will be attached to RATHER, and the S of SENTENCE will sit at the end of the screen. It can be tedious to edit these sentences when a little care can solve the problem before it begins, as it were. First, get into the habit of pressing the space bar between words. Typists do this automatically. Second, get into the habit of watching where the quotation marks occur in the line above. Any letter directly underneath the quote marks, regardless of whether there are two or twenty print or other commands in the line, will sit at the end of the screen as in our example. To correct the matter now, all that is necessary is to bring the cursor up to the *s* and insert a space.

If you use a tab function, things can get a little more tricky. Take the following example:

```
10  ?:?TAB(3);"THIS IS A
RATHER LONG SENTENCE"
```

On the screen this looks as follows:

```
    THIS IS A RATHER LO
NG SENTENCE
```

The rule for correcting such a situation before it begins is to use the quotation marks again, bringing the beginning of any word that is likely to wrap around to a point one place to the right of the quotation marks, like this:

```
10  ?:?TAB(3);"THIS IS A
RATHER        LONG SENTENCE"
```

Do this only if you want to justify the left edge, of course. If you want to use the left-most edge of the screen, the rule is to bring the beginning of the word to a point one space less than the number in the tab to the left of the quotation marks, like this:

```
10  ?:?TAB(3);"THIS IS A
RATHER   LONG SENTENCE"
```

The result will look like this:

```
    THIS IS A RATHER
LONG SENTENCE
```

This keeps at least one space on the left of the screen. The screen can be very difficult to read if it becomes too cluttered, particularly if small children are the intended audience.

USING COLOR

Color enhances the output greatly. You can use different colors very easily to highlight certain parts of your program: warnings in red, congratulations in blue or in green, whatever you will.

Let's take a simple example such as our names program.

```
10  ?"WHAT IS YOUR NAME?"
20  INPUTA$
30  POKE 646,3
40  ?"HELLO ";A$
```

The result is a nice green HELLO JOHN or whatever your name might be.

Let's explore this just a little further: Add the lines which follow:

```
15  FOR A=1TO8
50  FORD=1TO4000: NEXT:NEXT
```

and change line 30 to read POKE646,A

At first nothing seems to happen, and then the HELLO portion comes up in different colors, moving through the range from 1 to 8. Nothing happens at first because the first color is white, which will not show up on a white screen. Just remember that the all-important poke here is 646, followed by the appropriate number or variable.

This approach is tremendously useful in teaching or in filing programs, where you can color-code various messages. All correct answers can be in one color and incorrect ones in another; the menu prompts can be in one color, your input in a second, and any incorrect input, illegal quantity, and so forth can be in a third. The result of any computation can be in a fourth color. Even balance sheets can be done with the appropriate black and red. How about purple for your income tax—to match your rage?

Note that all of this can be done without the Super Expander. By astute use of the material in chapter IV, in which we explored the generation of special characters, you can place any sort of character on the screen in whatever color you wish and even make the characters change colors. To be sure the whole procedure is a lot easier with the Super Expander but it is not impossible without it.

An excessively stupid program follows:

```
5  ?[CLR HOME]"
10  ?"WHAT IS YOUR NAME?"
20  INPUTA$
30  POKE 646,1
40  ?"HELLO ";A$
50  FORD=1 TO 2000:NEXT
60  POKE 646,2
70  ?"WOULD YOU LIKE TO SEE WHAT YOU TYPED?"
80  INPUTB$
90  IFB$="YES" THEN 40
100  ?"OK ";:POKE646,3:?"BE ";
110  POKE646,4:?"LIKE ";:POKE 646,5:?'THAT"
```

PROGRAMMING STRATEGIES

When you see programs in magazines or books, you might at first be struck by the apparent complexity of the listing. The point of most of the programs in this book is to demonstrate that quite clever, effective, and useful programs can be created with very simple programming means. I find that I tend to become a little conditioned by the format of programs as printed in a textbooks, and although such programs are very useful, they tend to stymie my thinking. I feel that a nod is as good as a wink and that a suggestion, without giving a large amount of detail, can often be a better means of getting going on your own programs than the detailed output of someone else's mind.

Let us take the game situation as an example. Have you ever seen a really different game in the arcades? Most games are nothing more than variations on the same sets of themes: open a door and make a choice; avoid the damaging blips while shooting down the blobs; find your way through the maze of tunnels, archways, caverns, or whatever.

To be sure such games make for a great deal of enjoyment but they depend largely upon skill with a joystick and free movement of the thumb. Another major feature of computer games is that they tend to isolate individuals. Why not try to develop a truly group game?

On the other hand, you must consider that the computer, any computer, is a great way to commune with your own thoughts. The challenge of programming a computer, whether you are producing a simple mailing list or struggling with a vastly complex game, is one that is hard to beat.

One of the problems you will face is that of dealing with other folk who want to see what you have produced. A few rules are in order here. Never, but never, try to develop a long program while someone is looking over your shoulder, particularly if that individual thinks he or she knows a little bit about programming. Work things out on your own; refine them in solitude; and only demonstrate your product when you know it runs well. Make sure that there are no bugs whatsoever in your program for the unknowledgeable fellow will be quick to tell you what you ought to have done.

Ignore folk who think you know nothing merely because you cannot immediately sit down at his new XYZ computer, straight out of the box, and program it! Computers are individuals, and although there are similarities, the differences will be quite enough to throw even the expert. It was much simpler in the days when almost all programming was performed in machine code.

Make sure that the instructions you give are more than adequate. Such instructions should be sufficient to allow someone to operate the program without you looking over his shoulder.

You will notice in many programs in magazines that there are strange lines with the word REM at the beginning. This is a highly useful device in long programs for it allows you to keep track of what each section of the program is supposed to do. So far, it has not used it in this book because none of the programs have been that long. REM stands for REMARK (or it could be REMinder, but no matter). You can type what you wish after a REM. It will not appear on the screen, but it will appear in the listing. The problem with REM is that it takes up memory, and if you do not have very much to start with, that can be a bother. As remarked, REM is not absolutely necessary in short programs but is highly useful in long ones, particularly if you do not use a program for some time and wish to upgrade it in some way. If you have forgotten what a portion of the program does or why you had programmed that particular section in just that fashion, you can end up with a proper mess!

There is a way around this problem of course, and that is to buy a printer, keep a couple of copies of the complete listing including REMs, and then remove the REMs from your working and running copy.

Always make back-up tapes of any worth-while programs. Make sure you keep the tapes well away from any magnetic influence. You will be surprised the awful damage even a stray screwdriver can do to a tape: the wicked part is that you will never know anything has happened until you try to run the program!

Shelves are favorite places for boxes of tapes to be kept, and yet it is behind those walls that electricians are fond of placing ac lines. Do you recall those experiments in school in which you placed a small compass near a wire carrying current and the needle would fluctuate wildly back and forth from just the 1½ volts given out by a dry cell? Be warned!

Keep tapes, disks, and so forth in well sealed containers so that no dust can get at them. Head cleaning tapes and disks are available and should be used once a week if you are an occasional computerist and more often if you are a mad enthusiast.

Concerning enthusiasm: do not carry on into the small hours of the morning struggling with the same problem. You will end up with a nasty case of burnout. Save partial programs. Do not try to bring a program to completion before saving it, for many strange things can happen. Fluctuations in the ac line can cause your system to crash. A fierce snow, wind, or rain storm can bring lines down with resultant loss of power—and all your work!

ADAPTING PROGRAMS FOR USE ON THE VIC-20

Programs that you will find in magazines and books are printed for your use. There is no doubt that a lot of money can be saved by entering such programs into your computer by hand. All that is needed is a great deal of patience and an even larger amount of understanding. Very often such programs will need some modification before they will run on your particular machine. Check to see what the author has said about the program before attempting to enter it. Make a note, if the author has not done so, of the purpose of variables and how they are used. Check to make sure that you understand the progress of each section of the program thoroughly before entering it. It goes without saying that you will make a note of alterations that are necessary for your machine.

Do make sure that there is enough memory in your machine to run the program efficiently. There is nothing worse than typing in a vast program without bother only to see the message OUT OF MEMORY as you attempt to run it.

Programs are transferable from the PET/CBM to the VIC-20, and vice versa. You can, with care, load tapes produced on either machine.

It is a good plan to indicate on any tape the name of the program, in the precise form in which you saved it, together with details of the number of bytes used. To find this, simply enter PRINT FRE(0) to get the number of bytes remaining and subtract that from the number you started with. The initial memory will depend upon the use of expansion cartridges. Be careful when using such expansion however, as the start location of programs alters accordingly. More details on this can be found in Appendix K.

When copying programs from printed sources, it is only courteous to place the author's name at the beginning. He or she wrote it, did all the work and possibly the cursing, and you reap the benefit! It is a little unfair to then make copies for your friends. Let them buy the book or magazine for themselves and allow the author some return benefit.

As remarked earlier, most games follow set lines, and if you can come up with a new twist, go ahead. Just do not try to market someone else's program as your own. Many ideas will occur to individuals who have never met at about the same time. It is unlikely, however, that they will program the material in precisely the same way, although that could happen!

The programs in this book are yours to do what you will. They are all available on both tape and disk, but only to save you the bother of typing seemingly endless streams of lines. Each program is an example of a particular programming technique. The programs are your springboard.

Chapter 12

This chapter will explore graphics a little more. I have chosen to use the Super Expander in order to get things going faster and so that you can begin to produce more effective displays than if you were using extensive peeks and pokes. When you have a thorough understanding of the proceedings, you can experiment for yourself without the expander; although I'll wager that you will soon have added more bells and whistles to your machine, including a mother board that will allow for vast expansion of memory.

ANIMATED GRAPHICS

First, let's look at a simple program that displays a special effect. This one does not need the expander and is nothing more than a version of a program that appears in the user's manual.

```
10  I=0
20  A=8+8*SIN(I)
30  PRINT TAB(A);"X"
40  I=I+.3:GOTO20
```

This little program does nothing more than produce a sinuous curve, called a sine curve, of Xs on the screen until you stop it. You can try playing with the numbers and indeed could allow the numbers that control the action to be put in as variables. A neat program demonstrating animated graphs could be developed from this.

The following program uses the Super Expander to explore another effect.

```
10  GRAPHIC 2:COLOR0,0,3,0
15  X=200:Y=300
18  FORI=1TO100
20  POINT 3,X,Y
25  FORD=1TO 20: NEXTD
30  POINT0,X,Y
40  X=X+5:Y=Y+5
50  NEXTI
55  GOTO 15
```

This program displays a point that moves down the screen diagonally and then disappears only to return to the starting point to do it all over again. You can change the starting point by altering the values for X and Y; you can alter the angle by changing the increment values for X and Y on line 40. By starting X and Y at high numbers and decrementing X and Y, you can make the dot move upwards from left to right or from right to left. Remember that for forward motion, that is from left to right, X must be a low figure, and for reverse motion X must be a high figure. For downward movement Y must be low, and for upward movement Y must be high. Obviously low figures are incremented and high ones decremented.

Now add the following lines:

```
 5  A=0:B=1:C=3
10  GRAPHIC2:COLOR,A,B,C,0
20  POINTC,X,Y
52  COLORA,B,C,0
53  A=A+1:B=B+1:C=C+1
```

Lines 52 and 53 are interchangeable depending upon whether or not you wish to do a complete color change at once.

It would perhaps be a good plan to increase the upper limit of the for next loop on line 18 so that the dot moves further. Note that there will be some color combinations that make the dot difficult to see. The program does not explore all the possible combinations. Perhaps you would like to expand it to include all possible changes, thus producing a program that would run for some considerable period of time!

Try this program next:

```
200  GRAPHIC2:COLOR0,0,3,0
210  A=0:B=0
220  CHARA,B,"HERE"
225  FORD=1TO20:NEXT
```

```
230  CHARA,B, "    "          There are four spaces there.
240  A=A+1:B=B+1
245  IFA=15ANDB=15THEN210
250  GOTO220
```

In order to see this program without going through the first one, assuming that you still have it in memory, just type RUN200. Again you can alter all the variables and make pretty patterns with the word HERE. The complete, two-part listing is included in Appendix N, program 33.

Note line 245. The word AND is what is known as a *logical operator*. The line, fairly obviously, checks to make sure that both X AND Y reach the value of 15 before doing anything. What happens if you change the value of just one of them? The result is an illegal quantity error. Both must be the same for the program to run properly. In order for you to be able to use two different values in that line, you must make use of another logical operator; OR. However in this instance there is not much point to it.

Change a few lines in this program and see what happens. First bring the cursor up to line 200 and type 205 instead. Line 200 has now been changed to 205. Then, first of all bringing the cursor down below the ready signal, type:

```
200  C=0
247  REGIONC
248  C=C+1
249  IFC=7 THENC=0
```

Now the word HERE flashes in different colors across the screen.

You will be well aware of the way Children's Television Workshop uses animation to point out the meanings of words. Here is an opportunity for you to write a small sequence for either your own or a neighborhood child, or even for your local elementary school, showing such notions as up, down, sideways, here, there, diagonal, left, right, inside, outside and so on and so forth. Such concepts as slow and fast are easy to do.

Add these lines for a chuckle.

```
201  DIMX(200)
202  Y=0
210  X=4+*SIN(Y)
220  CHARX(Y),Y,"HERE"
230  CHARX(Y),Y,"    "
240  Y=Y+.1
```

A neat replacement of the word HERE occurs. What if you change line 230 to read CHAR, Y, "   " ?, that is without the subscript (Y). The letter H is left behind after each of four printings of the word.

The following sequence demonstrates a number of points that have been used before and collects them together-

```
300  DIMA$(20)  (You can expand this later if you wish, but 20 will do.)
301  A$(1)="HERE"
302  A$(2)="THERE"
303  A$(3)="WHERE?"
304  A$(4)="THERE?"
305  A$(5)="HERE?"
306  A$(6)="UP"
307  A$(7)="DOWN"
308  A$(8)="SIDEWAYS"
309  A$(9)="LEFT"
310  A$(10)="RIGHT"
311  A$(11)="DIAGONAL"
312  A$(12)="SLOW"
313  A$(13)="FAST"
319  B$="        "
320  GRAPHIC2:COLOR0,0,3,0
330  X=INT(RND(1)*13)+1:Y=INT(RND(1)*13)+1
340  Z=INT(RND(1)*13)+1
350  CHARX,Y,A$(Z)
360  FORD=1TO450:NEXT
370  CHARX,Y,B$
380  GOTO330
```

This funny little program shows how different words can be placed at random all over the screen. Notice that there are occasions when the same word occurs twice in succession. There are also occasions when the word wraps around the screen with the last letter appearing on the left instead of the right. You can alter this program to reduce the number of spaces in the string B$. This leaves the odd character on the screen. Notice how the character is removed whenever a new word is printed over the top.

Try another:

```
10  GRAPHIC 2:COLOR0,0,2,0
20  A$="([C=]DUUUF)" (Press the Commodore logo key and the characters.)
21  B$=" o  o  "
22  R=1:C=1
23  FORI =1TO 18
25  FORX=1TO18
40  CHARR,C,A$
45  CHARR+1,C,B$
50  FORD=1TO50:NEXT
60  CHARR,C,"     "
62  CHARR+1,C,"     "
65  C=C+1
70  NEXTX
```

```
75  R=R+1
77  C=1
80  NEXTI
```

When you run this program, you will see some sort of wheeled vehicle passing across the screen. The operation of the program is fairly obvious. It is not a difficult matter to arrange for the truck to change color on each row.

See what you can do with circles:

```
10  GRAPHIC2:COLOR0,0,3,0
20  X=20:Y=20:RX=10:RY=10
25  FORI=1TO500
30  CIRCLE3,X,Y,RX,RY
40  CIRCLE0,X,Y,RX,RY
45  X=X+20:Y=Y+20
50  NEXT I
```

Watching the little creature worm his way down the screen can be maddeningly fascinating! Can we make him come towards us? Yes we can, simply by adding to line 45 the following two statements: RX=RX+2:RY=RY+2. Now change RY to increment by 4. Then change RX to increment by 6. Finally, to show that you have complete control over such creatures, make him crawl off into the distance starting low on the screen and working his way up. Combine the movement with suitable sound.

CREATING SPECIAL EFFECTS

The next section makes use of the VIC in unexpanded form, in fact expansion creates a bit of a problem that will be addressed at a later point in time. You are going to produce double size characters.

You will recall how you developed some characters of your own which are called user-defined characters, in an earlier chapter. You began by reserving an area of memory in which in put your new characters so that they would not be disturbed. You then prepared a series of data lines to be read one by one to produce the Chinese and Greek characters. You might not feel that such characters are any more useful than the ones given in the Appendix N dealing with Russian and Hebrew language alphabets.

The following program will allow your child or your store, using the VIC-20 to display advertisements, to use large letters, which can be seen at a distance.

Characters are normally constructed in a 8×8 dot matrix. As we have already seen, it is possible to create characters in an 8×16 matrix so that the characters are twice as tall. The address to poke here is 36867, and the expression is: POKE 36867,PEEK(36867) OR1. The results look a little odd, but if you examine it closely, you will see that the characters are printed in pairs one above the other. If you then type the letters of the alphabet in order, you will see how the pairs are grouped. You will find that A is missing. To get A, you must type the @ sign.

The program exists as a subroutine that you can call after you have put your material in the main program. I have used the simple name program as a basis, but you are, of course, free to make whatever use you wish of the routine.

```
    5 PRINT"♥"
   10 PRINT"THIS PROGRAMME WILL ALLOW YOU TO INSERT
      DOUBLE SIZE CHARACTERS AT WILL"
   20 PRINT"WHAT IS YOUR NAME?"
   30 INPUT A$
   40 GOSUB 10000
   45 PRINT"♥"
   50 PRINT"HELLO ";A$
   55 FORD=1TO5000:NEXT
 9999 END
10000 POKE56,28:CH=32776
10010 FORX=7184TO7600STEP2:POKEX,PEEK(CH):POKEX+1,
      PEEK(CH)
10020 CH=CH+1:NEXT
10030 POKE36879,25
10040 POKE36869,255:POKE36867,47
10050 RETURN
```

The program works by reading each part of a character twice into the RAM space you have prepared. Obviously you no longer have all 64 characters available to you, but then, it is unlikely that you will want such things as arrows and pound signs in your display. If you do then you must make the appropriate arrangements.

You might be more than just a little curious about the statement at the beginning of line 10030, which pokes a 25 into 36879. This has to do with border control. As has been noted, location 36879 has a lot to do with the color of screen border and characters.

Try this little sequence:

```
100  X=36879
105  FORI=1TO255
110  POKEX,I
120  FORD=1TO1000:NEXT
125  ?"ALL CHANGE"   (That's not quite true as you will see!)
130  NEXT
140  END
```

You can change the value of the duration loop on line 120 to suit your own purposes, and then incorporate the routine into a message program for a store window, or change certain portions of a teaching program or finance program to draw attention to certain features. There are two points to take into consideration however: never overdo a trick with color (or any other type of trick for that matter), and do make sure that the difference between colors allows for the material to be read with ease. It would be a good plan to make a note of the effect of the number after the comma. You will find that effects take place in groups. You can play quite happily with this by poking the address with values in the direct mode. If you do try this and get lost, just poke 36879,27 to get back to normal.

When you have had enough of this exploration, try inserting the information you have acquired into the double-size letter program to produce a grand effect.

Let us explore location 36879 just a little more.

```
  5 PRINT"■"
 10 C=36879
 15 PRINT:PRINT:
 20 PRINT"THIS PROGRAMME DEMON- STRATES THE EFFECT"
 22 PRINT"OF THE LOCATION ";C
 25 PRINT"CHOOSE THE EFFECT YOU WISH TO EXAMINE FROM
    THE FOLLOWING LIST"
 27 PRINT:PRINT:
 30 PRINT"SCREEN","1"
 32 PRINT"BORDER","2"
 34 PRINT"AUXILIARY";"3"
 35 INPUTA
 40 IFA=1THEN100
 42 IFA=2THEN200
 44 IFA=3THEN300
100 PRINT"■■ENTER A NUMBER FROM 0 TO 15■■"
110 INPUTX

120 The formula goes here. See the notes following for details.

130 PRINT"ENTER A NEW NUMBER TO SEE NEW EFFECT"
140 PRINT"ENTER 200 TO RETURN"
150 INPUTX
160 IFX<=15THEN120
170 GOTO5
```

Now, pay close attention because, although the action which follows might seem very much like a digression, it will be very useful in other programs and save a bit of time entering this one.

Bring the cursor up to the first digit in the line 100. Then press 2 and the return key. Line 100 now appears to be 200. Don't worry! You have *not* lost line 100. The cursor is now over the first digit in line 110. Press 2 again followed by the Return key. Continue doing this to the end of the 100's and then bring the cursor back to the top of the 100's again. This time follow the same procedure, but enter a 3 in each case. When you have done all this, list the program. You will find that you have all the line numbers. Now all you need to do is enter the appropriate formula and make one further change.

```
120  POKEC,PEEK(C)AND15OR(16*X)
220  POKEC,PEEK(C)AND2400RX
320  POKEC,PEEK(C)AND2480RX
```

You will probably have spotted the further change you need to make:

```
160  IF etc . . . . _ . 120
260  IF etc . . . . . .  220
360  IF etc . . . . . .  320
```

A complete listing can be seen in Appendix N, program 8.

Chapter 13

It is time to concern ourselves with those curious statements known as Input/Output Statements, usually shown in documentation as I/O statements. As you are aware, the computer talks to the keyboard and the screen quite automatically. In addition, the commands save and load talk to the Datasette quite automatically too. However, you might wish to do other things with our recorder than merely save whole programs.

Programs take up space. Not only that, but there are occasions when we wish to keep certain information separated from the program itself. An example might well be a program that makes use of special characters that you have designed. As you can imagine, the process of printing out a different program for each character set can be an arduous one indeed. A better way of dealing with special characters would be to produce what is known as a *utility* program that contains empty spaces for the specific data you wish to use.

First you would arrange for your utility program to accept what are known as *data files*. The data files themselves would be created by another program. All that happens is that the program that allows you to create the files would transmit the information you have gathered, but not the program itself, to the recorder. You would give the information a name. Then, loading your utility, you would place your data file tape in the recorder and allow the utility to read it. You would thereby create a third program. This could be used directly from memory or itself saved to tape.

The trouble with descriptions of a set of activities is that the process always seems to sound more complex and more troublesome than it really is. A suitable analogy, although analogies are seldom all that accurate, might be that of the loose-leaf binder into which all kinds of discrete chunks of information can be put and out of which they can be taken in whatever order you wish.

The computer can be made to talk to each of its own parts and other things that you attach to it under your control. In other words, you can begin to determine what it does and how it does it. This personal control is achieved through the I/O Statements.

INPUT/OUTPUT STATEMENTS

The commands in question are:

OPEN This statement opens a line or channel to a specific device that can either be an integral part of the VIC-20, like the screen and keyboard or a peripheral device, such as tape, disk, or printer. Whenever you wish to access a particular device, the open statement must be the first one you use.

CMD This statement tells the computer that the file that follows, whether it be data or a listing, will no longer be sent to the screen as usual but is being sent to some other device, like the tape-drive, disk, or the printer instead.

Print# This statement does the donkey work of sending the information to whatever device has been opened. It also serves as a kind of end-of-transmission signal, causing the screen to receive the transmission once more.

CLOSE This closes the channel to whatever device has been addressed. It is the last statement in any collection of statements that you might use for Input/Output.

GET# performs much the same function as the get statement, but with reference to whatever device you wish to acquire data from. It takes the data one byte at a time. If there is no data it works the same way as the get statement.

INPUT# is the device counterpart of the input statement. It gathers whole variables from tape or disk.

Each device has its own number. I/O statements must include this number so that the VIC-20 can keep track of where information is to go to or come from.

The device numbers are as follows:

Keyboard	0	(That's a zero)
Datasette	1	

RS232	2	(Used for communications via telephone or to talk to another computer next to the VIC-20)
Screen	3	
Printer	4	
Printer	5	(Yes you have a choice—more on that later)
Disk drive	8	

Let us say we wish to open a channel to the tape recorder. The format would be **OPEN1,1** where the first number is the file number and the number after the comma is the device number. You can give your files any number you like; this is your personal way of labeling groups of information. If you wished to send the same data to the printer, the statement would read **OPEN1,4** (or of course, 5). For the disk drive, the statement would be **OPEN1,8**. Quite clearly, you can have a variety of files, each with a specific number, and send them wherever you wish: this one to tape, this one to the printer. Of course, it is impossible to retrieve data from the printer.

You can add another number, which indicates a particular command, to the open statement. 0 (zero) added to **OPEN1,1** in the form **OPEN1,1,0**, followed by the filename, would instruct the computer to read data from the cassette. Using a 1 as the third number would tell the computer to write to the cassette. A 2 is the command not only to write to the cassette but also to put an end-of-tape (EOT) marker at the end of the data so that the machine does not keep on running with nothing happening. The zero when used with device 4, the printer, sends uppercase and graphics; and a 7 sends upper and lowercase.

In tabular form the whole instruction set for devices is as follows:

OPEN1,0	The keyboard is read.
OPEN1,1,0, ''name''	The cassette is read.
OPEN1,1,1,''name''	The cassette is written to.
OPEN1,1,2,''name''	The cassette is written to and an end-of-tape marker is placed at the end of the file.
OPEN1,2,0''string''	A channel to an RS232 (communications) device is opened.
OPEN1,3	The screen is written to/read.
OPEN1,4,0,''string''	Uppercase characters and graphics are sent to the printer.

OPEN1,4,7, "string"	Upper- and lowercase characters are sent to the printer.
OPEN1,5,0, "string"	Upper- and lowercase characters are sent to the printer, device# switched
OPEN1,8,15, "command"	A command is sent to the disk.

Bear in mind that the first number can be any in a range from 1 to 255 and refers to the file number. This is important when you wish to retrieve the correct material. It means that a series of program lines can collect data from a tape or disk and allow the computer to work on it (if it is, for example, numerical data that requires some computation) and can then send the result to the appropriate device. If you were doing your income tax, you would want to be able to figure your totals correctly before printing them. On the other hand you might wish to compare two sets of complex results to see which is the more advantageous.

One necessary precaution is to make sure you close a file when you have finished with it. Another precaution to take is to place a PRINT#(file number) *before* closing to make sure that everything you want sent to that device is actually sent. Otherwise there is a chance of losing material.

PRINT# is followed by the file number and then the variable or variables you wish sent, for example: PRINT#1, A$,B$C$, X$. The variables would have been declared previously and the contents typed in. Each variable could then be transmitted as and when necessary. Dimensioned variables will work in the same way. Don't confuse yourself by declaring either too many variables or too many dimensioned ones!

Let's look at the following short program that sends data to a printer:

```
10  OPEN1,4,0     (Uppercase is being sent)
20  CMD1:READA$,B$,C$
30  PRINT#1,A$,B$,C$,
40  CLOSE1
50  DATA"THIS IS THE FIRST ITEM"
60  DATA"THIS IS THE SECOND ITEM"
70  DATA"THIS IS THE THIRD ITEM"
```

In this example, the items will be spaced across the page with the third item extending to the next line after the word *the*. If you place semicolons after each variable in the PRINT# statement, the items follow each other immediately. To cause each item to be printed in a column, three separate lines, each with a PRINT# statement and one of the variables are necessary. There is another way of doing this that can be explained more efficiently in the section on the printer.

For the transmission of data to the tape, the first line would read OPEN1,1,1 followed by the filename. Retrieval of the data from tape would require the first line to read OPEN1,1,0 followed by the filename.

The following are a pair of sequences that will write and read files to and from tape:

```
10  OPEN1,1,1,"THIS IS A TAPE FILE"
20  NL$= CHR$(13)
30  PRINT#1,1;NL$;3;NL$;2;NL$;17
40  PRINT#1,22
50  PRINT#1,301
```

The variable NL$ is made to equal a carriage return, which separates each of the items of data.

```
100  OPEN1,1,0,"THIS IS A TAPE FILE"
110  PRINT"FILE NOW OPEN"
120  INPUT#1,A,B,C,D
130  INPUT#1,E
140  INPUT1,F: CLOSE1
```

USING A PRINTER

Let us now turn our attention to the printer. There are two printers that are used with the VIC-20: The VIC-1515, which will only work with the VIC-20, and the VIC-1525, which will also work with the Commodore 64. The model I am using is the latter.

The printer is actually made by Seikosha, and is also marketed by them in their GP series. The Seikosha version has both serial and parallel interfaces, which allows for use with computers other than those made by Commodore. The VIC version has only the serial interface; the little slot at the back in which the parallel interface connection would normally fit is empty. Yet even the serial interface is not the normal interface. Certainly it has the DIN plug, but the pin configuration differs from normal. You can get a custom cable made to do the proper conversion.

Setting up the printer is a simple affair: plug the cable with the DIN plugs into the printer and the computer. At the computer end, the plug sits in the remaining round plug hole (the other is already being used by the connection to the TV). Before plugging into the power, it will be necessary to do one or two housekeeping chores.

These are documented in the manual that comes with the printer. The entire manual deserves careful reading, but there are one or two things that you should be particularly careful with.

Fitting the ribbon can be a trifle awkward because it involves tilting the small plastic housings and then pressing them down on the supports. Make sure that the ribbon passes between the print head and the platten and also fits between the small feeder grip and the plastic swivel piece diametrically opposite the print head.

There is a small lever that sits in small depressions and is placed between the print head and the ribbon grip. The manual tells you (at least mine told me) that this controls the print contrast by increasing the print density when moved to the right, or clockwise, and decreasing the print density when turned to the left, or counterclockwise. Mine works the other way around.

The printer will take only specially perforated printer paper up to 9½ inches wide. The sides of this paper tear off so that you can remove the tractor holes and have standard 8¼ inch wide paper. In my set up, the box of paper sits on the floor and the paper is fed up behind the desk, down under the roller, around the platten, through the tractor mechanism, and then back down behind the desk once more. It is important to make sure that the paper is fed without any possibility of its getting caught on anything and that it moves in line with the tractor. The tractor sprockets should be properly adjusted so that they are neither too tight nor too close together. Once set they will stay where they are very nicely.

Before turning on the computer it is important to check the printer alone by running a test pattern. At the back of the printer there is a switch that can be set in one of three positions: T, 4, or 5. On the Seikosha machine they are T, S, and P (for serial and parallel). The VIC version allows you to select the printer to be either device 4 or 5. First set the switch to one of the two numbers, and then switch it on using the switch on the left rear part of the case. The printer, assuming that you have, of course plugged it into an ac outlet, will oblige you with a little dance of the print head to the middle of the paper and back to the left edge. Now you can move the rear switch to T, which is to your right; whereupon the printer will begin producing a complete compendium of its repertoire, and it will do it over and over again until you switch to either 4 or 5.

Here is a program that gets the printer to do a number of tricks. The first part of the program will be quite familiar; the various CHR$s will be explained after the listing.

```
10  OPEN1,4,7
15  NL$=CHR$(13)
16  L$ =CHR$(10)
17  FORI=1TO10:PRINT#1,L$:NEXT
20  CMD1:READA$,B$,C$
30  PRINT#1,A$,B$C$
32  PRINT#1,A$
33  PRINT#1,B$
34  PRINT#1,B$
35  PRINT#1,A$;NL$;B$;NL$;C$;NL$
36  PRINT#1,A$;NL$;L$;B$;L$;NL$;L$;C$;NL$
40  PRINT#1,CHR$(14) ,A$;B$;C$
50  PRINT#1,CHR$(18) ,CHR$(14) ,A$;B$;C$
99  CLOSE1
100  DATA"THIS IS THE FIRST ITEM"
110  DATA"THIS IS THE SECOND ITEM"
120  DATA"THIS IS THE THIRD ITEM"
```

A line by line examination of this program tells you that the printer is the device addressed and that you wish to send lowercase material. The printer will ordinarily print in uppercase and will revert to that state when the program has been completed and the file closed. NL$ is a carriage return; L$ is a line feed, which will put blank lines between text. Line 17 gives us ten line feeds. Line twenty is a common or garden variety read statement. The concomitant data lines are 100-120. Line 30 causes the three data lines to be printed one after the other without gaps between them. Lines 32 to 34 cause the same material to be printed, but, as with the screen, a new program line produces a new printed line. Line 35 does the same thing by using the NL$ variable. Line 36 does it all over again but introduces a line feed between each line. Line 40 causes some fancy footwork: CHR$(14) is a command to the printer to produce double width characters. CHR$(18) causes the material to be printed in reversed mode. Figure 13-1 shows how this looks from the printer.

You will note that the listing of the program is in double width mode. This is because I forgot to reset it by commanding the printer to use CHR$(15) before getting the program listing. Note that all other aspects have been reset: the print is uppercase and in normal (not reversed) mode. This figure also displays the results of some other codes together with the program that produced the output. Note lines 40 and 50. The tab setting is placed inside the quotation marks, and there is no space between the numbers and the first character.

There are other controls that would not, perhaps, be used very often. Any of the controls can be mixed, but do be very careful to

```
CHR$(14) PRODUCES DOUBLE WIDTH
CHR$(15) PRODUCES STANDARD WIDTH
          CHR$(16) SETS PRINT-START POSITION ON THE LINE
12IT CAN BE USED IN COMBINATION WITH CHR
$(14)
CHR$(145) SELECTS THE CURSOR UP MODE
THIS IS THE MODE USED WHEN THE PRINTER IS FIRST SWITCHED ON
chr$(17)selects the cursor down mode
[illegible]
AND CHR$(146) TURNS THE REVERSE FIELD OFF
10 OPEN1,4
20 PRINT#1,CHR$(14)"CHR$(14) PRODUCES DO
UBLE WIDTH"
30 PRINT#1,CHR$(15)"CHR$(15) PRODUCES ST
ANDARD WIDTH"
40 PRINT#1,CHR$(16)"10CHR$(16) SETS PRIN
T-START POSITION ON THE LINE"
50 PRINT#1,CHR$(14)CHR$(16)"12IT CAN BE
USED IN COMBINATION WITH CHR$(14)"
60 PRINT#1,CHR$(14)CHR$(145)"CHR$(145) S
ELECTS THE 'CURSOR-UP' MODE"
70 PRINT#1,"THIS IS THE MODE USED WHEN T
HE PRINTER IS FIRST SWITCHED ON"
80 PRINT#1,CHR$(17)"CHR$(17) SELECTS CUR
SOR DOWN MODE"
90 PRINT#1,CHR$(18)"CHR$(18) SELECTS THE
 REVERSE FIELD MODE"
100 PRINT#1,CHR$(146)"AND CHR$(146) TURN
S REVERSE FIELD OFF"
110 CMD1:LIST:CLOSE1
```

Fig. 13-1. The double character program and its printed results.

keep track of which controls you have used, which are still in use, and which are to be countermanded. Do believe me, the sweat you exude doing careful work before you run a program will repay you with a printout that will run sweetly the first time.

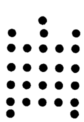
You will probably recall how you worked and struggled to get a neat bar chart to appear on the screen. As I commented then, there is really nothing like a bar chart to get managers tingling to the roots of their corporate hair and down their corporate spine making them to protrude their corpulence. It is not too difficult to produce impressive charts on the printer. Consider the following program, which is actually based upon that found in the printer manual and rewritten just to make things clearer for you.

```
10  OPEN1,4:FORI= 0TO5
20  READA:A$=A$+CHR$(A) :NEXT
30  FORD= 0TO5 :READB :B$=CHR$(B)
40  C$= CHR$(255)+CHR$(59)+CHR$(15)+CHR$(32)
50  D$= STR$(1+D)
60  PRINT#1,CHR$(15)D$A$B$C$b
70  NEXTD
80  DATA8,27,16,0,53,26
90  DATA63,22,45,67,94,102
```

This program produces a chart that numbers items and produces a block graphic and a number. The length of the graphic is proportional to the number that follows it creating a neat and tidy display.

The process of designing characters of your own to be printed out is similar to the process of designing them to be displayed on the screen. However, there is a rather important difference. Whereas the data lines that create characters to be displayed on the screen are read horizontally, those for the printer are read vertically. In addition there is one line fewer to deal with.

Let us say you want to design a small tank. (You can design what you want but I'm going to design a tank!) Take a piece of graph paper (I had none around but I did have some spare sheets from a stamp album, which served just as well) and draw lines around a seven by seven block of squares. Now design your object. Mine looks like this:



With a bit of imagination and the right game setting you would think it was a tank!

The left-most column is empty, and the value will therefore be zero. The next column is added vertically using the decimal equivalents of the binary code: the value of the top and in this case, empty, spot being 1, the second one down, 2, the third, 4, and so on. Empty spaces become 0 in decimal. The second column total value in decimal is therefore 126. Now add to the number 128 that figure, the value of the nonexistent eighth row. This results in 254. Calculate values for each of the columns, remembering to add 128 to the result. The data for our tank will read: 0,254,188,191,188,254,0

As these sets of figures are data, we place them in a data statement, which can go right at the beginning of the program if you wish. Here is the program:

```
10  DATA0,254,188,191,188,254,0
20  FORI=1TO6
30  READA
40  A$=A$+CHR$(A)
50  NEXT
60  OPEN1,4
70  FORI=1TO5
80  PRINT#1,CHR$(8)A$;
90  PRINT#1,CHR$(15)"THIS IS A TANK"
100  NEXT
```

I don't mind in the least if you leave out the line that tells what the design is. Instead, you might design a small logo that could be printed right across the top of your page as a personal headline.

To produce a row of tanks, all you need to do is put a for-next loop in place. To produce serried ranks of tanks, lined up in groups in staggered fashion, you need only introduce a series of nested loops, with a line feed control (CHR$(10)) in the appropriate spot.

Note that in all the program sequences I have printed here, I have used the same file number. This is because the sequences are of an expository and exploratory nature. In practice you would assign a different file number to each, thus keeping the whole affair from becoming confusing. You may label a file with any number from 1 to 255; however, you may not have more than 10 files open at any one time.

The printer manual is quite clear on the method of printing out a program listing, but I will include the instructions here in order to completely describe the fundamental printer operation.

You can list a program in the direct mode by typing OPEN 1,4:CMD1: LIST:PRINT#1:CLOSE1, all of which must be entered in one line with the return key being pressed only after the last character. Alternatively, you may include the line as a program line at the end of your program. The PRINT# statement ensures that everything in the buffer has been sent to the printer.

Quite clearly, the statements you have dealt with here have some bearing on the use of both tape and disk, particularly the parts dealing with handling files, opening devices, and dealing with the input and output of information. Thus, a program such as the one you have just explored would have extra commands if you wished to save it to disk.

The value of a printer cannot be overstressed. The problem of trying to find a bug by going through the program on the screen has already been mentioned. A complete listing of your program on a sheet of paper can often allow you to spot the bug immediately. Another advantage of the printer lies in its ability to produce printed documents in finished form. There is a wod for that: *word processing*. It deserves a separate chapter.

Before going on, consider the following, very familiar program:

```
  5 PRINTCHR$(147)
 10 DIMA$(20)
 20 INPUT"HOW MANY NAMES";N
 30 FORI=1TON
 40 INPUTA$(I):NEXT
 50 PRINTTAB(8)"NOW SORTING"
 60 FORI=1TON-1
 70 IFA$(I+1)>=A$(I)THEN120
 80 B$=A$(I+1)
 90 A$(I+1)=A$(I)
100 A$(I)=B$
110 GOTO60
120 NEXTI
130 FORI=OTON+1:PRINTA$(I):NEXTI
140 OPEN1,4
150 FORI=0TON+1:PRINT#1,A$(I):NEXTI
160 CLOSE1,4
170 RUN
```

There is a possibility that you might have a little trouble when using the printer and the tape or disk, particularly if you are attempting to save versions of a program, print versions of a program, and generally debug it.

The scenario is as follows: your program is taking shape, and you decide to save it to tape. You do so and all goes well. Then you try to list the program on the printer. The machine buzzes merrily away for a few lines and then stops. You examine the output and find that it is not only incomplete but also total garbage. Instead of printing 20 PRINT"THIS DEMON- STRATES THE EFFECT" you get something like this (and this is an actual example): 20 PRINT"THS POGRAME DMON- STRAES TH EFFECT" You try the run/stop and restore keys and do it again. You get a file-open error because you forgot to close it. You do that and start again. Once more you get garbage. Believe me, you can spend an hour fiddling with the machine and all to no avail! The solution, the quickest solution that is, is to switch off both the computer and the printer and then switch the computer back on. Rewind the tape and load your program once more. Now you will find that it will print perfectly—at least it will print exactly that which you have put in your program whether the program is perfect or not, and there will be no garbage!

If you wish to save the same program twice consecutively, merely bring the cursor up to the save instruction and press the return key. If you have kept the record and play buttons down, you will find that the program will save automatically. The screen will look a little messy but your program will be fine.

Chapter 14

This chapter on word processing is being written on the VIC-20, using a program produced and marketed by Commodore and called the VIC Typewriter. The program is part of the Home Calculation Program Pack VT-107-A.

Word Processing might seem like just another buzzword suitable only for large office complexes. It is, most certainly, a tool that you can use, if not in everyday circumstances, perhaps every other day. A word processing program allows you to use your VIC-20 not just like another typewriter, but rather like a machine that costs thousands of dollars.

Those of us who have graduated from the old clackety-clack manual typewriter to the wonders of the electric know the advantage of the speed of the latter over the former. Still there are the bugs. You know the sort of thing I mean: you have typed your page and are now looking it over; there on line three is a spelling error! Problem: do you try to align the page back in the machine, precisely as before, and attempt to erase the offending letters by using eraser tape and then insert the correct characters? Or do you retype the whole page with a fair chance of reproducing and of course compounding your errors? Or do you practice all those words that your children aren't supposed to know!?

USING A WORD PROCESSING PROGRAM

A word processing program can enable you to avoid all that!

Just a few moments ago, I left out one of the e's from the word *electric*. It was such a simple matter to move the cursor back to the word, create a space by using the familiar INST DEL key, and insert the missing character. Not one imprecation or expletive passed my lips!

"Ah", you will say, "You are used to using computers. It's easy for you!"

Used to computers I may be, but this is the first time I have ever used a word processing program. I bought the package, having written three quarters of the book you are now holding, just to provide you with some thoughts on the kind of software that is available. It is the first software that I have ever bought for any computer.

The process was quite simple (the process of using the program, that is; buying it caused the usual pain in the pocketbook), and consisted of placing the tape in the recorder and typing LOAD. I am using a VIC 8K expander in my machine at the moment and so was informed immediately that I had so far used 0 lines and that I had 417 lines as the total number allowable.

At this point in the chapter the screen tells me that I have used 119 VIC lines. I have just under 300 left.

The keyboard is used exactly like an ordinary typewriter: I press the shift key to type uppercase letters and special symbols like quotation marks and exclamation points. However, I must use the shift and returns keys to place a checkmark when I want a new paragraph.

Cursor left, right, up, and down are operated in the now familiar fashion, as is the delete key; I've used that a lot so far. There are times when my fingers tend to be large toes and so odd words have been erased in their entirety. I find that I can bring the cursor to the end of the offending word and tap the delete key for the total number of letters very rapidly without waiting for each letter to disappear. The delete function does its job without further attention. It is a little slow but very much quicker than the old method of inserting eraser tape or unscrewing the correction fluid bottle, and it doesn't smell!

Even the process of producing enhanced printing was easy: I just typed a CTRL2, which gave me a reversed e on the screen, and then I carried on typing. At the end of the line I typed CTRL3, which allows the print to revert to normal.

I have now used 174 VIC lines, so I am told. VIC lines are, of course, the lines that appear on the screen and are just a bit longer than a quarter of the printer line.

The function key plays a significant part by doing various housekeeping chores. f1 allows me to see the next screen of text. This is highly useful when reading through the material to check for errors, meaningless sentences, and unclear statements and to evaluate the general flow. f2 allows me to print the material directly on the VIC-1525 printer, which I am using for program listings in certain parts of this book. f3 allows me to insert a whole line, thus f1 and f3 can be used quite safely without having to use the shift key. I am not likely to accidentally print before I am ready. f4, which of course requires the shift key, deletes a line. Again, use of the shift means that one is not likely to erase a line by accident. f5 allows me to save my material to tape or disk. f6 allows me to load from tape or disk.

I find that I sometimes try to put in a return at the end of a VIC line or at the end of the screen. That just shows how conditioned one can become to ancient devices!

I have not yet tried reversed printing. I'll do it now by typing a CTRL9. The CTRL9 produces a reversed, lowercase r. To switch it off I type a CTRLO, and the CTRLO produces a reversed uppercase R. These cues remain on the screen at each point they are used, although the actual material they control looks quite normal.

I think I will save what I have so far, just in case there is a problem with the AC line in my house. The procedure for saving material is quite simple. First of all I placed an end-of-text marker at the end of the line ending in the word *house*. This was done by typing a CTRL6. In this way the recorder will switch off automatically instead of saving a lot of empty space. Next I went back through the text, by pressing f1, function key, until I reached the beginning of my text. I inserted a few blank lines and gave the material a name. I called it type. The format for naming material is: 1#:FILENM, where the number is 1 for tape and 8 for disk, and the program name is six characters or less. The computer supplies the extra spaces if you have less than six characters in the name. It took some time for the tape to stop after I had pressed function key f5.

It is a good plan to save material on tape or disk, if you have it in fairly short segments. Keep an eye on the VIC line number so that you do not run out of memory space. On an unexpanded VIC you are allowed 45 lines; a 3K memory expander gives you 185 lines; you have 417 lines with 8K, 790 lines with 16K, and 1207 lines with the

VIC expansion module. Just to give you an idea of how much can be put in, an unexpanded VIC takes you from the beginning of this chapter to the end of the fourth paragraph; 3K takes you to just before the part about enhanced characters, and the whole text up to just a little beyond the end of this sentence is 416 VIC lines of screen text.

My next task was to print my material. This is where the fun began. I did not do anything wrong, but I did discover that the printer will carry on merrily typing right over the perforated edge between sheets. I knew it would, of course, but I had to see how the printed material compared with what I really wanted and with the screen.

The line on the display screen is, as you already know, 22 characters wide. The material on the printer is presented in lines of 64 characters. (Note that although the printer can produce 80 characters per line, you must make alterations in the program to make it do so.) A conversion must take place so that you can leave reasonable margins at the top and bottom of each printed page. How much you leave is really up to you, but the procedure will be the same whatever you do.

Having made your decision as to where the breaks must come, insert extra blank lines at that point. Make sure that you introduce enough blank lines to span the perforation neatly.

If you wish to learn or practice your typing skills, the other side of the Typewriter tape contains a VIC Typing Tutor. Perhaps when I have finished this book I will have time to explore that and improve my own typing skills!

Chapter 15

This chapter contains a few oddities. You might find them useful; on the other hand you may only regard them as curious or even comical.

MORE TRICKS WITH THE SCREEN

Normally the screen has a nice even border around it. As you might expect, there is an address in the VIC-20 that controls this, but it doesn't do it without help. The address is 36865, and the normal value to be found at this address is 25. If you lower that number to 24, the screen will move up by two rows of dots. Normally you will only notice the difference because the border shifts, but anything else you put on the screen will shift as well. Try it in direct mode first, changing the number at that address by two each time. Then try increasing the number and watch the change.

The number of columns on the VIC-20 is normally 22. You might think that this is fixed, but it isn't. You can poke address 36866 to change the number of columns. First of all, type a large number of characters on the screen in the direct mode; these could be your name and address, for example. Press the return key to move the cursor. You will get an error message but don't worry about that: just leave it there. Now type POKE36866,PEEK (36866)AND128 OR 11 and then press the return key. Straight away the material is condensed into half the screen. Everything you typed is there, but it looks a little funny. If you wish to present only a set phrase without all the poke instructions, error messages, and what-not, type the following in the direct mode, of course.

```
?"[CLR HOME]": POKE36866,PEEK(36866)AND128 OR11:?"THIS IS THE
WAY TO DO IT"
```

and then press the return key. Only the phrase will appear. You might find it necessary to put a space between the first quotation mark and the first letter of your phrase in order to present the material without split words. Just as an example try the following short sequence:

```
5  ?"[CLR HOME]"
10  PRINT"WHAT IS YOUR NAME?"
20  INPUTA$
25  POKE36866,PEEK(36866)AND128 OR11
30  PRINT"HELLO ";A$
```

You might like to put a screen clear instruction at line 22 so that the answer alone appears on the screen. I must confess that I have occasionally walked into a store where the VIC-20 is sold, poked that address with some strange number, and then stood back to watch the salesperson panic. Poking a number higher than 22 causes a mess with a whole lot of colored garbage on the bottom of the screen. I must confess that I have not found a use for more than 22 column screens.

Using Two Screens

A program to demonstrate a further use of this feature follows: Making use of a principle similar to that followed when reserving space for user-defined characters or graphics, you can produce the effect of having more than one screen. The first screen can contain one set of information, which you input in the normal way. The second screen is accessed by using function key f1. When you press f1 the original material disappears and you can then enter new material. You can switch back and forth manually by pressing f1. Experiment with the idea of bringing the change under program control.

```
  5 PRINT"♥"
 10 POKE56,28:CLR    (This reserves two pages of memory, just as was
 20 DIMX%(23)            done with the user-defined graphics)
 25 PRINT"♥"
 30 GOSUB70:PRINTCHR$(147):GOSUB70
 40 X$=CHR$(133)    (This is function key f1. This will allow you to move
 50 GETY$:IFY$=X$THENGOSUB70        between two different screens)
 60 PRINTY$;:GOTO50
 70 A=PEEK(648)
 80 IFA=28THENA=30:B=150:GOTO110
 90 IFA=30THENA=28:B=22
110 POKE648,A:POKE36866,B
120 FORI=0TO23
130 X=PEEK(I+217):POKEI+217,X%(I)
```

```
140 XX%(I)=X
150 NEXTI
160 PRINT:RETURN
```

When you run this program, nothing seems to happen except that the screen clears. Type in something on the screen, fill it if you wish. Then press the f1 key. The screen goes blank again. Now type something different. Now when you press the f1 key, you will be able to switch between the two screens. This feature could be useful in lots of ways, particularly if you wished to compare two pages of figures or wished to present a variation on some information or other in a teaching program. In as much as the basic principles are the same as for producing your own characters, you could produce alternate sets of Russian and English or any other combination of material that needs to be easily compared.

If you place the identical material on each screen, but move it to a slightly different position (up a line, over one space etc.), you can animate the material. It is a little slow however. Change B in line 80 to 140. Now you have a full screen and a half screen. 135 gives you a third of a screen. 145, three-quarters of a screen. Changing B in line 90 to 20 also gives a smaller screen. Try lower numbers, higher numbers tend to produce garbage.

The CHAR Instruction

The Super Expander instruction CHAR allows you to put text on almost any spot you wish, in whatever color you wish. There is a problem, however, if you want to put a set of numerals on the screen. The CHAR instructions only deals with letters—or does it?

If you try to put in a numeric variable in the spot for text, in the form CHAR15,3 C, for example, you will get a type mismatch error. This is where the STR$ function becomes very useful. First, assign a variable name to the number. If it represented an amount of fuel, for example, you could use F. Then set up a transfer in the form F$=STR$(F). Follow this with the region statement, the color code you desire, and then the line CHAR ro,co,F$. The following program is an embryo attempt at doing this. You might wish to clean it up and include it somehow in the Jupiter Rendevous program.

```
10 GRAPHIC3:COLOR0,0,3,2
12 A$(1)="FUEL"
13 A$(2)="SPEED"
20 REGION11
30 CHAR15,3,A$(1)
40 CHAR18,3,A$(2)
50 REGION9
55 FORF=10TO1STEP-.5
56 FORC=1TO100
57 F$=STR$(F)
58 REGION12
59 CHAR15,12,F$
60 NEXTC
61 C$=STR$(C)
62 REGION5
63 C$=STR$(C)
64 CHAR18,12,C$
65 NEXTF
```

Appendix A
Sorts

The following are simple sort routines in BASIC.

BUBBLE SORT

```
100  DIMA(30)
110  FORI=1TO 30
112  A(I)=INT(RND(X)*60+1)
114  ?A(I)
116  NEXT:?
120  X=59
130  S=0
133  FORI=1TOX
135  IF A(I)<=A(I+1)THEN 160
140  AA=A(I)
143  A(I)=A(I+1)
145  A(I+1)=AA
150  S=1
155  X=I
160  NEXT
165  IFS=1THEN130
170  FORI=1TO60
175  ?A(I);
180  NEXT
190  END
```

You will need to change the line numbers in order for the routine to fit into your program. Break points are at 130 and 170. Do not forget to change the address of the GOTOs if you change the line numbers.

The same conditions apply to the following programs.

SHELL SORT

```
100  DIMA(30)
110  FOR I=1TO30
112  A(I)=INT(RND(X)*60+1)
114  ?A(I)
116  NEXT:?
120  B=1
130  B=2*B
133  IFB<=30THEN130
135  B=INT(B/2)
140  IFB=0THEN 200
143  FORI=1TO(30-B)
145  C=I
150  D=C+B
153  IFA(C)<=A(D) THEN 180
155  AA=A(C)
160  A(C)=A(D)
```

```
165  A(D)=AA
167  C=C-B
170  IFC>0THEN 150
180  NEXT :GOTO 135
185  FORI=1TO 30
190  ?A(I);
195  NEXT
200  END
```

SORT 1

```
100  DIMA(30)
105  N=30
110  FORI=1TO N
115  A(I)=INT(RND(X)*60+1)
120  ?A(I)
125  NEXT
130  M=A(1)
135  IM=1
140  FORI=2TON
145  IFA(I)>=MTHENM=A(I)
150  IM=I
155  NEXT
160  AA=A(N)
165  A(N)=A(IM)
170  A(IM)=AA
175  N=N-1
180  IFN<1 THEN 130
190  FORI=1 TO 30
195  ?A(I);
200  NEXT
205  END
```

SORT 2

```
100  DIMA(30)
105  N=30
110  S=1
115  FORI=1TO 30
120  A(I)=INT(RND(X)*60+1)
125  ?A(I)
130  NEXT
135  MN=A(S)
140  IM=S
145  MX=MN
150  IX=S
155  FORI=S TO N
160  IF A(I)>MX THEN MX=A(I)
165  IX<=I
170  IF A(I)>MN THEN MN=A(I)
175  IM=I
180  NEXT
185  IF IM=N THEN IM=IX
190  AA=A(N)
195  A(N)=A(IX)
200  A(IX)=AA
205  N=N-1
210  AA=A(S)
215  A(S)=A(IM)
220  A(IM)=AA
225  S=S+1
230  IFN>S THEN 135
235  FORI=1TO 30
240  ?A(I)
245  NEXT
250  END
```

Appendix B
Converting Programs to VIC BASIC

There will come a time when you will wish to type in a program from a magazine or book. Many times you will find programs that have been written expressly for the VIC-20 or perhaps for the PET series of computers. With these you have no problems at all.

More often than not, however, you will find that the program has been written for some other computer, and that is where some of your difficulties will begin. As I remarked earlier, BASIC varies, just as English does. In some instances, the BASIC command may look like VIC BASIC but behave in a slightly different fashion. The most important ones to look out for are the following:

DIM	This statement, if written for Sinclair machines, means I items of X characters each.
INKEY$	This statement does not exist on the VIC, but it behaves somewhat like the get statement does. It tests to see if a key has been pressed, but it can allow for more manipulation than the get statement does under certain circumstances.
LET	This statement is mandatory on many machines, but as you know, it can be omitted in VIC BASIC.
LPRINT	This statement is a composite command that addresses

the printer. Wherever you see this, you must translate it into the proper commands for the VIC printer.

USR	This statement allows you to access machine code routines that you have written almost as SYS does. These statements are really outside the scope of this book.
AT	This statement is sometimes written PRINT@. It allows you to designate the column at which material will start being displayed. Although the @ sign exists on the VIC, it cannot be used for this purpose. The best way to duplicate the action of print at statements is to use cursor controls or the SPC(X) function.
PAUSE	This statement is another composite command found on Sinclair machines. It behaves in the same way as a FORD=1TOX:NEXTD duration loop. In fact, many ZX81 or Timex/Sinclair programmers prefer to use the loop to avoid a flicker on the screen.
PEEK and POKE	If you see these used in a program for another machine, even a PET, then watch out. Unless you understand absolutely thoroughly the precise, exact address referred to in the utmost detail, you will send your VIC to a place where there aren't any phones!

Some commands you will see will refer to graphics. For example:

SCREEN	In the form SCREEN1,1	selects text/graphics and color set.
COLOR	In the form COLOR2,3	selects foreground and background color.
PCLS	In the form PCLS 8	clears a graphics page.
PSET	In the form PSET(120,90,4)	sets a point.
PPOINT	In the form PPOINT(120,90)	tests a point. (RDOT)

CIRCLE	In the form CIRCLE(120,90),30	is the same as CBM Super Expander CIRCLE.
PAINT	In the form	is the same as VIC PAINT.
DRAW	In the form	is not the same as the VIC DRAW; it does more.
LINE	In the form	is nearly the same as DRAW; will draw lines.

HTAB and VTAB are statements peculiar to the Apple and refer to horizontal and vertical tabbing.

PLOT is a statement that is often found and must either be dealt using the pokes discussed quite early in this book or using a Super Expander.

UNPLOT is a statement that erases areas where material had previously been plotted. CHR(32) serves as well as anything in most circumstances.

Appendix C
Programming Problems

There will be times when you will have difficulty entering material. Somehow or other, whatever you try to do the VIC-20 will seem to foul up. This situation occurs most often when you wish to alter a line that you are currently entering. The right cursor won't work except to put extra characters on the screen; the delete key will not work; in general you want to tear your hair out in frustration!

Just keep calm; press the return key, even though the line is not complete; and either bring the cursor back up to the line and move it over the correct material, altering only the incorrect material, or without bringing up the cursor, type the entire line again. In correcting a line in a program you have already written, it may be less frustrating in the long run to type the line over again without attempting to correct just one portion of it.

The VIC is very good about telling you when things are wrong, but you must work out for yourself exactly what is causing the error message to appear. Most of the error messages are self-explanatory, and yet, you can gaze at a line in which there is supposed to be a syntax error without seeing it at all. It could be a comma instead of a colon, a misplaced or missing parenthesis or even an extra parenthesis, an expression in inappropriate form, or quotation marks in the wrong place, omitted, or too many!

Extreme caution must be exercised when assigning variable names. Although it is possible to use an entire word as a variable in order to make your program clear, it is better to keep variables to no more than two characters, either a letter and a numeral or two letters. A curious situation can occur otherwise.

Let us say that you wish to deal with a number of shapes. You have decided to label from FORM1, FORM2, FORM3, etc. It is all perfectly logical to you, but the computer will interpret the labels in an entirely unexpected fashion. The VIC is likely to throw the error message FOR WITHOUT NEXT at you! You see, it has taken the word FORM1 to be FOR M1 and so looks for a NEXT to correspond.

The word FOR falls into the class of *reserved words* that BASIC uses as its basis of operation. Even if a reserved word is embedded in a long word, unlikely but nevertheless possible, the computer will read it as a BASIC word, or try to, and either give you an error message or do something strange.

Curiously enough, the computer is able, via the clever work done by the designers of the BASIC language, to distinguish between the OR in OR and the OR in FOR. If, however, you label a variable BORD for BORDER the VIC will pick out the OR and try to read it as a reserved word. Just be careful when assigning variable names!

Appendix D PET Programs and Other Commercial Products

Many companies besides Commodore are producing all sorts of goodies that enhance the operation and therefore the enjoyment of computing with the VIC-20. Most of the items are in the form of what is known as *firmware,* a combination of hardware and software that plugs into the VIC in the form of a cartridge that contains additional circuitry and fundamental programming to allow extra facilities. One example is the Super Expander. VICMON is a cartridge that allows you to program in machine code, a procedure which would otherwise be quite bothersome.

Transfer between computers of the Commodore line is both easy and not so easy. A taped program in BASIC which has been developed in a PET/CBM machine will load onto a VIC-20 if, and only if, the program is small enough to fit into the memory you have available. This can vary, of course, for there are expansion memory cartridges which will allow you to add RAM in units of 3, 8, and 16K. An expansion chassis is one means of adding two or three cartridges at a time and accessing them either all at once or in individual units as required. As the total available RAM can thus be in excess of 32K, programs written on the PET 4016, a popular machine in schools, can be used for practice at home by those students who have the VIC-20 available to them. (This assumes that Father and Mother don't hog the machine!). The VIC-20 will load such a program at the start of available memory and carry out the link address corrections with no need of user intervention.

With machine-language programs, the problems are manifold. It is likely that such a program will not load in the first place. If it does it will more than likely thwart efforts to run.

Transfer in the other direction, from VIC-20 to PET/CBM is not a great problem if you remember to do a couple of pokes before you begin. A VIC without a memory expansion stores BASIC programs starting at 4097. In the PET, however, the normal starting address is 1025. If you load a VIC program into the PET it will be stored at 4097 unless you perform the following pokes: POKE 41,16 and POKE4096,0. After that has been done, the program should both load and run successfully.

If the VIC-20 that has been used to develop a program has the Super Exander or any other memory expander in place, you should be able to load and run the program without need of these pokes as the programs begin at the same address.

It might seem as if problems are created when you don't know whether or not the VIC program has been generated using an expansion module. In such a situation you must first of all place the tape in the Datasette and then type an OPEN1 command. Then press play on the recorder. Eventually the screen will show READY even though the program has not been loaded. Next type in the following on the keyboard: PRINT PEEK(829)+(256*PEEK(830)). This will tell you the starting address of the program. If the number 1025 comes up you will know that expansion cartridges of one kind or another have been used.

What about length of program? It is all very well to know the starting address, but if the program is too long to fit into the memory you have available, it is useless. Simply type: PRINT PEEK (831)+(256*PEEK(830)). The result will be the number of types used. Do not confuse this with the result of PRINT FRE(0), which tells you how many bytes there are left to work with. If you in fact enter PRINTFRE(0) after performing the above operations, you will find that you have all the memory in the world for none has been yet used.

It can happen, believe it or not, that someone tries to load a tape that contains not a program, but a performance of a favorite instrumental group. In days when computers have made use of existing technology surrounding the cassette tape deck, it is likely that some confusion will arise from time to time. For those who have tried listening to a program, it will not be news to tell them that the result is a nasty buzz. Unfortunately you cannot hear what is going on, or rather coming off, a tape merely by turning up the sound on the TV. The way to find out on the VIC is to perform a **PRINT PEEK(828)** If the value that appears on the screen is a 1 then you have a program. Otherwise the tape is either Beethoven's first movement for not paying his rent or, and this is important, a data file.

Appendix E
Error Messages

BAD DATA This occurs when string data is received as opposed to the numeric data that was expected; for example, Harry instead of 1234

BAD SUBSCRIPT This occurs when there is an attempt to access an item which is outside the range indicated in a DIM statement. On occasion the error is not very obvious. Let us say that you have set DIMA$(20) followed by a for-next loop which allows you to indicate the number of entries: FORI=1TOX: INPUT A$(I). If X is set, using an INPUTX, to higher than 20, you will get a bad subscript error.

CAN'T CONTINUE Typing CONT will not work under certain circumstances; for example, when you have listed line numbers from X to Y, the computer will not respond to a command to CONT and continue listing. On other occasions, the failure occurs because the program was never run or there has been an error.

DEVICE NOT PRESENT You might be able to see the printer sitting there, but the computer will not know it is there unless the printer is switched on. The same is true of the Datasette, disk, modem, or other device.

DIVISION BY ZERO Mathematicians will tell you that this is impossible, even with a computer!

EXTRA IGNORED It is possible to use multiple INPUT statements, however, should you try to enter more than is expected, this error message will appear.

FILE NOT FOUND This is probably the most frustrating message you can get. It can often appear if you mislabel the file in the load instruction. Keep an exact account of the name of a file. Also keep an exact note of where it is to be found on the tape.
FILE NOT OPEN The golden rule is to open your file first.
FILE OPEN This means that you are attempting to use the same file number over again without having closed the file after previous use.
FORMULA TOO COMPLEX I am tempted to remark here that your VIC-20 is only human! The solution here is to break the formula into manageable chunks. There are two benefits from doing this: the computer will be able to deal with the problem, and you will be able to understand your program more easily after a long period of nonuse.
ILLEGAL DIRECT This will occur if you attempt to use the INPUT command in direct mode. This can't be done!
LOAD This appears in the form LOAD ERROR when, for some reason, there is a problem with the program on tape. Always make back-up copies of your programs. The problem could be one of many, but think first that the program or part of it has been overwritten with another program or part of one.
NEXT WITHOUT FOR. This one seems obvious, but can cause quite a headache when you are trying to sort through a pile of nested FOR-NEXT loops. A good case can be made for keeping FOR-NEXT loops in noncompacted form at least until you are sure your program runs well.
NOT INPUT FILE This means that the file you tried to access to enter material was designated (by you, so don't blame the computer) as an output file only.
NOT OUTPUT FILE This is the exact opposite of the previous error message.
OUT OF DATA Let us say that you have a FOR-NEXT loop which controls a read statement. The loop says `FORI=0TO10`, yet there are only 10 items in the data statement. You can puzzle over this until you realize that 0 to 10 is actually 11 items. This error message is the result. You should change your FOR-NEXT loop.
OUT OF MEMORY This one can be painful! The first thing to do is remove all unnecessary spaces from your program. See if you see the same thing over and over again; for example, label an address such as 36879 with a variable and then replace all occurrences of the number with the variable. See if you have a large number of GOSUBS. The statement ON-GOTO is tremendously useful be-

cause it obviates the necessity of a series of IF-THEN statements. For example, the program that allows you to play with the screen and border colors could have a line 40 ON A GOTO 100,200,300 instead of three if-then lines. Try it and see. Finally you can use the trick of *overlaying* programs: making one program load a new program. See Appendix J.

OVERFLOW There is a limit to the size of the numbers that the VIC-20 can handle. Unless you are attempting to compute the number of atoms in the universe or the amount of money, in pennies, that OPEC has earned in the last five years, you are unlikely to see this error.

REDIM'D ARRAY You can dimension an array only once in a program. Watch where you put the DIM statement. Sometimes a return statement can redirect the program so that this message appears. It can cause a lot of headaches.

REDO FROM START This is the equivalent of the BAD DATA error but in direct user form. Whereas the BAD DATA ERROR concerns a file from a device, the REDO FROM START concerns the user at the keyboard. Don't type letters when numbers are asked for.

RETURN WITHOUT GOSUB This can happen inadvertently when you are trying out a program. It is a good plan to place an end statement on the line before the first line of a subroutine. Otherwise the program will go toddling through, meet the return statement, go back to the pick-up point, carry on through to the subroutine again, meet the return statement again, and give you the RETURN WITHOUT GOSUB message.

STRING TOO LONG The old question of how long a piece of string is finally has an answer. It can be no longer than 255 characters. This has to do with the fact that the largest number that can be represented by an eight digit binary number is 255. Using the information in the section on user-defined characters, prove it to yourself: 11111111 = 1+2+4+8+16+32+64+128=255

SYNTAX This was invented by grammarians before the IRS started taxing certain establishments. It has to do with things like commas, colons, semicolons, and parenthesis. If one of these is missing, in the wrong place, or used when it shouldn't be, up pops this message. It also pops up when you type something in the direct mode on occasion. In writing a program you can sometimes be so carried away with enthusiasm that you forget to put in the next line number. That'll do it !

TYPE MISMATCH This one is the opposite of REDO FROM START. Don't enter numbers when letters are asked for.
UNDEF'D FUNCTION This means that you have attempted to make use of a user-defined function, but you did not define that function using the DEF FN statement.
UNDEF'D STATEMENT This occurs most often when you have reorganized a program by deleting a line, yet still have a GOTO or GOSUB that references that line number. Watch out for this one for the meaning is not obvious at first. Some other computers will not perceive a problem and will go merrily on to the line number that follows the deleted one. That can sound like a good idea, but it can cause even greater problems!
VERIFY This message appears in an unadorned fashion and can look like something you entered yourself. What it means is that the program you have on tape does not match the one in memory. You'll get it after you have given the VERIFY "N" command if things are not as they should be.

Some of these messages will occur seemingly quite out of the blue, particularly if the VIC-20 is not the first computer you have worked with. The reason may well be that the machine you are used to will accept certain forms that the VIC-20 will not. This does not mean that the VIC-20 is in any way deficient. It is just a matter of the familiar being the most comfortable.

Just as there are expressions in the English language as spoken in North America that are either totally meaningless or even rude in Britain (and vice versa), so there are small differences in the dialects of BASIC that are understood by different machines.

There is an excellent book by David Lien called *The Basic Handbook* which takes you, statement by statement through the BASIC language, provides a neat test to see if your computer operates in the common or uncommon fashion, and gives you alternatives if your computer does not have that command.

One curious feature of the VIC-20 is the provision of the@sign, giving it large prominence among the operators. I assumed at first that it acted like any other@sign, that is in the form 10?@5"WHAT", which would cause the word WHAT to be placed starting on column 5. As far as I can tell the@sign does nothing special on the VIC-20, unless it serves a special purpose for some of the cartridges.

Appendix F
The Data Base Program

The file program, which is listed in full in Appendix N, program 36, is not exactly a full-fledged data base program. It is a file program. It does give you an idea, however, of what a data base program can do. This program is almost an exact translation of a program that I wrote for another small computer, and some features reflect that fact.

The amount of material that you can store in the program is directly related to the amount of memory you have. The program is designed to allow easy entry of material, easy alteration of the file, and equally easy printing of the file contents.

After each stage of activity, the program returns you to the menu. There is no such thing as a fully satisfactory file program. Each person will have different requirements. This is why businesses will pay large sums of money to programmers (only the good ones, of course!) to write programs that are specifically addressed to the client's needs. I do not expect that you will find this program to be exactly what you are looking for; however, with the knowledge and skills you should have acquired working through this book, you should have little difficulty in altering the program to suit your needs. It might take you some time, but it will more than likely be worth it.

You might wish to add a new section to perform certain types of calculations. To do this, add your new sequence at the end and alter the ON GOTO statement on line 210, adding a new line number. You may well be dissatisfied with the slow sorting procedure.

Perhaps you might like to substitute one of the other sorting routines that appear in Appendix A.

The figures relating to number and size of entries refer to an 8K memory. For larger memories, the figures can be increased. You might wish to alter the program to allow data to be kept on the tape or disk so that it can be accessed by the program rather than to be obliged to save the entire program each time.

If you have saved a file that includes data on tape by calling it an appropriate name in the approved fashion, be sure that you do not type RUN when you wish to view the material or otherwise work on it. You will lose your data!

Recall that the run command resets all variables to 0. You will still have the material safe in the taped version, but you will not have it in memory. Instead, type GOTO 20, or GOTO10. This will ensure that you will be able to see and work on the material saved on a previous occasion. The program warns you about this sort of thing each time the INITIALIZE FILE choice is selected.

Appendix G
The Talk Program

The talk program, which is listed in full in Appendix N, program 48, seeks to demonstrate a variety of things, not least of which is the enormous amount of fun you can have producing a look of total astonishment on the face of any friend—or even an enemy! The program must not under any circumstances be taken seriously! There are only the bare bones, with just a few suggestions as to how to finish the program.

The user is asked to provide a name and told if it is too long. The user is then engaged in conversation with the computer. You will note that almost the first thing the computer meets during the program is the GOSUB 8000. This merely allows the VIC-20 to learn the contents of the dimensioned string arrays, which are used thereafter according to the needs of the programmer. In addition, the responses given by the user are placed in storage so that they can be used, probably against the user (!) on later occasions. There are some gaps left in the dimensioned arrays so that you put your own material in.

A sample run of the program might begin as follows:

```
OK JOHN WHAT IS YOUR PROBLEM?
BROTHER
WHAT'S THE MATTER WITH YOUR BROTHER?
SICK
OH. IS IT SERIOUS?
YES
TOO BAD
```

Your own imagination can work wonders on such a program.

My original version of this program was written for a different computer and is fairly lengthy. There are moments when the machine seems to stop and think. I even have one spot where all sorts of odd things appear on the screen while the thinking takes place. The program is full of conversational fillers such as "It never rains but it pours;" "Things must get worse before they get better;" "Have you ever been in such a situation before;" "Now let me get this straight... You say that your (father, brother, sister, girlfriend) is (sick, pregnant, insane, cheating,). Have I got that right?"

With astute use of loops, the program can go around and around making use of little bits of the program over and over again. Be careful to keep track of exactly where the loops go, or else the user might find him or herself trapped within the program—come to think of it, that may not be such a bad notion after all.

In any case, the program is there for you to do with what you will. Should you become so bold as to produce a full version, you are quite free to publish it as long as you provide acknowledgment of the source of the notion.

Appendix H
The Amazement Program

The program Amazement is a trifle odd. I have included it because it is odd and because it deserves some work to make it into something. You will recognize a number of tricks in it, particularly those referring to screen and user defined graphics. It makes use of the joystick. The little object to be steered around the maze is supposed to be a tank. It points in certain directions but not in others. Can you find out why? The maze is not all there (and you have probably suspected that about the programmer!) and is supposed to appear as you move the tank around. Sometimes it does not appear, in which case you can steer the tank where you wish, even off the bottom of the screen! The idea is to move the tank up to the top left corner, whereupon you can start all over again. You could try changing the data lines that hold the images of the tank so that you get boats or planes. You can move the joystick diagonally at corners and so clip a little off your time. You could add a little sequence after line 180 to produce an explosion or a randomly flashing screen. You can play with the color of the text as it appears and change the color of the maze, the background, and the tank. Show yourself what you have learned from this book!

```
10  PRINT"♥";:GOSUB260:GOSUB380
20  P=253
30  POKESX+P,4:POKECM+P,2
40  DIMDIR(3)
50  DIR(0)=22:DIR(1)=23:DIR(2)=21:DIR(3)=1
60  T=TI
```

```
 70  FORI=0TO3
 80  POKECM+P+DIR(I),4
 90  POKECM+P-DIR(I),4
100  NEXTI
110  POKE37154,127:X=(NOTPEEK(37151))AND60-((PEEK(37152)
     AND128)=0):POKE37154,255
112  IFX=0THEN110
115  TP=P-22*((XAND8)>0)+22*((XAND4)>0)-((XAND1)>0)+
     ((XAND16)>0)
120  CH=-(3*((XAND16)>0)+4*((XAND1)>0)+5*((XAND1)>0)+
     ((XAND8)>0))
130  IFCH<30RCH>6THENCH=5
140  IF(XAND32)THEN250
150  IFPEEKSX+TP)<>32THEN170
160  POKESX+P,32:POKESX+TP,CH:POKECM+TP,2:P=TP
170  IFP<>23THEN70
180  PRINTCHR$(147):POKECC,8
185  FORD=1TO200:NEXTD
188  POKECC,27
190  PRINT"[illegible]HOORAY[illegible]!":POKE36869,240
200  SEC=INT((TI-T)/60)
210  PRINT"[illegible]IN";SEC;"SECONDS"
220  PRINT:PRINT"[illegible]PRESS SPACE BAR TO",,"RETURN[illegible] "
240  GETA$:IFA$=""THEN240
250  RUN
260  CH=7168:POKE51,240:POKE52,CH/256-1:POKE55,240:
     POKE56,CH/256-1
265  POKE51,240:POKE52,CH/256-1:POKE55,240:POKE56,CH/256-1
270  FORI=0TO7:POKECH+256+I,0:NEXT
280  READA:IFA=-1THENRETURN
290  FORJ=0TO7:READB:POKE7168+A*8+J,B:NEXTJ
300  GOTO280
310  DATA3,126,102,60,236,60,102,126,0
320  DATA4,16,214,254,170,186,254,198,0
330  DATA5,252,204,120,110,120,204,252,0
340  DATA6,0,198,254,170,170,254,214,16
345  DATA 7,255,255,255,255,255,255,255,255
350  DATA-1
380  POKE36869,255
385  PRINT"[illegible]AMAZEMENT[illegible]";
390  SX=7680:CM=38400:CC=36879
400  DIMA(3):A(0)=2:A(1)=-44:A(2)=-2:A(3)=44
410  A=SX+23:WL=7:HL=32
420  FORI=1TO21:PRINT"   [illegible]GGGGGGGGGGGGGGGGG":
     NEXT:POKEA,4
430  J=INT(RND(1)*4):X=J:POKESX+505,J+128:POKECM+505,8*RND(0)
440  B=A+A(J)
450  IFPEEK(B)=WLTHENPOKEB,J+1:POKE+A(J)/2,HL:A=B:GOTO430
460  J=-(J+1)*(J<3):IFJ<>XTHEN440
470  J=PEEK(A):POKEA,HL:IFJ<5THENA=A-A(J-1):GOTO430
475  PRINT"[illegible]";:POKESX+505,32
480  RETURN
```

Appendix I
Using Memory Expanders

You know how it is: you have lived in a house for a few years, and you know where everything is and can lay your hand on any item at once. Then you move to a larger house. Suddenly, all those things seem to have been misplaced. A similar thing happens when you expand the memory of the VIC.

You already know where most of the important locations are, things like screen and color memory. You have been using them quite a lot. Just in case you are uncertain, let us produce a formula to find out where screen memory begins. On the screen of your VIC type the following (in direct mode):

```
SC=4*(PEEK(36866)AND128)+64*(PEEK(36869)AND112)
```

Then press the return key. Type ?SC and the value 7680 will appear.

You could put this into a short program if you wish.

```
5?"[CLR HOME]"
10  SC=4*(PEEK(36866)AND128)+64*(PEEK(36869)AND112)
20  ?SC
```

Next add the line that will tell you the start of color memory:

```
30  C=37888+4*(PEEK(36866)AND128)
40  ?C
```

The value should be 38400. The complete program is listed in Appendix N, program 9.

If you have an 8K expander, get ready to put it in the computer, if it is not already in, that is. Save the short program under the name MEM, and then switch off the VIC-20. Insert the 8K module; switch the computer on again; and load MEM. When all is ready, type RUN, and you will see that the values for screen and color have changed to 4096 and 37888 respectively.

This is vitally important information, for without it, you will not know how to alter your programs so that they will run when the VIC-20 is expanded.

For example, the short program that does things with the ball will not play. Neither will those programs that deal with bar charts. In fact, no program that thinks the memory starts at 7680 will run with the expander in place. Note that the locations will remain the same with the Super Expander in place as with the unexpanded VIC-20.

In order to make programs run with the expander, first change the screen and color addresses to the new values. If you have assigned those values to variables, you will find things a lot easier than going through an entire program searching out each value and possibly getting into such a muddle that you get cross with the whole thing.

Appendix J
Overlaying

There will be occasions when you will wish you had more memory. It does not matter how much you have; there will come a time when more is necessary, and yet, there is no real need to go rushing out to buy another 8K and a mother board to put all the cartridges in. There is a cheap but by no means nasty method.

The method is called overlaying. Let us assume that you have a program that can be divided into two or more parts. It could be a finance package, for example, that requires certain input that could be stored on a separate tape. The second part could access the data and work on it. Thus, load part I; remove the tape from the Datasette; insert a fresh tape; and work through part I. As the data is loaded on the tape, the program continues to operate. As soon as all data is entered an EOT marker is put in place, and you are told to remove the data tape and insert the original program tape without winding forward or backward from where you ended the original load. The program directs you to press the play button and the second program is loaded automatically by a load instruction on the last line of the original program. It will probably read something like 10000 LOAD "PART II". When the loading is complete, you will be told to insert the data tape that you have just created and you will then press the play button upon request.

To create a data tape, use the program 10 in Appendix N as a model. To read the data you have created from the tape program 11 in Appendix N might serve.

Appendix K
Producing Music

Figure K-1 shows the approximate numbers that must be poked to produce the corresponding musical notes. Note that the voice at location 36876 is the highest, the one at 36875 sings all its notes an octave lower, and the one at 36874, an octave below that. All three voices cover a range of three octaves as in the diagram, thus a five-octave range is available in all.

The above values for notes are approximations. The result will sound a little odd for the *intonation,* the state of being in tune, will be off to a small degree. It is possible to make use of the technique of *frequency modulation,* which involves rapid switching back and forth between two close frequencies, in much the same way that a violinist produces a vibrato.

A program that demonstrates the use of this technique appears in the Programmer's Reference Guide. However, as a musician I find some of the values assigned to notes to be slightly off key and the DIMN is incorrect producing an out-of-data error. The corrected program is found in Appendix N, program 32, although you are to feel free to disagree with my ear for intonation.

A long discussion of intonation would be out of place in this text. Suffice it to say that over the centuries there have been many attempts to produce a true intonation on keyboard instruments that is satisfactory in all keys. There were many systems developed, particularly for the harpsichord and now for the piano. These attempts resulted in a hybrid system, the result of which is that the

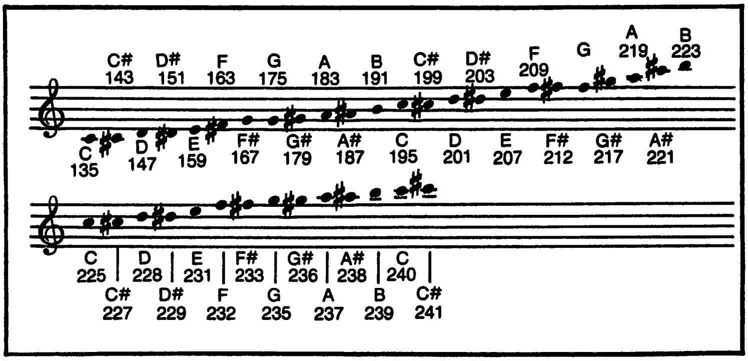


Fig. K-1. Poke Numbers to be poked to produce musical notes.

piano is not actually in tune. The problem revolves around a curiosity known as the *comma of Didymus,* which you can learn more about by consulting a good reference on the subject of piano and harpsichord tuning.

Do not assume that there is something wrong with your VIC-20 merely because the machine does not play in tune. You are free to experiment.

You will note from the program that there are two values for each note, which are sometimes the same and sometimes different. Alteration of the difference between pairs and sometimes the generation of differences where they do not currently exist in the data lines are the means of developing a true scale.

Discovery of the correct values to produce a well-tempered system can proceed by using just two voices. Make one voice produce the note C while the other produces the note G. Don't choose a pitch that is too low or you will not hear the effect you are looking for.

As the two notes sound together, you will know if they are exactly in tune by the clean, bare, nonoscillating tone that is produced. Once you have achieved this by juggling the number used to produce the note G, you can begin to work to introduce a small degree of oscillation. You see, if there were no oscillation between the bulk of the notes, you would end up with some notes wildly out of tune. It is a small degree of "out-of-tune-ness" that you want.

Alter the number for the note G until you can hear a very slow beat, as the oscillation is called. The oscillation is to be produced by the note G being either flat or sharp. You need the flat version, which is why you started with the perfect nonoscillating interval to begin with.

Having tuned G, you can now take the note D below it. Again produce a clean, bare sound and then sharpen the pitch of the D, by increasing the value of the number, until it too produces a very slow beat. Continue through all the notes working with the musical intervals of a fifth and fourth alternately. The fifths should be small and the fourths large. You should finish on the note B, having gone through all the sharps. Check the final note with an E or an F#. The result should be in tune. If it is not, I'm afraid you will have to go back through the whole thing to adjust the error, and if you think that is a chore, just imagine that I must go through this every week with two harpsichords and a clavichord—a total of 305 notes!

Once you have done it, you have a permanent system. Write a program that allows you to select each note in order. Then you can rent your services out as an instrument tuner. There are such things as electronic tuners on the market. They cost as much as your VIC did! And they won't do your tax, sort names, play games, or draw pretty pictures for you. In addition, by no stretch of the imagination can one play tunes on them.

For the owner of the Super Expander, the process of music writing is much easier. However there is no control over the precise pitch of the notes produced when using music mode. The sound instruction, on the other hand, makes use of the values as set out above, but without the need for read data statements. The voices are poked for you and are merely set out in the order s1, s2, s3, and s4 followed by the value, ranging between 0 and 15, for the volume.

The music mode is accessed by holding down the CTRL key and the left arrow, followed by entering the tempo, volume, voice, octave and the note. The arrow can be followed by the letter P, which allows the screen to display the sounds. The code Q turns the screen display off.

Appendix L
Experimenting with Graphics

The following program allows you to generate those pretty patterns that fascinate! The Super Expander is mandatory.

```
10 GRAPHIC2:COLOR0,0,1,5
15 READK
20 IFK<0THEN150
30 FORX=0TO1000
40 IFX*K>200THEN100
50 POINT5,X,X*K
60 NEXTX
100 GOTO15
150 GOTO150
160 DATA.01,.1,.2,.5,.8,1,5,10,30,-1
```

How would you make the lines come lower down on the screen?

```
10  GRAPHIC2:COLOR0,0,1,5
20  READK
30  IFK<0 THEN150
40  IFK<=1THENGOSUB60
45  IFK>1THENGOSUB110
50  GOTO20
60  FORX=0TO500
70  IFX*K>200 THEN 100
80  POINT5,X,X*K
90  NEXT
100  RETURN
110  FORY=0TO200
120  POINT5,Y/K,Y
130  NEXT
```

```
140  RETURN
150  GOTO 150
160  DATA.004,.2,.8,1,2,5,70,-1
```

While this next one is cooking you can go dig the garden, do your laundry, and watch a TV show!

```
10 R=50
12 A=1:C=0
15 GRAPHIC3:COLOR0,0,3,7
20 FORX=-RTORSTEP.1
30 Y=SQR(R*R-X*X)
40 POINT3,500+7*X,500+3*Y
50 POINT3,500+7*X,500-3*Y
60 NEXT
65 C=C+1
70 R=R-3
75 IFC=20THEN90
80 GOTO15
90 END
```

Try variants on the variables and see if you make the thing vertical.

The next program doesn't look very promising at first, but give it a chance. Try changing the character color: it looks fabulous in red or purple! Black on purple or black on red are frightfully with it. Program 40 in Appendix N is another variation of this program.

```
10  GRAPHICS3:COLOR0,0,3,0
20  FORA=0TO 60 STEP.05
30  R=A
40  POINT3,500+6*R*COS(A),500+2*R*SIN(A)
50  NEXT A
```

You could also change the range of A up to no more than 75, if you keep the formulae on line 40 as they are. You can increase A if you reduce the figure after the 500; for example, 500+5 *R*COS(A),500+2+R*SIN(A). You can also increase the step size up to .2 for a very charming effect. The greater the difference between the two "second" figures in the formula, the greater the apparent 3-D effect.

Using the same basic formula and program, and changing the two "second" figures back to 7 and 4, change R in the following ways: R=20*COS(3*A/2)......, reduce A to 50, and increase the 20 to 50. How far can it be increased without getting the illegal quantity message?

Next change R= 50*COS(SIN(10*A)). Changing the first number on the right of this equation alters the size of the figure. What happens if you change the figure in parenthesis to, say, 15? What if you change A's limit to 100?

Now change R again: R=50*COS(SIN(4*A/3)). The number in parenthesis can be increased from 4 to 8 with a positive effect. If you wish to enlarge the figure, you must increase the 50 to, say, 60. As in previous programs, the aspect ratio, the height and width, can be altered in line 40.

Now let R=80*SIN(3*A), or let R=EXP (A/6), but make sure you don't shoot off the screen! How can you prevent that from happening?

Now let R=30*COS(2*A/3). Make both "second numbers" =8 in the formulae in line 40. Next make the second formula read: ,500+12*R*SIN(A). Now you have the answer to the question about inverting images!

Now try FORA=0TO80, and R=30*COS(2.5*A/3).

Now, here is a challenge task for you. Develop a program that allows the user to see the effect of each formula upon demand. You will use the same program. However, for each selection, the program will branch to a spot that contains the precise expression for R. Work the program backwards after that, by showing the pattern and then displaying the formula alone or even displaying it beside the pattern. You can safely put such information on row 1. You can split long expressions between the top and bottom rows.

Appendix M
A Memory Map

In much the same way that people organize their houses with specific places for specific activities, so the circuitry in a computer is organized into specific compartments. A description of these compartments and their functions is called a memory map. The memory map that follows is fairly elementary as one of the many decisions made when drafting this book was to omit any discussion of machine language.

Some of the memory in the VIC-20 is reserved by the machine for its own purposes. The remainder is made available to you, the user, for you to do with what you wish—within limits, of course! Some of it is only available if you add extra memory. The VIC-20 knows when it is there!

To start at the top: the area 65535 down to 57344 is reserved for the VIC Kernal Operating System. This organizes all the other activities of the computer from the moment you switch on the machine.

57343 down to 49152 contains the VIC BASIC interpreter. This, quite clearly, interprets the BASIC commands, operators, functions, and statements that you enter on the keyboard.

The area from 49151 down to 40960 is reserved for expansion Read Only Memory. Cartridges that you buy to plug in the back of the machine contain expansion ROM, unless, of course, you have plugged in expansion RAM.

The area from 40959 down to 38912 is unused. The area from

38911 to 38400 contains the character color control table; that is, the color of any letters, digits, or diagrams that you place on the screen, but not the screen color, border color or auxiliary color. When you expand the VIC-20 the color control table moves to the area from 38399 down to 37888.

The input and output chip registers are found from 37167 down to 37136.

The VIC chip, which is actually tautologous for the C stands for chip, the full name being Video Interface Chip, has its registers at 36879 down to 36846.

Character representations are kept at locations 36863 down to 32768. There is a 16K gap in the memory from 32768 down to 8185 for expansion.

The area from 8185 down to 7680 should be familiar as the screen memory. You will recall that we poked characters in these locations on the screen and then used the color control table to make the characters appear.

The memory available to you in the unexpanded VIC runs from 7679 to 4096, followed by a gap from 4095 to 1024 for the 3K memory expansion module.

All that remains is the area used by BASIC itself and the operating system, and that is from 1023 down to 0.

You will recall that within some of these areas there were other subdivisions; for example, the four voices and the volume control are kept at 36874,36875,36876,36877, and 36878.

Appendix N
The Programs

This appendix contains listings of most of the programs in the book. They are listed here in complete, accurate form for your convenience in typing them into your VIC-20.

Program 1: Double Size Characters

```
    5 PRINT"♥"
   10 PRINT"THIS PROGRAMME WILL ALLOW YOU TO INSERT
      DOUBLE SIZE CHARACTERS AT WILL"
   20 PRINT"WHAT IS YOUR NAME?"
   30 INPUT A$
   40 GOSUB 10000
   45 PRINT"♥"
   50 PRINT"HELLO ";A$
   55 FORD=1TO5000:NEXT
 9999 END
10000 POKE56,28:CH=32776
10010 FORX=7184TO7600STEP2:POKEX,PEEK(CH):POKEX+1,
      PEEK(CH)
10020 CH=CH+1:NEXT
10030 POKE36879,25
10040 POKE36869,255:POKE36867,47
10050 RETURN
```

Program 2: Sample Printer Commands

```
10 OPEN1,4
20 PRINT#1,CHR$(14)"CHR$(14) PRODUCES DOUBLE WIDTH"
30 PRINT#1,CHR$(15)"CHR$(15) PRODUCES STANDARD WIDTH"
40 PRINT#1,CHR$(16)"10CHR$(16) SETS PRINT-START
   POSITION ON THE LINE"
```

```
 50 PRINT#1,CHR$(14)CHR$(14)"12IT CAN BE USED IN
    COMBINATION WITH CHR$(14)"
 60 PRINT#1,CHR$(15)CHR$(145)"CHR$(145) SELECTS
    THE CURSOR UP MODE"
 70 PRINT#1,"THIS IS THE MODE USED WHEN THE PRINTER
    IS FIRST SWITCHED ON"
 80 PRINT#1,CHR$(17)"CHR$(17)SELECTS THE CURSOR
    DOWN MODE"
 90 PRINT#1,CHR$(18)"CHR$(18) SELECTS THE REVERSED
    FIELD MODE"
100 PRINT#1,CHR$(146)"AND CHR$(146) TURNS THE REVERSE
    FIELD OFF"
110 CMD1:LIST: CLOSE1
```

Program 3: Screen

This program demonstrates some special effects on the screen and illustrates the creation of windows.

```
  5 PRINT"♥"
 10 A$="THIS IS THE PLACE"
 14 PRINTA$
 15 FORD=1TO1500:NEXT
 20 POKE36865,17
 22 PRINTA$
 25 FORD=1TO1500:NEXT
 30 POKE36865,25
 32 PRINTA$
 35 FORD=1TO1500:NEXT
 45 PRINT"♥"
 50 POKE36866,PEEK(36866)AND128OR28
 60 PRINTA$
 70 FORD=1TO5000:NEXT
 80 POKE36866,PEEK(36866)AND128OR22
100 A=36867:POKEA,PEEK(A)AND129OR(10*2)
105 PRINT"♥"
110 FORD=1TO4000:NEXT
120 POKEA,PEEK(A)AND129OR(5*2)
130 FORD=1TO4000:NEXT
140 POKEA,PEEK(A)AND129OR(15*2)
150 FORD=1TO4000:NEXT
160 POKEA,PEEK(A)AND129OR(23*2)
170 PRINT"♥"
200 POKEA,PEEK(A)OR1
210 FORD=1TO4000:NEXT
220 POKEA,PEEK(A)AND254
```

Program 4: Two Screens

This program allows you to create two different screens that can be switched back and forth by pressing the f1 key.

```
 5 PRINT"♥"
10 POKE56,28:CLR
20 DIMX%(23)
25 PRINT"♥"
30 GOSUB70:PRINTCHR$(147):GOSUB70
40 X$=CHR$(133)
```

```
 50 GETY$:IFY$=X$THENGOSUB70
 60 PRINTY$;:GOTO50
 70 A=PEEK(648)
 80 IFA=28THENA=30:B=150:GOTO110
 90 IFA=30THENA=28:B=22
110 POKE648,A:POKE36866,B
120 FORI=0TO23
130 X=PEEK(I+217):POKEI+217,X%(I)
140 X%(I)=X
150 NEXTI
160 PRINT:RETURN
```

Program 5: Road

```
 1 PRINTCHR$(147)
 2 REM "RANDOM ROAD"
 3 PRINT"THIS GENERATES A ROAD-"
 4 PRINT"WAY WHICH TRAVELS UP"
 5 PRINT"THE SCREEN"
 6 PRINT"ALTER LINE 20 TO      "
 7 PRINT"CHANGE THE WIDTH    "
 8 FORD=1TO6000:NEXT
 9 PRINTCHR$(147)
10 FORI=1TO100STEP.5
15 X=INT(RND(1)*3)+1
18 IF X=2 OR X=21 THEN X=5
20 PRINT TAB(X) CHR$(41);CHR$(125)
21 PRINT TAB(X+12)CHR$(125);CHR$(40)
25 FORD= 1TO 50:NEXTD
30 NEXTI
```

Program 6: Poke 646

This program demonstrates the results of using POKE 646.

```
 5 PRINT"♥"
 6 POKEC,6
10 PRINT"THIS IS WHAT POKE646 DOES"
20 C=646
30 FORI=0TO15:POKEC,I
35 PRINTI
40 FORD=1TO500:NEXT
50 NEXT
60 POKEC,6
```

Program 7: Animate

This program produces a pattern of squares, a moving ball, and moving letters.

```
  5 PRINT"♥"
  6 POKEC,6
 10 C=36879
 15 POKEC,8
 20 FORI=7680TO8185STEP3:POKEI,224
 30 NEXT
 60 POKEC,6
150 POKEC,8
160 FORI=7680TO8185STEP2
```

```
170 POKEI,81:FORD=1TO30:NEXT
180 POKEI,32:NEXT
200 FORI=7680TO8185STEP23:POKEI,81:FORD=1TO30:NEXT
210 POKEI,32:NEXT
220 FORI=8185TO7680STEP-21:POKEI,81:FORD=1TO25:NEXT
230 POKEI,32:NEXT
240 FORA=1TO255
250 FORI=7680TO8185STEP6:POKEI,A:FORD=1TO55:NEXT
260 POKEI,32:NEXTI:NEXTA
```

Program 8: Colors

This program allows you to view a variety of screen, border, and auxiliary colors.

```
  5 PRINT"■"
 10 C=36879
 15 PRINT:PRINT:
 20 PRINT"THIS PROGRAMME DEMON- STRATES THE EFFECT"
 22 PRINT"OF THE LOCATION ";C
 25 PRINT"CHOOSE THE EFFECT YOU WISH TO EXAMINE FROM
    THE FOLLOWING LIST"
 27 PRINT:PRINT:
 30 PRINT"SCREEN","1"
 32 PRINT"BORDER","2"
 34 PRINT"AUXILIARY";"3"
 35 INPUTA
 40 IFA=1THEN100
 42 IFA=2THEN200
 44 IFA=3THEN300
100 PRINT"■■ENTER A NUMBER FROM 0 TO 15■■"
110 INPUTX
120 POKE(C),PEEK(C)AND15OR(16*X)
130 PRINT"ENTER A NEW NUMBER TO SEE NEW EFFECT"
140 PRINT"ENTER 200 TO RETURN"
150 INPUTX
160 IFX<=15THEN120
170 GOTO5
200 PRINT"■■ENTER A NUMBER FROM 0 TO 15■■"
210 INPUTX
220 POKEC,PEEK(C)AND240ORX
230 PRINT"ENTER A NEW NUMBER TO SEE NEW EFFECT"
240 PRINT"ENTER 200 TO RETURN"
250 INPUTX
260 IFX<=15THEN220
270 GOTO5
300 PRINT"■■ENTER A NUMBER FROM 0 TO 15■■"
310 INPUTX
320 POKEC,PEEK(C)AND248ORX
330 PRINT"ENTER A NEW NUMBER TO SEE NEW EFFECT"
340 PRINT"ENTER 200 TO RETURN"
350 INPUTX
360 IFX<=15THEN320
370 GOTO5
```

Program 9: Memory Check

This program checks to determine whether or not a memory expansion unit has been added.

```
 5 PRINT"♥"
 8 PRINT"MEMORY EXPANSION CHECK"
10 SC=4*(PEEK(36866)AND128)+64*(PEEK(36869)AND112)
20 PRINTSC
30 C=37888+4*(PEEK(36866)AND128)
40 PRINTC
```

Program 10: Data to Tape

This program allows you to save user-defined character sets on tape.

```
10 OPEN1,1,2"DATA
20 READA:PRINT#1,A:IFA=0THEN40
30 FORA=ATOA+7:READB:PRINT#1,B:NEXT
40 CLOSE1:PRINT"DATA ON TAPE"
50 END
```

Program 11: Data from Tape

This program allows you to use load user-defined character sets from tape.

```
1000 OPEN1,1,0"TAPE"
1010 INPUT#1,A:IFA=0THEN1040
1020 FORA=ATOA+7:INPUT#1,B
1030 POKEI,B:NEXT
1040 GOTO1010
```

Program 12: Homed

This home medicine program demonstrates the use of computer aided diagnosis. It is an example of how useful the VIC-20 can be in providing fast information. The program is far from complete but, in its present form, can fit quite nicely into the three and a half K of the unexpanded VIC-20.

Obviously, further expansion of the program should be done with care. The program is not intended to replace a doctor, but it will give an idea of the probable nature of a few health problems. As you can see, the symptoms are kept in a series of arrays. These are called up when appropriate. The items entered by the user are checked against the roster of symptoms, and the program reacts accordingly.

For your own experimentation, see if you can do something with the way the material is presented on the screen. Then, using what you have learned as a basis, create programs that deal with other matters such as historical facts, tax regulations, or anything else you like.

```
 5 PRINT"♥"
10 CLR:PRINT:PRINTTAB(2);"SELECT EACH SYMPTOM
   ONE AT A TIME"
 15 FORD=1TO2000:NEXT
 20 PRINT:PRINT" IF THE PATIENT DOES   NOT SHOW THE
    SYMPTOM  FOLLOWED BY THE '?'"
 25 PRINT:PRINT" TYPE 'NO'"
 30 FORD=1TO3000:NEXT
 40 DIMZ$(100)
 50 GOSUB9000
100 PRINTA$(1),,A$(2)
110 INPUTZ$(1)
115 PRINT"♥"
120 PRINTZ$(1)
125 PRINT:
130 IFZ$(1)=A$(1)THENPRINTA$(2);"?"
135 IFZ$(1)=A$(2)THENPRINTA$(1);"?"
140 INPUTZ$(2)
150 IFZ$(2)=A$(2)THENPRINTZ$(2)
155 IFZ$(2)=A$(1)THENPRINTA$(1)
160 IFZ$(2)=A$(100)THENX$(1)=Z$(1)
165 IFZ$(2)=A$(99)THENPRINT"♥"
180 PRINT"  ANY OTHER SYMPTOM?    IF SO PLEASE SELECT
    FROM THE FOLLOWING:"
183 PRINT"[illegible]";A$(3),,A$(6),,A$(5)
184 PRINT:PRINT"OR TYPE 'NO'"
185 INPUTZ$(3)
186 PRINT"♥"
188 IFZ$(3)=A$(6)THEN230
190 IFZ$(3)=A$(100)THEN8000
191 IFZ$(3)=A$(3)THEN8050
192 IFZ$(3)=A$(5)THENPRINT" 1.RASH  OR",,"2.BLISTERS?"
193 PRINT:PRINT"SELECT NUMBER"
194 INPUTD
195 IFD=1THEN1000:IFD=2THEN2000
200 INPUTZ$(4)
203 PRINT"♥"
205 IFZ$(4)=A$(99)THENPRINTZ$(4):PRINTA$(1)+A$(6)
220 PRINTA$(98)
225 INPUTZ$(5)
230 IFZ$(5)=A$(100)THEN8999
235 PRINTA$(8);"?"
240 INPUTZ$(6)
245 IFZ$(6)=A$(99)THEN8010
1000 PRINT:PRINT"1.ON BACK AND NECK"
1002 PRINT:PRINT"  OR"
1004 PRINT:PRINT"2.ON TRUNK  AND AB-   DOMEN?"
1006 PRINT:PRINT"  SELECT NUMBER"
1010 INPUTC
1012 PRINT"♥"
1013 IFC=1THEN1500:IFC=2THEN3000
1500 PRINTX$(1),X$(7)
1502 PRINT"THE RASH SIGNALS THE  END OF THE INFECTION"
1503 PRINT"IN 3 DAYS IT SHOULD   BE CLEARED UP"
1505 PRINTSPC(2)"BE SURE TO WARN ANY"
1506 PRINTSPC(2)"FRIENDS WHOSE CHILD-  REN MIGHT HAVE
     COME   INTO CONTACT WITH"
1507 PRINTSPC(2)"THE DISEASE--IT CAN   BE DANGEROUS
     IN THE   CASE OF PREGNANCY"
1508 PRINTA$(97)
1509 FORD=1TO9000:NEXT
```

```
1510 GOTO10
2000 PRINT"J"
2001 PRINTX$(1),X$(8)
2002 PRINT"QQQQQQQQQQQQQ";A$(97)
2005 FORD=1TO7000:NEXT
2009 GOTO10
3000 PRINTX$(1),X$(6)
3005 FORD=1TO7000:NEXT
7999 GOTO10
8000 PRINTX$(1),,X$(2)
8005 FORD=1TO7000:NEXT
8008 GOTO5
8010 PRINTX$(1),X$(5)
8012 PRINT"QQQQQQQQQQQQQ";A$(97)
8013 FORD=1TO7000:NEXT
8015 GOTO5
8050 PRINT"J"
8055 PRINT"IS THERE ANY CHEST    PAIN UPON DEEP
     BREATH-ING?"
8060 INPUTZ$(11)
8065 IFZ$(11)=A$(100)THEN8090
8070 IFZ$(11)=A$(99)THENPRINTX$(1),X$(4),A$(97)
8074 FORD=1TO8000:NEXT
8075 GOTO5
8090 PRINTX$(1),X$(3)
8091 PRINT"QQQQQQQQQQQQQ";A$(97)
8098 FORD=1TO8000:NEXT
8099 GOTO5
9000 DIMA$(100)
9002 DIMX$(100)
9005 A$(1)="FEVER"
9006 A$(2)="COUGH"
9007 A$(3)="SPUTUM"
9008 A$(4)="SWOLLEN GLANDS"
9009 A$(5)="SPOTS"
9010 A$(6)="FREQUENCY"
9011 A$(7)="CHEST PAIN"
9012 A$(8)="BURNING"
9013 A$(9)="BLISTERS"
9014 A$(10)="TINGLING IN SKIN"
9015 A$(11)="SURFACE SKIN PAIN"
9016 A$(12)="BLACKENED SPOTS"
9017 A$(13)="RASH"
9505 X$(1)="IT IS LIKELY TO BE"
9512 X$(2)="COMMON COLD"
9513 X$(3)="BRONCHITIS"
9514 X$(4)="PNEUMONIA"
9515 X$(5)="URINARY,KIDNEY OR     BLADDER INFECTION"
9516 X$(6)="CHICKENPOX"
9517 X$(7)="GERMAN MEASLES"
9518 X$(8)="HERPES ZOSTER"
9987 A$(97)="▓▓CONSULT YOUR PHYSICIAN▓▓"
9988 A$(98)="ANY OTHER SYMPTOMS?"
9989 A$(99)="YES"
9990 A$(100)="NO"
9991 RETURN
```

Program 13: Hebrew

This program enables you to display the Hebrew character set.

```
  1 POKE52,28:POKE56,28:CLR
  3 FORI=7168TO7696:POKEI,PEEK(I+25600):NEXT
  5 POKE36869,255
  8 REM"HEBREW"
 10 FORA1=7176TO7183:READA:POKEA1,A:NEXT
 20 DATA65,99,52,24,44,70,99,0
 30 FORB1=7184TO7191:READB:POKEB1,B:NEXT
 40 DATA127,63,2,10,2,127,127,0
 50 FORV1=7344TO7351:READV:POKEV1,V:NEXT
 60 DATA127,63,1,1,1,63,127,0
 70 FORG1=7224TO7231:READG:POKEG1,G:NEXT
 80 DATA7,3,1,1,3,7,13,0
 90 FORD1=7200TO7207:READD:POKED1,D:NEXT
100 DATA127,127,2,2,2,2,2,0
110 FORH1=7232TO7239:READH:POKEH1,H:NEXT
120 DATA127,127,1,33,33,33,33,0
130 FORZ1=7376TO7383:READZ:POKEZ1,Z:NEXT
140 DATA6,3,2,2,2,2,2,0
150 FORW1=7352TO7359:READW:POKEW1,W:NEXT
160 DATA6,3,1,1,1,1,1,0
170 FORX1=7360TO7367:READX:POKEX1,X:NEXT
180 DATA127,127,33,33,33,33,33,0
190 FORT1=7328TO7335:READT:POKET1,T:NEXT
200 DATA102,43,83,65,65,67,126,0
210 FORY1=7368TO7375:READY:POKEY1,Y:NEXT
220 DATA12,12,4,8,0,0,0,0
230 FORK1=7256TO7263:READK:POKEK1,K:NEXT
240 DATA127,127,3,19,3,126,126,0
250 FORL1=7264TO7271:READL:POKEL1,L:NEXT
260 DATA96,32,32,62,2,12,8,0
270 FORM1=7272TO7279:READM:POKEM1,M:NEXT
280 DATA108,46,81,65,65,95,94,0
290 FORN1=7280TO7287:READN:POKEN1,N:NEXT
300 DATA14,6,2,2,2,14,15,0
310 FORS1=7320TO7327:READS:POKES1,S:NEXT
320 DATA126,63,33,97,97,98,124,0
330 FORE1=7208TO7215:READE:POKEE1,E:NEXT
340 DATA50,59,17,17,14,28,56,0
350 FORP1=7296TO7303:READP:POKEP1,P:NEXT
360 DATA126,63,17,61,33,63,126,0
370 FORF1=7216TO7223:READF:POKEF1,F:NEXT
380 DATA126,62,49,49,33,63,126,0
390 FORC1=7192TO7199:READC:POKEC1,C:NEXT
400 DATA99,35,20,8,4,63,127,0
410 FORQ1=7304TO7311:READQ:POKEQ1,Q:NEXT
420 DATA96,62,1,33,34,36,40,32
430 FORR1=7312TO7319:READR:POKER1,R:NEXT
440 DATA127,127,1,1,1,1,1,0
450 FORU1=7336TO7343:READU:POKEU1,U:NEXT
460 DATA1,82,123,41,73,82,124,0
470 FORO1=7288TO7295:READO:POKEO1,O:NEXT
480 DATA64,82,123,41,73,82,124,0
490 FORI1=7240TO7247:READI:POKEI1,I:NEXT
500 DATA64,126,33,73,65,33,97,0
510 FORJ1=7248TO7255:READJ:POKEJ1,J:NEXT
520 DATA64,126,33,65,65,33,97,0
```

Program 14: Russian

This program enables you to display the Russian character set.

```
  1 POKE52,28:POKE56,28:CLR
  3 FORI=7168TO7696:POKEI,PEEK(I+25600):NEXT
  5 POKE36869,255
  8 REM".AUSSIAN"
 10 FORA1=7176TO7183:READA:POKEA1,A:NEXT
 20 DATA0,8,20,20,34,68,65,65
 30 FORB1=7184TO7191:READB:POKEB1,B:NEXT
 40 DATA0,124,66,64,124,66,66,126
 50 FORV1=7344TO7351:READV:POKEV1,V:NEXT
 60 DATA0,65,42,28,28,28,42,120
 70 FORG1=7224TO7231:READG:POKEG1,G:NEXT
 80 DATA0,60,34,32,32,32,32,124
 90 FORD1=7200TO7207:READD:POKED1,D:NEXT
100 DATA0,62,34,34,34,34,95,67
110 FORH1=7232TO7239:READH:POKEH1,H:NEXT
120 DATA0,97,50,28,8,28,38,67
130 FORZ1=7376TO7383:READZ:POKEZ1,Z:NEXT
140 DATA0,56,68,4,56,4,68,56
150 FORW1=7352TO7359:READW:POKEW1,W:NEXT
160 DATA0,124,66,66,124,66,66,124
170 FORX1=7360TO7367:READX:POKEX1,X:NEXT
180 DATA0,64,78,81,113,81,81,78
190 FORT1=7328TO7335:READT:POKET1,T:NEXT
200 DATA0,127,73,8,8,8,8,8
210 FORY1=7368TO7375:READY:POKEY1,Y:NEXT
220 DATA0,0,0,68,68,100,84,100
230 FORK1=7256TO7263:READK:POKEK1,K:NEXT
240 DATA0,68,72,112,72,68,66,67
250 FORL1=7264TO7271:READL:POKEL1,L:NEXT
260 DATA0,127,83,19,19,19,83,39
270 FORM1=7272TO7279:READM:POKEM1,M:NEXT
280 DATA0,65,99,85,73,65,65,65
290 FORN1=7280TO7287:READN:POKEN1,N:NEXT
300 DATA0,102,102,102,126,102,102,102
310 FORS1=7320TO7327:READS:POKES1,S:NEXT
320 DATA0,28,34,96,96,96,34,28
330 FORE1=7208TO7215:READE:POKEE1,E:NEXT
340 DATA0,124,64,68,124,68,64,124
350 FORP1=7296TO7303:READP:POKEP1,P:NEXT
360 DATA0,126,102,102,102,102,102,102
370 FORF1=7216TO7223:READF:POKEF1,F:NEXT
380 DATA0,8,62,73,73,73,62,8
390 FORC1=7192TO7199:READC:POKEC1,C:NEXT
400 DATA0,68,68,68,68,68,68,59
410 FORQ1=7304TO7311:READQ:POKEQ1,Q:NEXT
420 DATA0,107,107,107,107,107,127,1
430 FORR1=7312TO7319:READR:POKER1,R:NEXT
440 DATA0,120,68,68,120,64,64,64
450 FORU1=7336TO7343:READU:POKEU1,U:NEXT
460 DATA0,34,34,18,10,6,34,72
470 FORO1=7288TO7295:READO:POKEO1,O:NEXT
480 DATA0,24,36,66,66,66,36,24
490 FORI1=7240TO7247:READI:POKEI1,I:NEXT
500 DATA0,65,67,69,73,81,97,65
510 FORJ1=7248TO7255:READJ:POKEJ1,J:NEXT
520 DATA0,85,73,67,69,73,81,96
600 PRINT"J"
610 PRINT"ETO KARANDAQ? NET,ETO NE KARANDAQ"
```

Program 15: Bike

This program produces the sounds of a motorbike starting and going away.

```
  5 REM"BIKE"
  6 A=36874:B=36875:C=36876:D=36877:E=36878
 10 POKEE,15
 20 FORI=128TO165
 30 POKEA,I:POKEB,I:POKEC,I
 40 FORX=1TO100:NEXTX:NEXTI
 50 POKEA,0:POKEB,0:POKEC,0
 60 FORI=128TO140
 70 POKED,I
 80 FORX=1TO20:NEXTX:NEXTI
 90 POKED,0:POKEE,8
100 FORI=140TO195
110 POKEA,I:POKEB,I:POKEC,I
120 FORX=1TO75
130 NEXTX:NEXTI
140 POKEA,0:POKEB,0:POKEC,0
150 POKEE,5
160 FORI=175TO200
170 POKEA,I:POKEB,I:POKEC,I
180 FORX=1TO200:NEXTX:NEXTI
190 POKEE,2
200 FORI=195TO205
210 FORG=145TO155
220 POKEA,I
230 NEXTG
240 FORY=168TO178
250 POKEB,I
260 NEXTY
270 POKEC,I
280 FORX=1TO300
290 NEXTX:NEXTI
300 POKEE,0
```

Program 16: Bomb

This program displays a moving bomb and produces the appropriate sound effects.

```
  5 S1=36874:S2=36875:S3=36876:S4=36877:V=36878
  8 PRINTCHR$(147)
  9 POKE36879,8
 10 POKEV,15
 20 FORI=225TO195STEP-1
 30 POKES1,I:POKES2,I+4:POKES3,I+20:POKES4,I+25
 40 FORD=1TO21:NEXTD:NEXTI
 50 FORI=7680TO8185STEP23
 60 POKEI,81
 70 FORD=1TO21:NEXTD
 80 POKEI,32
 90 NEXTI
100 POKES1,0:POKES2,0:POKES3,0:POKES4,0
110 POKES4,220
120 FORC=1TO3
```

```
130 FORL=15-CTO8STEP-0.125
140 POKEV,L
150 FORX=8TO255STEP43
160 POKE36879,X:NEXTX
170 NEXTL:NEXTC
180 POKEV,0
190 POKE36879,27
200 END
```

Program 17: President

This program, which requires the Super Expander, produces a simple quiz that uses sound effects.

```
 10 PRINT"♥"
 15 PRINT"THE NAME OF THE PRESIDENT OF THE USA IS...?"
 20 PRINT:PRINT:PRINT"CHOOSE ONLY ONE NUMBER.."
 30 PRINT"1.MARGARET THATCHER"
 35 PRINT"2.PIERRE TRUDEAU"
 40 PRINT"3.RONALD CORBETT"
 45 PRINT"4.RONALD REAGAN"
 50 PRINT"5.HAILIE SELASSIE"
 60 INPUTA
 70 IFA=4THENGOTO200
 80 IFA=5THEN300
 85 IFA=1THEN400
 90 IFA=2THEN500
 95 IFA=1THEN100
100 PRINT"■T2V9S202CRRS101GG#GRGRRRBRRS202CRRRRR"
110 GOTO10
200 PRINT"■T3V9S303GRRERCRRERRGRRBRRRRRR"
210 GOTO10
300 PRINT"■T3V9S201CRRRRCRRRCRCRRRR$ERRRDRDRRCRRCRRRRS
    101BRRS201CRRRR"
310 GOTO10
400 PRINT"■T3V9S201GRRGRR#FRGRARRERRRRDRRRR"
410 GOTO10
500 PRINT"■T3V9S102#FRRRARRRARDRRRRRRRERR#FRRGRRARRBR
    RERRRRRR"
510 GOTO10
```

Program 18: The Hill

This program, which requires the Super Expander, produces a changing picture.

```
 10 GRAPHIC2:COLOR11,6,6,6
 50 DRAW2,0,1023TO100,820
 60 DRAW2,100,820TO110,840
 65 COLOR11,5,5,5
 70 DRAW2,110,840TO190,623
 80 DRAW2,190,623TO300,790
 85 COLOR11,6,1,6
 90 DRAW2,300,790TO460,510
100 DRAW2,460,510TO825,900
105 COLOR11,5,7,5
110 DRAW2,825,900TO1023,1023
115 FORD=1TO1500:NEXT
```

```
120 CIRCLE15,890,100,20,20
130 DRAW7,460,510TO870,120
140 DRAW7,825,900TO910,80
142 COLOR11,5,5,5
145 FORD=1TO1800:NEXT
150 CIRCLE1,500,100,70,30
160 CIRCLE1,600,120,95,30
170 PAINT1,500,100
180 PAINT4,600,120
190 CHAR3,8,"THE HILL"
195 FORD=1TO2000:NEXTD
198 FORI=1TO3
200 PRINT"■T7V3S2O2CCEG"
210 PRINT"■S3O2CCC"
220 PRINT"■S2O2AA
222 FORD=1TO400:NEXT
225 PAINT6,0,100
230 REGION2:CHAR10,5,"GOODNIGHT"
```

Program 19: Wave

This program uses a mathematical formula to create a special wave pattern.

```
10 GRAPHIC2:COLOR2,3,7,5
15 Z=500
16 A=1
18 V=30:H=15
19 FORI=1TO10
20 FORX=200TO450:Y=Z+SIN(Z/V)*COS(X/H)*50
30 POINTA,X,Y
40 NEXTX
50 Z=Z+20
60 V=V+1:H=H+1
65 A=A+1
70 NEXTI
80 END
```

Program 20: Pattern

This simple program produces an interesting graphic pattern.

```
 20 CLR
 30 FORT=1TO31:A$=A$+CHR$(RND(11)+118):NEXT
 40 FORX=1TO14
 50 PRINTA$;:NEXT
 60 PRINTA$;
 70 PRINTA$;
 80 PRINTLEFT$(A$,15);
 90 FORI=1TO900:NEXT
100 GOTO20
```

Program 21: Circle

This program requires the Super Expander.

```
10 GRAPHIC2:COLOR1,7,0,10
20 CIRCLE1,500,500,275,400
30 DRAW1,250,300TO625,165
```

```
 40 DRAW1,625,165TO760,500
 50 DRAW1,760,500TO680,800
 60 DRAW1,680,800TO370,823
 70 DRAW1,370,823TO250,300
 80 DRAW2,250,300TO760,500
 90 DRAW2,625,165TO680,800
100 DRAW2,760,500TO370,823
110 DRAW2,680,800TO250,300
120 DRAW2,370,823TO625,165
```

Program 22: Squiral

This program, which requires the Super Expander, uses a mathematical formula to create a design that changes colors.

```
  5 U=0:V=1
  6 SCNCLR
 10 GRAPHIC2:COLORU,3,V,7
 20 R=.8:P1=π/30:P2=2*π/3
 30 FORT=0TO4.08STEPP1
 40 R=R*1.17557
 50 X1=COS(T)*R+500:Y1=SIN(T)*R+500
 60 A=T*P2
 70 X2=COS(A)*R+500:Y2=SIN(A)*R+500
 80 DRAW1,X1,Y1TOX2,Y2
 90 B=T+2*P2
100 X1=COS(B)*R+500:Y1=SIN(B)*R+500
110 DRAW1,X1,Y1TOX2,Y2
120 X2=COS(T)*R+500:Y2=SIN(T)*R+500
130 DRAW1,X1,Y1TOX2,Y2
140 NEXT
145 U=U+1:V=V+1
147 FORD=1TO2000:NEXT
150 GOTO6
```

Program 23: Torch

This program, which requires the Super Expander, also uses a mathematical formula to create a design that changes colors.

```
  5 U=0:V=1
  6 SCNCLR
 10 GRAPHIC2:COLORU,3,V,7
 20 A=100:B=180:C=90
 30 AN=360/A
 40 FORN=1TOASTEP4:T=N*AN
 50 X=B*COS(T)+200
 60 Y=SIN(T)+150
 70 DRAW1,800,800TOX,Y
 80 NEXT
 90 FORD=1TO2500:NEXT
 95 U=U+1:V=V+1
100 GOTO6
```

Program 24: Hat

This program, which requires the Super Expander, uses a complex set of mathematical formulas to produce a hat.

```
 10 GRAPHIC2:COLOR0,5,7,7
 20 P=490:Q=450
 30 XP=300:XR=2.2*π
 40 YP=100:YR=1:ZP=75
 50 XF=XR/XP:YF=YP/YR:ZF=XR/ZP
 60 FORZI=-QTOQ-1
 70 IFZI<-ZPORZI>ZPGOTO150
 80 ZT=ZI*XP/ZP:ZZ=ZI
 90 XL=INT(.5+SQR(XP*XP-ZT*ZT))
100 FORXI=-XLTOXL
110 XT=SQR(XI*XI+ZT*ZT)*XF:XX=XI
120 YY=(SIN(XT)+.4*SIN(3*XT))*YF
130 GOSUB170
140 NEXTXI
150 NEXTZI
160 STOP
170 X1=XX+ZZ+P
180 Y1=YY-ZZ+Q
190 POINT2,X1,Y1
200 IFY1=0GOTO220
210 POINT2,X1,Y1-1,X1,0
220 RETURN
```

Program 25: Orion

This program, which requires the Super Expander, shows the constellation Orion.

```
10 GRAPHIC3:COLOR0,0,1,7
20 POINT1,20,20,225,50,100,275,150,250,200,225,30,500,
   275,450
30 CHAR18,2,"ORION"
```

Program 26: Formula

This program, which requires the Super Expander, again also produces graphic effects using mathematical formulas.

```
 10 GRAPHIC3:COLOR0,0,4,0
 20 FORA=0TO60STEP.2
 30 R=30*COS(2*A/3)
 40 POINT3,500+8*R*COS(A),500+12*R*SIN(A)
 60 CHAR2,2,"R=30*COS(2*A/3)"
500 NEXTA
```

Program 27: Stave

This program, which requires the Super Expander, produces a set of music lines.

```
 5 PRINT"♥"
10 GRAPHIC3:COLOR0,0,1,0
12 Y=20
15 FORI=1TO5:Y=Y+30
20 DRAW1,20,YTO1000,Y
25 NEXTI
30 Y=190
35 GOTO15
```

Program 28: Ripple

This program is a sorting subroutine that you can insert in your own programs. A$ is used for the strings.

```
110 C=0:N=N-1
120 IFN=0THEN200
130 FORI=1TON
140 IFA(I)<=A(I+1)THEN190
150 T=A(I)
160 A(I)=A(I+1)
170 A(I+1)=T
180 C=1
190 NEXTI
195 IFC=1THEN110
200 PRINTT
```

Program 29: Simple

This program produces a simple screen display. Change the number of spaces in line 20 for different effects.

```
 5 PRINT"J"
10 FORI=1TO20
20 PRINT"*
25 FORX=1TO100
26 NEXTX
30 NEXTI
35 GOTO35
```

Program 30: Trumpet

This program produces some fast moving "music". Figure out why it blips at the end.

```
10 READH:READL
16 DATA200,20,206,20,212,20,200,20
17 DATA206,20,212,20,215,20,206,20
18 DATA212,20,215,20,221,20,212,20
20 DATA200,20,206,20,212,20,200,20,206,20
21 DATA212,20,215,20,206,20,197,20,200,20
26 DATA,206,20,197,20,200,20,212,20
27 DATA206,20,200,20,206,20,215,20
30 DATA200,20,197,20,200,20,0,20,0,20
31 DATA0,20,200,20,206,20,200,20,206,20
36 DATA200,20,206,20,200,20,206,20
37 DATA200,20,206,20,200,20,206,20
38 DATA200,20,206,20,200,20,206,20
40 DATA200,20,206,20,200,20,206,20,200,20
50 DATA200,1000,1
200 IFH=1THEN900
400 POKE36878,15
500 POKE36875,H
600 FORT=1TOL:NEXT
700 POKE36878,0
800 GOTO10
900 END
```

Program 31: Song

This program produces an innocent quiz game with a bang. You can extend the program to include other songs.

```
 10 PRINT:PRINT
 15 PRINT:PRINT"DO YOU KNOW THE TUNE  'BELIEVE ME IF
    ALL CHARMS THOSE ENDEARING YOUNG
 20 INPUTX$
 30 PRINT"I CAN PLAY IT";
 40 PRINT:PRINT"YOU CAN PLAY IT TOO
    USING THE NUMBERS  1 TO 8"
 45 FORI=1TO5000:NEXTI
 60 PRINTCHR$(147)
100 PRINT"THIS IS HOW IT GOES"
110 PRINT:PRINT"THE. NUMBERS ARE PLAYED IN THIS ORDER:"
115 FORI =1TO 1500:NEXTI
120 PRINT" 3 2 1 2 1 1 3 5 4 6 8.8"
200 S2=36875:V=36878
210 POKEV,12
215 READP
220 IFP=-1THEN290
230 READD
240 POKES2,P
250 FORN=1TOD:NEXTN
260 POKES2,0
270 FORN=1TO20:NEXTN
280 GOTO215
290 POKEV,0
300 DATA207,200,201,200,195,600,201,200,195,400
310 DATA195,400,207,400,215,400,209,400,219,400
320 DATA225,400,225,800,-1
330 PRINT:PRINT:PRINT"OK--YOU TRY IT. START ON NUMBER 3"
400 POKEV,8
405 POKES2,0
410 FORY=1TO8
420 READA(Y)
430 NEXTY
440 DATA195,201,207,209,215,219,223,225
450 GETA$:IFA$=""THEN450
460 Y=VAL(A$)
470 IFY=8THEN600
480 POKES2,0
490 FORT=1TO25:NEXTT
500 POKES2,A(Y)
510 GOTO450
520 POKES2,0
600 S4=36877:PRINTCHR$(147)
610 POKES4,220
620 FORC=1TO4
630 FORL=15-(C*2)TO8-(C*2)STEP-0.5
640 POKEV,L
650 FORX=9TO255STEP45
660 POKE36879,X:NEXTX
670 NEXTL:NEXTC
680 POKEV,0
685 POKES4,0
690 POKE36879,27
```

```
695 FORI=1TO2000:NEXTI
700 PRINT"GOT YOU!!!!!"
710 RUN
```

Program 32: Chromatic

This program produces a chromatic scale.

```
  10 S1=36874:S2=S1+1:S3=S2+1:V=S1+4
 100 DIMN(36,1):FORI=0TO35:READN(I,0),N(I,1):NEXT
 200 FORI=0TO36:POKEV,15:FORJ=0TO200:POKES1,N(I,0):
     POKES1,N(I,1):NEXTJ:POKEV,0:NEXTI
9000 DATA131,131,140,145,145,151,151,154,156,161,162,
     166,167,173,174,178,178,181 ,182
9010 DATA185,186,189,190,192,195,197,197,200,200,203,
     203,206,207,208,209,211,212 ,214,214
9020 DATA216,216,218,219,220,221,222,223,224,224,226,
     226,227,228,229,229,231,231,232,232
9030 DATA233,233,234,235,235,236,237,238,239,239,239,
     240,240,241
```

Program 33: Dot

This program, which requires the Super Expander, produces an animated display.

```
5 A=0:B=1:C=3
10 GRAPHIC2:COLORA,B,C,0
15 X=1000:Y=1000
18 FORI=1TO100
20 POINTC,X,Y
25 FORD=1TO50:NEXT
30 POINT0,X,Y
40 X=X-5:Y=Y-5
50 NEXTI
52 COLORA,B,C,0
53 A=A+1:B=B+1:C=C+1
54 IFA=15THEN:GOTO60
55 GOTO15
60 GRAPHIC0
200 GRAPHIC2:COLOR0,0,3,0
210 A=0:B=0
220 CHARA,B,"HERE"
225 FORD=1TO20:NEXTD
230 CHARA,B,"    "
240 A=A+1:B=B+1
245 IFA=15ANDB=15THEN210
250 GOTO220
```

Program 34: Critter

This program, which requires the Super Expander, produces a circle that both moves and expands.

```
10 GRAPHIC2:COLOR0,0,3,0
20 X=20:Y=20:RX=10:RY=10
25 FORI=1TO500
30 CIRCLE3,X,Y,RX,RY
```

```
40 CIRCLE0,X,Y,RX,RY
45 X=X+20:Y=Y+20:RX=RX+2:RY=RY+4
50 NEXTI
```

Program 35: Globe

This program requires the Super Expander.

```
 10 GRAPHIC2:COLOR0,0,5,0
 20 CIRCLE5,511,511,370,510
 30 CIRCLE5,511,511,300,510
 40 CIRCLE3,511,511,200,510
 50 CIRCLE3,511,511,100,510
 60 CIRCLE3,511,511,370,80
 70 CIRCLE3,506,300,327,80
 80 CIRCLE3,514,773,322,80
 90 CIRCLE3,508,98,205,30
100 CIRCLE3,514,945,205,30
```

Program 36: File

This program produces a simple, menu-driven data base program.

```
   5 REM"FILE"
  10 PRINT"♥"
  15 CLOSE1
  20 PRINTTAB(2);1;"INITIALISE FILE"
  30 PRINT:PRINTTAB(2);2;"ADD ENTRIES"
  40 PRINT:PRINTTAB(2);3;"SEARCH,DELETE,
     PRINT,ALTER"
  50 PRINT:PRINTTAB(2);4;"SAVE TO TAPE"
  60 PRINT:PRINTTAB(2);5;"EXAMINE FILE"
  70 PRINT:PRINTTAB(2);6;"PRINT FILE"
  80 PRINT:PRINTTAB(2);7;"NUMBER OF SLOTS       EMPTY"
  90 PRINT:PRINT:PRINTTAB(4);"# ENTER CHOICE #"
 100 INPUTC
 200 IFC<1ORC>7ORC<>INT(C)THEN100
 210 ONCGOTO1000,1500,2000,2500,3000,3500,4000
 299 STOP
 300 PRINT"♥"
 310 FORI=STO1STEP-1
 320 PRINTA$(I):NEXTI
 330 STOP
 500 A$(S)=""
 510 S=S-1:RETURN
1000 PRINT"♥"
1001 PRINT:PRINT:PRINTTAB(2);"THIS PROCEDURE WILL
     DESTROY ALL PREVIOUS   RECORDS"
1003 PRINT:PRINT:PRINTTAB(4);"PRESS 'C' TO";
1004 PRINT:PRINTTAB(4);"CONTINUE OR
     'RETURN' TO                RESTART"
1005 INPUTC$
1006 IFC$="C"THEN1010
1008 GOTO10
1010 PRINT"♥"
1015 PRINT:PRINTTAB(2);"ENTER SIZE OF EACH    ITEM"
1020 PRINT:PRINTTAB(7);"MAX 255"
1030 INPUTA
```

```
1040 IFA<1ORA>255ORA<>INT(A)THEN1030
1045 GOSUB5000
1050 PRINT:PRINT:PRINTTAB(2);"ENTER NUMBER OF
     ITEMS"
1060 PRINT:PRINTTAB(2);"MAX=INT(10000/A)-1"
1070 INPUTB
1080 IFB<1ORB>INT(10000/A)-1ORB<>INT(B)THEN1070
1090 DIMA$(B+1)
1100 S=0
1110 PRINT"♥"
1120 IFS>=BTHEN6000
1130 PRINT"ENTER ITEM FOR SLOT # ";S+1;"ENTER '@@@' TO
     END"
1140 S=S+1
1145 IFS>BTHEN6000
1150 INPUTA$(S)
1155 IFA$(S)=""THEN1150
1160 IFA$(S)="@@@"THEN1200
1170 PRINT"PREVIOUS ENTRY",,A$(S)
1175 IFS>BTHEN1190
1180 GOTO1130
1190 PRINT"♥"
1195 PRINT:PRINT:PRINT:PRINT"ALL SLOTS FULL...WAIT
     WHILE SORTING"
1200 IFA$(S)="@@@"THENGOSUB500
1210 FORI=1TOS:FORJ=1TOS
1220 B$=A$(J)
1230 IFA$(J+1)>A$(J)THEN1250
1240 GOTO1270
1250 A$(J)=A$(J+1)
1260 A$(J+1)=B$
1270 NEXTJ:NEXTI
1290 GOTO10
1500 PRINT"♥"
1550 GOTO1110
2000 PRINT"♥"
2010 PRINT:PRINT"ENTER NAME OR OTHER SEARCH DATA",,,
2020 INPUTD$
2030 IFLEN(D$)>ATHENPRINT:PRINT:PRINTTAB(2);"LENGTH
     TOO GREAT"
2040 IFLEN(D$)>ATHEN2020
2050 FORI=STO1STEP-1
2070 IFA$(I)=D$THENPRINT"SLOT #";I,,A$(I)
2090 NEXTI
2100 PRINTTAB(2)"PRESS",," 'A' TO ALTER"," 'P'
     TO PRINT"," 'D' TO DELETE"
2110 INPUTC$
2120 IFC$<>"P"ANDC$<>""ANDC$<>"A"ANDC$<>"D"THEN2110
2130 IFC$=""THEN10
2300 PRINT"                          "
2310 PRINT:PRINT"ENTER SLOT#"
2320 PRINT"                          "
2325 INPUTN
2330 IFC$="P"THEN2450
2335 IFC$="D"THEN2400
2340 PRINT"ENTER NEW DATA FOR SLOT #";N
2350 INPUTA$(N)
2360 GOTO1200
2399 STOP
```

```
2400 A$(N)=""
2410 S=S-1
2420 GOTO1200
2430 GOTO10
2435 OPEN1,5,0,A$(N)
2490 GOTO10
2500 PRINT"♥"
2510 PRINTTAB(3);"ENTER NAME OF FILE"
2520 INPUTN$
2530 PRINT:PRINT:PRINTN$
2540 PRINT"WILL NOW BE SAVED"
2550 PRINT"PRESS RETURN WHEN TAPE IS READY"
2560 INPUTC$
2570 SAVEN$
2580 GOTO10
3000 PRINT"♥"
3010 FORJ=STO1STEP-16/X
3020 FORI=JTO(J-16/X)+1STEP-1
3025 IFI=0THEN3200
3030 PRINTA$(I)
3040 NEXTI
3050 PRINTTAB(2)"PRESS RETURN TO        CONTINUE"
3060 INPUTC$
3070 PRINT"♥"
3100 NEXTJ
3120 GOTO10
3200 I=J-1
3210 J=1
3220 GOTO3050
3500 OPEN1,4
3510 FORI=S-1TO1STEP-1
3515 PRINT#1,A$(I)
3520 NEXTI
3525 PRINT#1:CLOSE1
3530 GOTO10
4000 PRINT"♥"
4010 PRINT:PRINT"NUMBER PF EMPTY SLOTS= ";B-S
4020 PRINT"PRESS RETURN TO CONTINUE"
4030 INPUTC$
4040 GOTO10
5000 X=A/22
5010 IFINT(X*22)<>ATHENX=INT(X)+1
5020 RETURN
6000 PRINT"♥"
6010 PRINT:PRINT:PRINT:PRINT"ALL SLOTS FULL"
6020 FORI=1TO50:NEXTI
6030 GOTO10
```

Program 37: Jupiter Rendezvous

This program, which requires the Super Expander, creates a challenging spacecraft game,

```
 5 PRINT"♥"
 6 PRINT:PRINT:PRINT:PRINT:
10 PRINTTAB(2)"JUPITER RENDEVOUS"
15 FORD=1TO2000:NEXTD
16 GOTO2000
```

```
 17 COLOR1,3,6,0
 19 PRINT"♥"
 20 PRINT"WOULD YOU LIKE INSTRUCTIONS?"
 25 INPUTA$:IFA$="YES"THEN GOSUB1000
 26 INPUT"NAME PLEASE";N$
 30 PRINT"♥":PRINT"GOOD......LUCK!!!"
 35 FORD=1TO2000:NEXTD
 40 PRINT:
 45 TI$="000000":H=500:V=50:F=120
 50 IFB=0THEN65
 55 PRINTTI$  ;TAB(4);H;TAB(12);
 57 PRINTV;TAB(4);F;TAB(12);
 58 PRINT"I";TAB(H/4+18);"**"
 60 GOTO70
 63 PRINTTAB(4)"TIME";TAB(12)"HEIGHT"
 65 PRINTTI$  ;TAB(4);H;TAB(12);
 66 PRINTTAB(4)"SPEED"
 67 PRINTV;TAB(4);F;TAB(12);
 68 PRINT"I"TAB(H/4+18);"**"
 70 INPUTB
 75 IFB<0ORB>39THEN150
 80 IFB>30THEN150
 85 IFB>FTHEN95
 90 GOTO100
 95 B=F
100 V1=V-B+5
105 F=F-B
110 H=H-.5*(V+V1)
115 IFH<=0THEN160
125 V=V1
130 IFF>0THEN50
135 IFB=0THEN145
140 PRINT"LLLOUT OF FUEL⊥⊥⊥"
145 PRINTTI/60;TAB(4);H;TAB(12);
147 PRINTV;TAB(4);F;TAB(12);
148 PRINT"I";TAB(H/4+18);"***"
150 B=0
152 FORD=1TO3000:NEXTD
160 PRINT"**WELCOME TO JUPITER**"
165 FORD=1TO3000:NEXTD
170 H=H+.5*(V+V1)
175 IFB=5THEN190
180 D=(-V+SQR(V*V+H*(10-2*B)))/(5-B)
185 GOTO195
190 D=H/V
195 V1=V+(5-B)*D
200 PRINT"JUPITER AT";T+D;"SECONDS"
205 PRINT"LANDING SPEED=";V1;"FT/SEC"
210 PRINTF;"UNITS OF FUEL LEFT"
212 FORD=1TO3000:NEXTD
215 IFV1<>0THEN2500
220 PRINT"GOOD SHOW...."
225 PRINT"HAVE YOU THOUGHT HOW YOU WILL GET BACK?"
227 FORD=1TO3000:NEXTD
230 IFABS(V1)<5THEN245
235 PRINT"JUPITER PILOT ";N$;" HAD A NASTY ACCIDENTTODAY"
236 PRINT"WHILE FAILING TO LAND SAFELY ON JUPITER"
240 PRINT"HE IS SURVIVED BY......."
```

```
 245 PRINT"WOULD YOU LIKE ANOTHER MISSION?"
 250 INPUT A$
 255 IFA$="YES"THENPRINT"GLUTTON FOR PUNISHMENT, AREN'T
     YOU?"
 256 FORD=1TO3500:NEXTD
 257 IFA$="YES"GOTO30
 260 PRINT:
 265 PRINT"SEE YOU"
 270 PRINT:
 275 END
1000 PRINT"   YOU HAVE APPROACHED THE PLANET JUPITER
     AND ARE READY TO LAND"
1001 FORD=1TO2000:NEXTD
1002 PRINT:PRINT:PRINT"    UNFORTUNATELY YOUR
     COMPUTER IS ONLY      PARTIALLY FUN CTIONAL"
1003 PRINT"DO YOU UNDERSTAND?"
1004 INPUTA$:IFA$="NO"THENGOSUB1500
1005 PRINT"J"
1006 PRINT"THE COMPUTER CAN GIVE YOU READINGS
     ON RATE   OF DESCENT, FUEL ";
1007 PRINT"      REMAINING AND OTHER     MESSAGES"
1008 FORD=1TO3000:NEXTD
1009 PRINT"J"
1010 PRINT"THIS MEANS...............YOU MUST
     LAND THE    CRAFT....YOURSELF"
1011 PRINT:PRINT:PRINT"AFTER  EACH SECOND OF
     FLIGHT YOU WILL BE    TOLD YOUR HEIGHT,";
1012 PRINT" YOUR          SPEED AND FUEL"
1013 FORD=1TO2000:NEXTD
1014 PRINT"A ? WILL APPEAR."
1015 FORD=1TO4000:NEXTD
1016 PRINT"J"
1017 PRINT:PRINT:PRINT"   YOU MUST ENTER THE NUMBER OF
     UNITS OF    FUEL";
1018 PRINT"YOU WISH TO BURN"
1019 FORD=1TO2500:NEXTD
1020 PRINT"SHOULD YOU CRASH THE    APPROPRIATE
     OBITUARY WILL APPEAR"
1025 RETURN
1500 PRINT"J"
1502 PRINT:PRINT:PRINT"THAT MEANS:............
     THE COMPUTER WORKS.....BUT ONLY JUST!!!"
1503 FORD=1TO3000:NEXTD
1505 RETURN
2000 GRAPHIC3:COLOR0,0,5,2
2005 CIRCLE3,500,500,200,300
2007 CIRCLE2,575,575,20,18
2010 PAINT2,500,500
2030 CHAR2,5,"JUPITER"
2035 CHAR19,4,"RENDEVOUS"
2040 FORD=1TO3000:NEXTD
2045 GRAPHIC4
2050 GOTO17
2500 GRAPHIC3:COLOR6,5,1,7
2505 CIRCLE1,500,1023,500,50
2510 DRAW1,450,973TO592,930TO603,940
2520 DRAW1,603,940TO641,980TO682,960TO702,973
2530 CHAR4,2,"C R A S H"
```

```
2540 POINT7,420,920,405,902,590,955
2550 CIRCLE7,700,700,5,5
2560 FORD=1TO5000:NEXTD
2570 GRAPHIC4:COLOR1,3,6,0:GOTO230
```

Program 38: ? Lines

This program, which requires the Super Expander, creates a pattern of perspective lines.

```
10 GRAPHIC2:COLOR0,0,1,5
15 READK
20 IFK<0THEN150
30 FORX=0TO1000
40 IFX*K>200THEN100
50 POINT5,X,X*K
60 NEXTX
100 GOTO15
150 GOTO150
160 DATA.01,.1,.2,.5,.8,1,5,10,30,-1
```

Program 39: Ellipse

This program, which requires the Super Expander, provides a slow method of drawing colored ellipses.

```
10 R=50
12 A=1:C=0
15 GRAPHIC3:COLOR0,0,3,7
20 FORX=-RTORSTEP.1
30 Y=SQR(R*R-X*X)
40 POINT3,500+7*X,500+3*Y
50 POINT3,500+7*X,500-3*Y
60 NEXT
65 C=C+1
70 R=R-3
75 IFC=20GOTO90
80 GOTO15
90 END
```

Program 40: Spiral

```
10 GRAPHIC3:COLOR0,0,4,0
20 FORA=0TO100STEP.2
30 R=A
40 POINT3,500+5*R*COS(A),500+1*R*SIN(A)
50 NEXTA
```

Program 41: Random

This program shows you what happens when you let the computer choose what to write.

```
 5 PRINT"♥"
10 DIMA$(20)
15 DIMB$(20)
```

```
 20 DIMC$(20)
 25 DIMD$(20)
 30 GOSUB501
 35 FORD=1TO1000:NEXTD
 40 PRINTA$(1)+A$(3)+A$(10)+A$(1)+A$(4)
 45 C=0
 70 J=INT(RND(1)*20)+1
 80 Z$=A$(J)
 90 PRINTZ$;
102 PRINT
105 C=C+1
110 FORD=1TO2000:NEXTD
125 IFC>=25THENEND
130 Z$=A$(J):PRINTZ$;
131 J=INT(RND(1)*20)+1
132 Z$=B$(J):PRINTZ$;
133 J=INT(RND(1)*20)+1
134 Z$=C$(J):PRINTZ$;
135 J=INT(RND(1)*20)+1
136 Z$=D$(J):PRINTZ$;
140 GOTO70
499 END
501 A$(1)="THE "
502 A$(2)="ANY "
503 A$(3)="SOME "
504 A$(4)="MANY "
505 A$(5)="NO "
506 A$(6)="A "
507 B$(1)="CAT "
508 B$(2)="DOG "
509 B$(3)="ELEPHANT "
510 B$(4 )="RAIN "
511 B$(5 )="HOUSE "
512 B$(6 )="PIANO "
513 C$(1 )="MEOWS "
514 C$(2 )="BARKS "
515 C$(3 )="RUNS "
516 C$(4 )="GROWLS "
517 C$(5 )="BITES "
518 C$(6 )="SCRATCHES "
519 C$(7 )="EATS "
520 D$(1)="FROM "
521 D$(2)="TO "
522 D$(3)="AROUND "
523 D$(4)="UNDER "
524 D$(5)="AT "
525 D$(6)="BEYOND "
600 RETURN
```

Program 42: Word

This program produces a party game.

```
10 PRINT"EACH TIME YOU SEE THE QUESTION MARK"
20 PRINT"TYPE IN THE PART OF SPEECH ASKED FOR"
30 FORD=1TO1500
40 PRINT"♥"
50 DIMA$(20)
60 INPUT"A NOUN";A$(1)
```

```
 70 INPUT"VERB INFINITIVE";A$(2)
 80 INPUT"ADVERB";A$(3)
 90 INPUT"LIVING THING (MALE)";A$(4)
100 INPUT"AN EXCLAMATION";A$(5)
110 INPUT"ADJECTIVE";A$(6)
120 INPUT"PLURAL NOUN";A$(7)
130 PRINT"FAMOUS CHAIN STORE"
135 INPUTA$(8)
140 INPUT"ADVERB";A$(9)
150 INPUT"PART OF BODY";A$(10)
160 PRINT"PART OF BODY...AGAIN!"
165 INPUTA$(11)
170 PRINT"VERB. 3RD PERSON IMPERFECT TENSE"
175 INPUTA$(12)
180 INPUT"ADVERB OF TIME";A$(13)
190 INPUT"PLURAL NOUN";A$(14)
200 INPUT"VERB";A$(15)
210 PRINT"VERB. IMPERFECT TENSE"
215 INPUTA$(16)
220 INPUT"VERB IMPERFECT TENSE";A$(17)
230 INPUT"PLACE-NOUN";A$(18)
240 INPUT"VERB.IMPERFECT TENSE";A$(19)
250 FORD=1TO2000:NEXTD
260 PRINT"⊐"
270 PRINT"ONCE UPON A TIME THERE WAS A ";
280 PRINTA$(1);" WHO USED TO LIKE TO ";
290 PRINTA$(2);". ONE DAY, THE ";
300 PRINTA$(1);" WAS DOING THIS WHEN ";
310 PRINTA$(3);" A ";A$(4);
320 PRINT" STROLLED BY "
330 FORD=1TO10000
340 PRINTA$(5);"! I HAVE NEVER SEEN SUCH "
350 PRINTA$(6);" "      A$(7);"  BEFORE. WHERE DID"
360 PRINT"YOU GET THEM?"
370 FORD=1TO9000:NEXTD
380 PRINT"⊐"
390 PRINT"THEY WERE ON SALE AT ";A$(8);
400 PRINT" WAS THE REPLY.'THEY WERE "
410 PRINTA$(9);" CHEAP "
420 PRINT"THE "  A$(4);"'S ";A$(10);
430 PRINT" BRIGHTENED, HIS "A$(11);
440 PRINT" WIDENED AND HE ";A$(13);
450 FORD=1TO9000:NEXTD
460 PRINT" I AM TRULY AMAZED, SAID THE ";A$(4);
470 PRINT" I MUST GO AND GET SOME ";
480 PRINTA$(14);" THEY WILL HELP MY "
490 PRINTA$(15);" "A$(16);
500 PRINT" I WILL COME WITH YOU AND SEE IF "
510 PRINT"THEY FIT, SAID THE "A$(1);
520 FOR D=1TO9000:NEXTD
530 PRINT"⊐"
540 PRINT"AS THEY ";A$(17);
550 PRINT" INTO THE "A$(18);
560 FORD=1TO2000:NEXTD
570 PRINT:PRINT:PRINT"THEY "
580 PRINT:PRINT:PRINT A$(19);"!"
590 FORD=1TO2000:NEXTD
600 PRINT:PRINT:PRINT:PRINT "THE END!"
```

Program 43: Demo

This program, which requires the 8K expansion unit, is a menu-driven program that shows what a computer can do, including teaching, calculating, and sorting.

```
   2 DIMA$(30)
   5 PRINT"]"
  10 PRINT:PRINT"THIS IS A DEMONSTRAT- ION OF THE
     KINDS OF   THINGS A COMPUTER CAN DO"
  15 FORD=1TO4500:NEXT
  20 PRINT"]"
  25 PRINT:PRINT:PRINTTAB(5) "CHOOSE FROM THE
     FOLLOWING:-"
  30 PRINT:PRINT"1.CALCULATE. 2.TEACH"
  35 PRINT:PRINT"3.KEEP FILES. 4. SORT"
  40 PRINT:PRINT"PLEASE PRESS THE      APPROPRIATE
     NUMBER"
  45 INPUTA
  50 ON-(A=1)-2*(A=2)-3*(A=3)-4*(A=4)GOTO1000,2000,
     3000,3000
1000 PRINTCHR$(147)
1010 PRINT"CALCULATION ROUTINE"
1015 PRINT:PRINT"A LADY OPENS AN       ACCOUNT AND
     DEPOSITS  $1 ON THE FIRST DAY"
1020 FORD=1TO4000:NEXT
1025 PRINT:PRINTTAB(2)"EACH DAY THEREAFTER   SHE
     DEPOSITS DOUBLE"
1030 PRINTTAB(2)"THE AMOUNT OF THE     PREVIOUS DAY"
1035 FORD=1TO4500:NEXT
1040 PRINT:PRINT" YOU STATE HOW MANY    DAYS SHE DOES
     THIS"
1045 PRINT" AND I WILL TELL YOU   THE TOTAL AMOUNT AT
     THE END OF THE PERIOD"
1050 INPUTB
1055 T=0
1060 FORN=1TOB
1065 S=2↑N
1070 T=T+S
1075 NEXTN
1077 PRINTCHR$(147)
1080 PRINTTAB(3)"AFTER ";B+1;"DAYS THE TOTAL AMOUNT IN
     THE   ACCOUNT IS:-$";CHR$(156)T
1085 FORD=1TO5000:NEXT
1087 PRINTCHR$(31)
1090 PRINT:PRINT"WOULD YOU LIKE TO TRY THAT AGAIN?"
1095 INPUT"YES OR NO";A$
1100 IF A$="YES" THEN 1000
1105 INPUT"WOULD YOU LIKE TO TRY A DIFFERENT CALCULAT-
     TION PROBLEM";A$
1107 IFA$="YES" THEN 1150
1110 IFA$="NO" THEN 25
1150 PRINT"]"
1152 PRINT"THIS IS A NEW PROBLEM"
1155 PRINT:PRINT
1160 PRINT"A KING DECIDED TO     GIVE AWAY HIS MONEY
    TO EACH OF 1 MILLION  SUBJECTS"
```

```
1170 FORD=1TO4000:NEXT
1171 FORD=1TO3000:NEXT
1175 PRINT:PRINTTAB(2)"TO THE FIRST HE GAVE  $1."
1180 PRINT"TO THE NEXT 2 HE GAVE $2 EACH. TO THE NEXT
     3 HE  GAVE $3 EACH"
1185 FORD=1TO8000:NEXT
1190 PRINT:PRINT"I WILL TELL YOU HOW   MUCH THE
     MILLIONTH    SUBJECT GOT"
1192 FORD=1TO6000:NEXT
1195 PRINT"ℑ"
1197 PRINT"▩▩"
1198 PRINT"I'M WORKING IT OUT NOW"
1200 A=0:B=1
1202 A=A+B
1205 IF A>=1000000 THEN1215
1210 B=B+1:GOTO1202
1215 PRINT"ℑ":PRINT:PRINT"THE LAST SUBJECT GOT ",B
1220 FORD=1TO4500:NEXT
1225 PRINT"■■"
1230 PRINTCHR$(147)
1235 PRINT"NOW YOU GIVE ME A     PROBLEM"
1240 FORD=1TO1000:NEXT
1245 PRINT"CHOOSE FROM THE LIST:-"
1250 PRINT"1.MULTIPLY 2.DIVIDE"
1255 PRINT"3.ADD       4.SUBTRACT"
1260 INPUTB
1265 ON-(B=1)-2*(B=2)-3*(B=3)-4*(B=4)GOTO1300,1400,
     1500,1600
1300 PRINTCHR$(147)
1305 PRINT"ENTER TWO NUMBERS:-"
1307 PRINTCHR$(14)"(ONE AFTER EACH ?)"
1310 INPUTX,Y
1312 PRINTCHR$(147)
1315 PRINTCHR$(142)"■THANKYOU■"
1320 PRINT"THE ANSWER IS:-"
1322 PRINTX;"X";Y
1325 PRINTX*Y
1330 FORD=1TO6000:NEXT
1335 GOTO1245
1400 PRINTCHR$(147)
1405 PRINT"ENTER TWO NUMBERS:-"
1407 PRINTCHR$(14)"(ONE NUMBER AT A TIME)"
1410 INPUTX,Y
1412 PRINTCHR$(147)
1415 PRINTCHR$(142)"■THANKYOU■"
1420 PRINT"THE ANSWER IS:-"
1422 PRINTX;"/";Y
1425 PRINTX/Y
1430 FORD=1TO6000:NEXT
1435 GOTO1245
1500 PRINTCHR$(147)
1505 PRINT"ENTER TWO NUMBERS:-"
1507 PRINTCHR$(14)"(ONE NUMBER AT A TIME)"
1510 INPUTX,Y
1512 PRINTCHR$(147)
1515 PRINTCHR$(142)"■THANKYOU■"
1520 PRINT"THE ANSWER IS:-"
1522 PRINTX;"+";Y
```

```
1525 PRINTX+Y
1530 FORD=1TO6000:NEXT
1600 PRINTCHR$(147)
1605 PRINT"ENTER TWO NUMBERS:-"
1607 PRINTCHR$(14)"(ONE NUMBER AT A TIME)"
1610 INPUTX,Y
1612 PRINTCHR$(147)
1615 PRINTCHR$(142)"■THANKYOU■"
1620 PRINT"THE ANSWER IS:-"
1622 PRINTX;"-";Y
1625 PRINTX-Y
1630 FORD=1TO6000:NEXT
1640 PRINTCHR$(147)
1650 PRINT"THE COMPUTER IS A VERYPOWERFUL TOOL FOR ALL
     KINDS OF ACCOUNTING"
1655 PRINT:PRINT" IT CAN ALSO ASSIST IN THE
     PREPARATION AND  PRODUCTION OF PRINTED DOCUMENTS"
1660 PRINT:PRINT" SUCH PREPARATION IS  CALLED ■WORD
     PROCESSING■"
1665 FORD=1TO9999:NEXT
1670 PRINT" MATERIAL CAN BE TYPED,CORRECTED, RESHAPED
     AND ALIERED IN ANY"
1675 PRINT"WAY THE WRITER CHOOSES ■BEFORE■ BEING
     PRINTED"
1680 FORD=1TO9000:NEXT
1687 PRINT:PRINT:
1690 PRINT"'■HARD COPY■' OF ANY    MATERIAL, WHETHER
     IT  BE ACCOUNTS, LITERATURE ";
1695 PRINT"OR RECORDS OF     STUDENT PROGRESS, CAN BE
     PREPARED IN A SHORTTIME ";
1700 PRINT"BY MEANS OF A ■   PRINTER■"
1705 FORD=1TO10000:NEXT
1710 PRINTCHR$(147)
1715 PRINTTAB(2)"A ■MODEM■ IS A DEVICE WHICH ALLOWS
     YOU TO "
1720 PRINT"TRANSMIT INFORMATION  FROM ONE COMPUTER TO
     ANOTHER VERY RAPIDLY"
1725 PRINT"OVER THE TELEPHONE."
1730 PRINT:PRINT"WITH SUCH A DEVICE    ATTACHED TO
     YOUR COMP-UTER, INFORMATION FROM";
1735 PRINT"ALL OVER THE WORLD IS AT YOUR
     FINGER-TIPS"
1740 FORD=1TO11000:NEXT
1745 PRINTCHR$(147)
1999 GOTO25
2000 PRINT"■"
2002 PRINT:PRINT
2005 PRINT:PRINT"A COMPUTER CAN TEACH  IN A VARIETY OF
     WAYS"
2007 PRINT:PRINT
2010 PRINT"ORDINARY DRILL AND    PRACTICE UNITS
     CAN"
2015 PRINT"REINFORCE ALREADY     LEARNED
     MATERIAL"
2020 PRINT"AND GIVE A SCORE ON   PROGRAMMED
     TESTS"
2025 FORD=1TO12000:NEXT
2027 PRINT:PRINT
```

```
2030 PRINT"TEACHING MATERIAL
     CAN BE PRESENTED IN A GREAT MANY WAYS"
2031 FORD=1TO9999:NEXT
2035 PRINT"♥"
2040 PRINT"▒▒MATERIAL CAN BE      EMPHASISED"
2045 PRINT"▒IN REVERSE█";" ▒AND IN▒"
2050 PRINT"█A";" ▒VARIETY ";"▙OF";" █COLOURS"
2052 FORD=1TO8000:NEXT
2055 PRINT"▒"
2060 PRINT"♥"
2065 PRINTTAB(25)"INSTRUCTIONAL PRO-
     GRAMMES TEACH AT    THE"
2070 PRINT"█STUDENT'S";"▒ RATE--
     ENSURING THAT THE"
2075 PRINT"SUBJECT MATTER IS     LEARNED THOROUGHLY"
2080 PRINT:PRINT"REMEDIAL INSTRUCTION  CAN BE PROVIDED
     WITH- OUT THE STUDENT BEING AWARE
2085 PRINT:PRINI"▒AND IN ABSOLUTE       PRIVACY"
2090 FORD=1TO8000:NEXT
2095 PRINT"▒"
2100 PRINT"ANSWER THE FOLLOWING  QUESTIONS:-"
2105 PRINT"THE COMPUTER FORCES   THE STUDENTS TO WORK
     AT A FIXED RATE"
2110 PRINT"YES OR NO?"
2120 INPUTX$
2130 IFX$="YES"THEN2200
2140 PRINT"♥"
2145 PRINT"YOU ARE RIGHT"
2150 PRINT"A CAREFULLY PREPARED  PROGRAMME WILL ALLOW"
2155 PRINT"A STUDENT TO WORK AT  HIS OR HER OWN RATE"
2160 FORD=1TO7000:NEXT
2175 PRINT"ALTHOUGH IT IS POSS-
     IBLE TO INCREASE THE  LEARNING RATE"
2180 PRINT"♥"
2185 PRINT"ONLY CERTAIN TYPES OF COMPUTER CAN BE USED
     TO TEACH--YES OR NO?"
2190 INPUTY$
2195 IFY$="YES"THEN2300
2198 IFY$="NO"THEN2400
2200 PRINT"♥"
2205 PRINT"WHILE IT IS POSSBILE  TO CONTROL OR EVEN
    INCREASE THE LEARNING"
2210 PRINT"RATE, MOST PROGRAMMES ARE DESIGNED TO ALLOW
     SELF-PACED LEARNING"
2215 PRINT"TO TAKE PLACE"
2216 FORD=1TO9999:NEXT
2220 GOTO2180
2300 PRINT"♥"
2305 PRINT"WHILE IT IS TRUE THAT SOME COMPUTERS HAVE
     FEATURES NOT FOUND"
2310 PRINT"ON OTHERS, IN GENERAL ANY COMPUTER CAN BE
     USED"
2315 FORD=1TO9999:NEXT
2320 PRINT"IF COLOUR AND GRAPHIC DISPLAY ARE ESSENTIAL
     TO THE MATERIAL THEN"
2325 PRINT"OBVIOUSLY A COMPUTER  SUCH AS THE VIC20 OR
     ATARI IS NEEDED."
2330 PRINT"HOWEVER, OVERUSE OF   SUCH FEATURES";
```

```
2335 PRINT" ▥CAN";"      ■O";"▮FTEN";"▩▩ DETRACT ";
2340 PRINT"▩FROM ";"■THE EFFECT   RATHER THAN
     ENHANCE   IT"
2345 FORD=1TO9999:NEXT
2350 PRINT"▯"
2355 PRINT"THE PREPARATION OF    GOOD, EFFECTIVE ▩CAI▩
     MATERIAL IS A LENGTHY PROCESS";
2360 PRINT" YET THE MATER-IAL CAN  BE USED OVER AND
     OVER AGAIN. IT CAN EVEN BE"
2365 PRINT"STORED ON TAPE TO BE  USED AT HOME."
2370 PRINT: PRINT"THE ONLY CONDITION IS THAT THE SAME
     TYPE OF MACHINE BE USED."
2375 FORD=1TO12000:NEXT
2380 PRINT:PRINT"ONCE WRITTEN, A UNIT  CAN BE ALTERED
     TO SUITNEW AND CHANGING"
2385 PRINT"REQUIREMENTS."
2390 FORD=1TO6000:NEXT
2395 POKE646,4
2398 PRINT:PRINT:
2399 FORD=1TO5000:NEXT:PRINT:PRINT:PRINT:
2405 POKE646,6
2410 FORD=1TO12000:NEXT
2414 PRINT"▯"
2415 GOTO25
3000 PRINT"▯"
3001 PRINT:PRINT:PRINTTAB(5)"┌─────────────┐"
3002 PRINT:PRINT:PRINTTAB(7)"RECORD"
3003 PRINT:PRINT:PRINTTAB(9)"KEEPING"
3004 FORD=1TO4000:NEXT
3005 PRINT:PRINT:PRINTTAB(5)"▮MADE PAINLESS"
3006 PRINT:PRINT:PRINTTAB(4)"▩└─────────────┘"
3007 FORD=1TO4000:NEXT
3010 PRINT"▯"
3015 PRINT"A COMPUTER CAN SORT A GREAT VARIETY OF
     ITEMS"
3020 PRINT"QUICKLY AND SILENTLY----- WITHOUT
     ▩EPITHETS!▩"
3025 FORD=1TO6500:NEXT
3030 PRINTCHR$(147)"FOR EXAMPLE:---"
3035 PRINT:PRINT:PRINT"ENTER A FEW NAMES IN
     ANY ORDER YOU WISH"
3037 FORD=1TO2200:NEXT
3040 PRINT:PRINT"JUST AS THEY OCCUR TO YOU, IN FACT"
3042 FORD=1TO2300:NEXT
3045 PRINT:PRINT"FIRST TELL ME HOW MANYNAMES YOU ARE
     GOING TOENTER"
3047 PRINT"(■LIMIT YOURSELF TO 30▩)"
3050 INPUTN
3052 PRINTCHR$(147)
3055 PRINT"NOW THE NAMES, PLEASE.(PRESS ENTER AFTER
     EACH NAME!)"
3065 FORI=1TON:INPUTA$(I):NEXT
3070 PRINTTAB(8)"NOW SORTING"
3075 FORI=1TON-1
~~3080 IFA$(I+1)>=A$(I)THEN3105~~
3085 B$=A$(I+1)
3090 A$(I+1)=A$(I)
```

```
3095 A$(I)=B$
3100 GOTO3075
3105 NEXTI
3110 FORI=0TON+1:PRINTA$(I):NEXT
3115 FORD=1TO7000:NEXT
3120 PRINT"■WAS THAT FAST ENOUGH?■"
3125 FORD=1TO4000:NEXT
3130 PRINT"WOULD YOU LIKE TO TRY THAT AGAIN?"
3135 INPUTX$
3140 IFX$="YES"THEN3045
3145 PRINTCHR$(147)"A COMPLETE FILING
     SYSTEM CAN BE EXAMINED"
3150 PRINT"ON THE ZX81 COMPUTER"
3155 FORD=1TO2300:NEXT
3160 PRINT:PRINT"ASK THE DEMONSTRATOR  TO SHOW YOU THIS"
3165 FORD=1TO10000:NEXT
3170 PRINTCHR$(147)
3180 PRINT"ANY RECORD KEEPING    SYSTEM MUST ALLOW THE
     USER TO ENTER MATERIAL";
3185 PRINT"IN ANY ORDER."
3187 PRINT:PRINT
3190 PRINT"IT MUST ALSO ALLOW FORTHE RETRIEVAL OF
     MATERIAL";
3195 PRINT" IN ANY ORDER."
3200 FORD=1TO8000:NEXT
3205 PRINT"JUST AS WITH A FILING CABINET, YOU MUST
     KNOWHOW MUCH SPACE YOU    HAVE"
3210 PRINT"BUT FINDING ITEMS IS  AS EASY AS TYPING IN
     THE TITLE"
3215 FORD=1TO10000:NEXT
3220 RUN
```

Program 44: Name Sort

```
  5 PRINTCHR$(147)
 10 DIMA$(20)
 20 INPUT"HOW MANY NAMES";N
 30 FORI=1TON
 40 INPUTA$(I):NEXT
 50 PRINTTAB(8)"NOW SORTING"
 60 FORI=1TON-1
 70 IFA$(I+1)>=A$(I)THEN120
 80 B$=A$(I+1)
 90 A$(I+1)=A$(I)
100 A$(I)=B$
110 GOTO60
120 NEXTI
130 FORI=0TON:PRINTA$(I):NEXTI
```

Program 45: Print Sort

This program sorts names and sends the results to the printer.

```
 5 PRINTCHR$(147)
10 DIMA$(20)
20 INPUT"HOW MANY NAMES";N
30 FORI=1TON
40 INPUTA$(I):NEXT
50 PRINTTAB(8)"NOW SORTING"
60 FORI=1TON-1
```

```
 70 IFA$(I+1)>=A$(I)THEN120
 80 B$=A$(I+1)
 90 A$(I+1)=A$(I)
100 A$(I)=B$
110 GOTO60
120 NEXTI
130 FORI=OTON+1:PRINTA$(I):NEXTI
140 OPEN1,4
150 FORI=0TON+1:PRINT#1,A$(I):NEXTI
160 CLOSE1,4
170 RUN
```

Program 46: The Flight of the Bumble bee.

This program plays this famous music. It is printed courtesy of my youngest son, Michael, who is a trumpet player. It took him quite a while to do the whole thing, and a small lack of care over pressing the keys on the tape recorder meant that a goodly part of the program was lost at one point! This could be the start of something big! A kind of Music-minus-One in reverse. Instead of you playing the solo, you play the accompaniment. Better still, why not get a copy of the music in A minor and program another VIC-20, or even a collection of them, to play the accompaniment. You might form the first VIC orchestra. The gap at the beginning is not an error. It is the spot where the accompaniment plays.

The program runs a lot slower than it should; that is, the music is a lot slower. This is because the program is fairly long. You could, of course, put the data on disk and access it in chunks. In that way you might speed up the process.

There was a point where we had a user-defined image darting all over the screen at random, trying to emulate a bee, but that made the music run even slower!

The moral of all this is to keep things simple and not attempt to program something of this nature that is too long. That puts Mahler, Bruckner, and Wagner right out of the picture.

```
 1 REM"FLIGHT OF THE BUMBLE BEE"
 2 PRINT"♥"
10 READ H
11 READ L
16 DATA 231,1,229,1,227,1,226,1
17 DATA 227,1,226,1,224,1,222,1
18 DATA 224,1,222,1,220,1,218,1
19 DATA 216,1,214,1,211,1,209,1
20 DATA 206.2,0,3300
21 DATA 206,1,203,1,200,1,197,1
22 DATA 192,1,208,1,206,1,203,1
23 DATA 206,1,203,1,200,1,197,1
24 DATA 192,1,197,1,200,1,203,1
25 DATA 206,1,203,1,200,1,197,1
26 DATA 192,1,208,1,206,1,203,1
```

```
27 DATA 206,1,203,1,200,1,197,1
28 DATA 192,1,197,1,200,1,203,1
29 DATA 206,1,203,1,200,1,197,1
30 DATA 200,1,197,1,192,1,189,1
31 DATA 192,1,197,1,200,1,203,1
32 DATA 206,1,208,1,206,1,203,1
33 DATA 206,1,203,1,200,1,197,1
34 DATA 200,1,197,1,192,1,189,1
35 DATA 192,1,197,1,200,1,203,1
36 DATA 206,1,211,1,214,1,216,1
37 DATA 218,1,216,1,214,1,211,1
38 DATA 208,1,220,1,218,1,216,1
39 DATA 218,1,216,1,214,1,211,1
40 DATA 208,1,211,1,214,1,216,1
41 DATA 218,1,216,1,214,1,211,1
42 DATA 208,1,220,1,218,1,216,1
43 DATA 218,1,216,1,214,1,211,1
44 DATA 208,1,211,1,214,1,216,1
45 DATA 218,1,216,1,214,1,211,1
46 DATA 214,1,211,1,208,1,206,1
47 DATA 208,1,211,1,214,1,216,1
48 DATA 218,1,220,1,218,1,216,1
49 DATA 218,1,216,1,214,1,211,1
50 DATA 214,1,211,1,208,1,206,1
51 DATA 208,1,211,1,214,1,216,1
52 DATA 218,1,220,1,218,1,216,1,218,3
53 DATA 181,1,181,1,181,1,181,1,181,1,181,1,181,1,185,1
54 DATA 216,3,216,1,216,1,216,1,216,1,216,1,216,1,218,1
55 DATA 181,1,181,1,181,1,181,1,181,1,181,1,181,1,185,1
56 DATA 216,1,216,1,216,1,216,1,216,1,216,1,216,1,216,1
57 DATA 218,1,220,1,218,1,216,1,218,1,220,1,218,1,216,1
58 DATA 218,1,220,1,218,1,216,1,218,1,220,1,218,1,216,1
59 DATA 218,1,220,1,222,1,224,1,226,1,224,1,222,1,220,1
60 DATA 218,1,220,1,222,1,224,1,226,1,224,1,222,1,220,1
61 DATA 218,3,200,1,200,1,200,1,200,1,200,1,200,1,200,1
       ,203,1
62 DATA 226,1,226,1,226,1,226,1,226,1,226,1,226,1,227,1
63 DATA 200,1,200,1,200,1,200,1,200,1,200,1,200,1,203,1
64 DATA 226,1,226,1,226,1,226,1,226,1,226,1,226,1
65 DATA 227,1,229,1,227,1,226,1,227,1,229,1,227,1,226,1
66 DATA 227,1,229,1,227,1,226,1,227,1,229,1,227,1,226,1
67 DATA 227,1,229,1,231,1,232,1,233,1,232,1,231,1,229,1
68 DATA 227,1,229,1,231,1,232,1,233,1,232,1,231,1,229,1
69 DATA 227,1,226,1,224,1,222,1,220,1,229,1,227,1,226,1
70 DATA 227,1,226,1,224,1,222,1,220,1,222,1,224,1,226,1
71 DATA 227,1,226,1,224,1,222,1,224,1,222,1,220,1,218,1
72 DATA 220,1,222,1,224,1,226,1,224,1,226,1,227,1,229,1
73 DATA 231,1,229,1,227,1,226,1,227,1,226,1,224,1,222,1
74 DATA 224,1,222,1,220,1,218,1,216,1,214,1,211,1,208,1
75 DATA 206,1,208,1,206,1,203,1,206,1,208,1,206,1,203,1
76 DATA 206,1,208,1,206,1,203,1,206,1,208,1,206,1,203,1
77 DATA 206,1,208,1,206,1,203,1,206,1,208,1,206,1,203,1
78 DATA 206,1,208,1,206,1,203,1,206,1,208,1,206,1,203,1
79 DATA 206,1,203,1,200,1,197,1,200,1,197,1,192,1,189,1
80 DATA 192,1,189,1,185,1,181,1,178,1,173,1,166,1,161,1
81 DATA 158,1,161,1,158,1,151,1,158,1,161,1,158,1,151,1
82 DATA 158,1,161,1,158,1,151,1,158,1,161,1,158,1,151,1
83 DATA 158,1,161,1,158,1,151,1,158,1,161,1,158,1,151,1
```

```
  84 DATA 158,1;161,1,158,1,151,1,158,1,161,1,158,1,151,1
  85 DATA 158,1,161,1,166,1,173,1,178,1,181,1,185,1,189,1
  86 DATA 192,1,197,1,200,1,203,1,206,1,208,1,211,1,214,1
  87 DATA 216,1,218,1,220,1,222,1,224,1,226,1,227,1,229,1
  88 DATA 231,1,232,1,231,1,229,1,231,1,232,1,231,1,229,1
  89 DATA 206,1,203,1,200,1,197,1,192,1,208,1,206,1,203,1
  90 DATA 206,1,203,1,200,1,197,1,192,1,197,1,200,1,203,1
  91 DATA 206,1,203,1,200,1,197,1,192,1,208,1,206,1,203,1
  92 DATA 206,1,203,1,200,1,197,1,192,1,197,1,200,1,203,1
  93 DATA206,1,203,1,200,1,197,1,200,1,197,1,192,1,189,1
  94 DATA192,1,197,1,200,1,203,1,206,1,208,1,206,1,203,1
  95 DATA206,1,203,1,200,1,197,1,200,1,197,1,192,1,189,1
  96 DATA192,1,197,1,200,1,203,1,206,1,211,1,214,1,216,1
  97 DATA218,1,216,1,214,1,211,1,208,1,220,1,218,1,216,1
  98 DATA218,1,216,1,214,1,211,1,208,1,211,1,214,1,216,1
  99 DATA218,1,216,1,214,1,211,1,208,1,220,1,218,1,216,1
 100 DATA218,1,216,1,214,1,211,1,208,1,211,1,214,1,216,1
 101 DATA218,1,216,1,214,1,211,1,214,1,211,1,208,1,206,1
 102 DATA208,1,211,1,214,1,216,1,218,1,220,1,218,1,216,1
 103 DATA218,1,216,1,214,1,211,1,208,1,211,1,214,1,216,1
 104 DATA218,1,222,1,224,1,227,1,230,1,232,1,230,1,229,1
 105 DATA230,1,229,1,227,1,226,1,224,1,232,1,230,1,229,1
 106 DATA230,1,229,1,227,1,226,1,224,1,226,1,227,1,229,1
 107 DATA230,1,229,1,227,1,226,1,224,1,232,1,230,1,229,1
 108 DATA230,1,229,1,227,1,226,1,224,1,226,1,227,1,229,1
 109 DATA230,1,0,50,178,1,181,1,185,1,189,1,192,1,197,1
 110 DATA200,1,197,1,192,1,189,1,192,1,189,1,185,1,181,1
 111 DATA178,1,181,1,185,1,189,1,192,1,197,1,200,1,203,1
 112 DATA206,1,208,1,206,1,203,1,206,1,208,1,206,1,203,1
 113 DATA206,1,0,50,178,1,181,1,185,1,189,1,192,1,197,1
 114 DATA200,1,197,1,192,1,189,1,192,1,189,1,185,1,181,1
 115 DATA178,1,181,1,185,1,189,1,192,1,197,1,200,1,203,1
 116 DATA206,1,208,1,206,1,203,1,206,1,211,1,214,1,216,1
 117 DATA218,1,216,1,214,1,211,1,214,1,211,1,208,1,206,1
 118 DATA208,1,206,1,203,1,200,1,197,1,192,1,189,1,185,1
 119 DATA181,1,0,50,214,1,211,1,214,1,211,1,208,1,206,1
 120 DATA208,1,206,1,203,1,200,1,197,1,192,1,189,1,185,1
 121 DATA181,1,0,50,218,1,216,1,218,1,220,1,218,1,216,1
 122 DATA218,1,220,1,218,1,216,1,218,1,0,300
 123 DATA230,1,232,1,230,1,229,1,230,1,232,1,230,1,229,1
 124 DATA230,1,232,1,230,1,229,1,230,1,0,300
 125 DATA237,1,0,180,155,1,161,1,166,1,173,1,178,1,181,1
 126 DATA185,1,189,1,192,1,197,1,200,1,203,1,206,1,208,1
 127 DATA211,1,214,1,216,1,218,1,220,1,222,1,224,1,226,1
 128 DATA227,1,229,1,230,1,233,1,234,1,236,1,237,1,0,900
 129 DATA237,1,0,900,218,1,0,900
3000 IF H=1 THEN 8050
4000 POKE 36878,3
5000 POKE 36876,H
6000 FOR T=1 TO L:NEXT
7000 POKE 36878,15
8000 GOTO 10
8050 POKE36876,0
9000 END
```

Program 47: Bee

This program is the same as the number 46, but visual effects have been added. In this form the program requires a minimum of 8K. To use the default colors, simply delete line 3.

```
1 REM"FLIGHT OF THE BUMBLE-BEE"
2 PRINT"⊐"
3 POKE646,7:POKE36879,PEEK(36879)AND15OR0
4 PRINT"♥▶▶◙◙◙◙◙◙◙**FLIGHT OF THE**"
5 PRINT"♥▶▶◙◙◙◙◙◙◙◙***BUMBLE-BEE***"
6 PRINT"♥▶▶◙◙◙◙◙◙◙◙◙◙*******BY********"
7 PRINT"♥▶▶◙◙◙◙◙◙◙◙◙◙◙**MIKE HERRIOTT**"
8 PRINT"♥▶▶◙◙◙◙◙◙◙◙◙◙◙◙◙◙******* ╲●╱*******"
9 FORW=1TO1500:NEXT
10 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶"
11 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙,´"
12 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙`"
13 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙╱●╲"
14 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙╲●╱"
15 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙╱●╲"
16 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙╲●╱"
17 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙╱●╲"
18 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙╲●╱"
19 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙╱●╲"
20 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙╲●╱"
21 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙╱●╲"
22 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙╲●╱"
23 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙◙╱●╲"
24 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙╲●╱"
25 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙╱●╲"
26 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙◙╲●╱"
27 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙╱●╲"
28 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙◙╲●╱"
29 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙◙◙╱●╲"
30 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙◙╲●╱"
31 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙◙╱●╲"
32 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙╲●╱"
33 PRINT"⊐▶▶▶▶▶▶▶▶▶▶◙◙◙◙╱●╲"
34 PRINT"⊐▶▶▶▶▶▶▶▶▶▶◙◙◙╲●╱"
35 PRINT"⊐▶▶▶▶▶▶▶▶▶◙◙◙╱●╲"
36 PRINT"⊐▶▶▶▶▶▶▶▶▶◙◙◙◙╲●╱"
37 PRINT"⊐▶▶▶▶▶▶▶▶◙◙◙◙╱●╲"
38 PRINT"⊐▶▶▶▶▶▶▶▶◙◙◙◙◙╲●╱"
39 PRINT"⊐▶▶▶▶▶▶▶◙◙◙◙◙╱●╲"
40 PRINT"⊐▶▶▶▶▶▶◙◙◙◙◙╲●╱"
41 PRINT"⊐▶▶▶▶▶▶◙◙◙◙◙◙╱●╲"
42 FORV=1TO1500:NEXT
100 READ H
110 READ L
160 DATA 231,1,229,1,227,1,226,1
161 IFH=1THENGOTO9000
162 POKE36878,3
163 POKE36876,H
164 FORT=1TOL:NEXTT
165 PRINT"⊐▶▶▶▶◙◙◙◙◙◙◙╱●╲"
166 PRINT"⊐▶▶▶▶◙◙◙◙◙◙◙◙╲●╱"
169 POKE36878,15
170 GOTO100
175 DATA 227,1,226,1,224,1,222,1
180 DATA 224,1,222,1,220,1,218,1
```

```
190 DATA 216,1,214,1,211,1,209,1
200 DATA206,1,0,3000
215 DATA206,1,203,1,200,1,197,1
220 DATA 192,1,208,1,206,1,203,1
230 DATA 206,1,203,1,200,1,197,1
240 DATA 192,1,197,1,200,1,203,1
250 DATA 206,1,203,1,200,1,197,1
260 DATA 192,1,208,1,206,1,203,1
270 DATA 206,1,203,1,200,1,197,1
280 DATA 192,1,197,1,200,1,203,1
290 DATA 206,1,203,1,200,1,197,1
300 DATA 200,1,197,1,192,1,189,1
310 DATA 192,1,197,1,200,1,203,1
320 DATA 206,1,208,1,206,1,203,1
330 DATA 206,1,203,1,200,1,197,1
340 DATA 200,1,197,1,192,1,189,1
350 DATA 192,1,197,1,200,1,203,1
360 DATA 206,1,211,1,214,1,216,1
370 DATA 218,1,216,1,214,1,211,1
380 DATA 208,1,220,1,218,1,216,1
390 DATA 218,1,216,1,214,1,211,1
400 DATA 208,1,211,1,214,1,216,1
410 DATA 218,1,216,1,214,1,211,1
420 DATA 208,1,220,1,218,1,216,1
430 DATA 218,1,216,1,214,1,211,1
440 DATA 208,1,211,1,214,1,216,1
450 DATA 218,1,216,1,214,1,211,1
460 DATA 214,1,211,1,208,1,206,1
470 DATA 208,1,211,1,214,1,216,1
480 DATA 218,1,220,1,218,1,216,1
490 DATA 218,1,216,1,214,1,211,1
500 DATA 214,1,211,1,208,1,206,1
510 DATA 208,1,211,1,214,1,216,1
520 DATA 218,1,220,1,218,1,216,1,218,3
530 DATA 181,1,181,1,181,1,181,1,181,1,181,1,181,1,185,1
~~540 DATA 216,3,216,1,216,1,216,1,216,1,216,1,216,1,216,1~~
550 DATA 181,1,181,1,181,1,181,1,181,1,181,1,181,1,185,1
560 DATA 216,1,216,1,216,1,216,1,216,1,216,1,216,1
570 DATA 218,1,220,1,218,1,216,1,218,1,220,1,218,1,216,1
580 DATA 218,1,220,1,218,1,216,1,218,1,220,1,218,1,216,1
590 DATA 218,1,220,1,222,1,224,1,226,1,224,1,222,1,220,1
600 DATA218,1,220,1,222,1,224,1,226,1,224,1,222,1,220,1
610 DATA 218,3,200,1,200,1,200,1,200,1,200,1,200,1,200,
    1,203,1
620 DATA 226,1,226,1,226,1,226,1,226,1,226,1,226,1,227,1
630 DATA 200,1,200,1,200,1,200,1,200,1,200,1,200,1,203,1
640 DATA 226,1,226,1,226,1,226,1,226,1,226,1,226,1
650 DATA 227,1,229,1,227,1,226,1,227,1,229,1,227,1,226,1
660 DATA 227,1,229,1,227,1,226,1,227,1,229,1,227,1,226,1
670 DATA 227,1,229,1,231,1,232,1,233,1,232,1,231,1,229,1
680 DATA 227,1,229,1,231,1,232,1,233,1,232,1,231,1,229,1
690 DATA 227,1,226,1,224,1,222,1,220,1,229,1,227,1,226,1
700 DATA 227,1,226,1,224,1,222,1,220,1,222,1,224,1,226,1
710 DATA 227,1,226,1,224,1,222,1,224,1,222,1,220,1,218,1
720 DATA 220,1,222,1,224,1,226,1,224,1,226,1,227,1,229,1
730 DATA 231,1,229,1,227,1,226,1,227,1,226,1,224,1,222,1
740 DATA 224,1,222,1,220,1,218,1,216,1,214,1,211,1,208,1
750 DATA 206,1,208,1,206,1,203,1,206,1,208,1,206,1,203,1
760 DATA 206,1,208,1,206,1,203,1,206,1,208,1,206,1,203,1
```

```
 770 DATA 206,1,208,1,206,1,203,1,206,1,208,1,206.1,203,1
 780 DATA 206,1,208,1.206,1,203,1,206,1,208,1,206,1,203,1
 790 DATA 206,1,203,1,200,1,197,1,200,1,197,1,192,1,189,1
 800 DATA 192,1,189,1,185,1,181,1,178,1,173,1,166,1,161,1
 810 DATA 158,1,161,1,158,1,151,1,158,1,161,1,158,1,151,1
 820 DATA 158,1,161,1,158,1,151,1,158,1,161,1,158,1,151,1
 830 DATA 158,1,161,1,158,1,151,1,158,1,161,1,158,1,151,1
 840 DATA 158,1,161,1,158,1,151,1,158,1,161.1,158,1,151,1
 850 DATA 158,1,161,1,166,1,173,1,178,1,181,1,185,1,189,1
 860 DATA 192,1,197,1,200,1,203,1,206,1,208,1,211,1,214,1
 870 DATA 216,1,218,1,220,1,222,1,224,1,226,1,227,1,229,1
 880 DATA 231,1,232,1,231,1,229,1,231,1,232,1,231,1,229,1
 890 DATA 206.1,203,1,200,1,197,1,192,1,208,1,206,1,203,1
 900 DATA 206,1,203,1,200,1,197,1,192,1,197,1,200,1,203,1
 910 DATA 206,1,203,1,200,1,197,1,192,1,208,1,206,1,203,1
 920 DATA 206,1,203,1,200,1,197,1,192,1,197,1,200,1,203,1
 930 DATA206,1,203,1,200,1,197,1,200,1,197,1,192,1,189,1
 940 DATA192,1,197,1,200,1,203,1,206,1,208,1,206,1,203,1
 950 DATA206,1,203,1,200,1,197,1,200,1,197,1,192,1,189,1
 960 DATA192,1,197,1,200,1,203,1,206,1,211,1,214,1,216,1
 970 DATA218,1,216,1,214,1,211,1,208,1,220,1,218,1,216,1
 980 DATA218,1,216,1,214,1,211,1,208,1,211,1,214,1,216,1
 990 DATA218,1,216,1,214,1,211,1,208,1,220,1,218,1,216,1
1000 DATA218,1,216,1,214,1,211,1,208,1,211,1,214,1,216,1
1010 DATA218,1,216,1,214,1,211,1,214,1,211,1,208,1,206,1
1020 DATA208,1,211,1,214,1,216,1,218,1,220,1,218,1,216,1
1030 DATA218,1,216,1,214,1,211,1,208,1,211,1,214,1,216,1
1040 DATA218,1,222,1,224,1,227,1,230,1,232,1,230,1,229,1
1050 DATA230,1,229,1,227,1,226,1,224,1,232,1,230,1,229,1
1060 DATA230,1,229,1,227,1,226,1,224,1,226,1,227,1,229,1
1070 DATA230,1,229,1,227,1,226,1,224,1,232,1,230,1,229,1
1080 DATA230,1,229,1,227,1,226,1,224,1,226,1,227,1,229,1
1090 DATA230,1,0,50,178,1,181,1,185,1,189,1,192,1,197,1
1100 DATA200,1,197,1,192,1,189,1,192,1,189,1,185,1,181,1
1110 DATA178,1,181,1,185,1,189,1,192,1,197,1,200,1,203,1
1120 DATA206,1,208,1,206,1,203,1,206,1,208,1,206,1,203,1
1130 DATA206,1,0,50,178,1,181,1,185,1,189,1,192,1,197,1
1140 DATA200,1,197,1,192,1,189,1,192,1,189,1,185,1,181,1
1150 DATA178,1,181,1,185,1,189,1,192,1,197,1,200,1,203,1
1160 DATA206,1,208,1,206,1,203,1,206,1,211,1,214,1,216,1
1170 DATA218,1,216,1,214,1,211,1,214,1,211,1,208,1,206,1
1180 DATA208,1,206,1,203,1,200,1,197,1,192,1,189,1,185,1
1190 DATA181,1,0,50,214,1,211,1,214,1,211,1,208,1,206,1
1200 DATA208,1,206,1,203,1,200,1,197,1,192,1,189,1,185,1
1210 DATA181,1,0,50,218,1,216,1,218,1,220,1,218,1,216,1
1220 DATA218,1,220,1,218,1,216,1,218,1,0,300
1230 DATA230,1,232,1,230,1,229,1,230,1,232,1,230,1,229,1
1240 DATA230,1,232,1,230,1,229,1,230,1,0,300
1250 DATA237,1,0,180,155,1,161,1,166,1,173,1,178,1,181,1
1260 DATA185,1,189,1,192,1,197,1,200,1,203,1,206,1,208,1
1270 DATA211,1,214,1,216,1,218,1,220,1,222,1,224,1,226,1
1280 DATA227,1,229,1,230,1,233,1,234,1,236,1,237,1,0,900
1290 DATA237,1,0,900,218,1,0,900,1,1
9000 PRINT"♥▮▮▮▮▮▯▯▯▯▯▯/●\"
9001 PRINT"♥▮▮▮▮▮▯▯▯▯▯▯\●/"
9002 PRINT"♥▮▮▮▮▮▮▯▯▯▯▯/●\"
9003 PRINT"♥▮▮▮▮▮▮▮▯▯▯▯▯▯\●/"
9004 PRINT"♥▮▮▮▮▮▮▮▯▯▯▯▯/●\"
```

```
9005 PRINT"⊐▶▶▶▶▶▶▶▶◙◙◙◙◙ \●/"
9006 PRINT"⊐▶▶▶▶▶▶▶▶◙◙◙◙ /●\"
9007 PRINT"⊐▶▶▶▶▶▶▶▶▶◙◙◙◙ \●/"
9008 PRINT"⊐▶▶▶▶▶▶▶▶▶◙◙◙◙◙ /●\"
9009 PRINT"⊐▶▶▶▶▶▶▶▶▶◙◙◙◙◙◙ \●/"
9010 PRINT"⊐▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙◙◙ /●\"
9011 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙◙◙ \●/"
9012 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙◙◙◙ /●\"
9013 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙◙◙ \●/"
9014 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙◙ /●\"
9015 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙◙ \●/"
9016 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙ /●\"
9017 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙ \●/"
9018 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙◙ /●\"
9019 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙ \●/"
9020 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙ /●\"
9021 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙ \●/"
9022 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙ /●\"
9023 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙ \●/"
9024 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙ /●\"
9025 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙ \●/"
9026 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙◙ /●\"
9027 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙◙ \●/"
9028 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙◙ /●\"
9029 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙◙ \"
9030 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶◙◙ /"
9031 PRINT"⊐▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶▶I"
9040 GOTO9040
```

Program 48: Talk

This program demonstrates how you can get your computer to act as a "counselor" to those in need of a sympathetic listener.

```
 5 REM"TALK"
 8 PRINTCHR$(147)
10 PRINT"THIS PROGRAMME IS     DESIGNED TO HELP
   YOU SOLVE SOME OF     YOUR PROBLEMS"
20 FORD=1TO5000:NEXT
30 PRINTCHR$(147)
40 PRINTSPC(10);"MAYBE"
42 FORD=1TO2500:NEXT
45 PRINTCHR$(147)
50 PRINT"WHEN YOU SEE THE ?,   TYPE IN JUST ONE WORD
   OR SELECT FROM THE    LIST"
51 FORD=1TO4000:NEXT
52 PRINTCHR$(147)
55 GOSUB8000
60 PRINT"WHAT IS YOUR NAME?"
62 PRINT"WHAT IS YOUR NAME?"
65 INPUTY$
70 PRINT:PRINT
75 IF LEN(Y$)>=10THENPRINT"JUST YOUR FIRST NAME WILL
   DO":GOTO60
77 FORD=1TO2000:NEXT
80 PRINT"OK ";Y$:DIMX$(100)
85 PRINTA$(1)
90 INPUTX$(1)
```

```
 95 IFX$(1)>Z$(1)ORX$(1)<Z$(8)THEN2000
100 PRINTCHR$(147)
105 PRINTA$(4)
110 INPUTX$(2)
115 PRINTCHR$(147)
120 PRINTA$(18)
125 INPUTX$(3)
130 IFX(3)=Z$(14)ORX$(3)=Z$(13)THEN500
8000 DIMA$(50)
8001 A$(1)="WHAT IS YOUR PROBLEM?"
8002 A$(2)="IS IT SERIOUS?"
8003 A$(3)="WHY NOT?"
8004 A$(4)="HOW SO?"
8005 A$(5)="WHEN?"
8006 A$(6)="WHERE?"
8007 A$(7)="WHO?"
8008 A$(8)="WHAT FOR?"
8009 A$(9)="COULD IT BE THAT"
8010 A$(10)="IS THAT TRUE?"
8011 A$(11)="DO YOU REALLY THINK SO?"
9000 DIMZ$(50)
9001 Z$(1)="MOTHER"
9002 Z$(2)="FATHER"
9003 Z$(3)="BROTHER"
9004 Z$(4)="SISTER"
9005 Z$(5)="WIFE"
9006 Z$(6)="HUSBAND"
9007 Z$(7)="SON"
9008 Z$(8)="DAUGHTER"
9009 Z$(9)="BOYFRIEND"
9010 Z$(10)="FRIENDS"
9017 Z$(17)="NO"
9018 Z$(18)="YES"
9019 Z$(19)="MAYBE"
9020 Z$(20)="HATE"
9500 DIMH$(50)
9501 H$(1)="SICK"
9502 H$(2)="WELL"
9503 H$(3)="DYING"
9504 H$(4)="GETTING BETTER"
9505 H$(5)="DEAD"
9509 H$(9)="SHE LOVES ME"
9510 H$(10)="SHE LOVES ME NOT"
9998 RETURN
```

Index

Mastering the VIC-20®

by John Herriott

If you think your VIC-20 is simply a game machine for playing prepackaged tapes . . . *think again*! Yours is an amazingly powerful computing machine capable of all kinds of exciting home and business applications . . . and lots more creative game playing than you'd ever have thought possible.

All you need to start tapping those extraordinary capabilities of your micro is this handbook, and a little practice! The first thing you'll notice when you open this book is that the author's engaging style makes it more interesting than the average programming manual. Plus, you'll quickly discover that the author believes in introducing advanced features without making you wade through tons of background material . . . so *you* find yourself doing much more than you thought you could, almost immediately!

Plus, you'll find more than 50 sample programs that you can modify and adapt to your own needs . . . creating programs that use all the VIC-20's advanced graphics, color, and sound, find out about programs that solve complex programs for business or financial applications, even learn about using your micro as a word processor!

You find out how and why programs work . . . information that will enable you to find errors in other programs as well as your own. (Never again will you be left wondering whether you've made a typing error, or there's actually an error in the original program.)

All of the programs included are open-ended modules so that you can add to them or change them to suit your own needs—another unusual aspect of this programming manual. Other features include hints and tricks that make it easier to troubleshoot and debug your programs, how-to's for constructing modular programs and making flow charts, and a handy guide for translating programs written in any BASIC dialect to the BASIC used on your VIC-20!

A college professor, musician, computer consultant, columnist, and broadcaster, John Herriott is also an expert microcomputerist and programmer.

FPT > $10.25

ISBN 0-8306-1612-8

PRICES HIGHER IN CANADA

995-1083