USING & PROGRAMMING THE COMMODORE 64

INCLUDING READY-TO-RUN PROGRAMS

BY JOHN HERRIOTT

USING & PROGRAMMING THE
COMMODORE 64
INCLUDING READY-TO RUN PROGRAMS

No. 1712
$13.95

USING & PROGRAMMING THE COMMODORE 64

INCLUDING READY-TO RUN PROGRAMS

BY JOHN HERRIOTT

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

FIRST EDITION

SECOND PRINTING

Library of Congress Cataloging in Publication Data

Herriott, John.
Using & programming the Commodore 64, including ready-to-run programs.

Includes index.
1. Commodore 64 (Computer)—Programming.
I. Title. II. Title: Using and programming the Commodore 64, including ready-to-run programs.
QA76.8.C64H47 1984 001.64'2 83-18092
ISBN 0-8306-0721-9
ISBN 0-8306-1712-4 (pbk.)

Contents

Acknowledgments

An author is indebted to many people for assistance even when that assistance has not been obvious to the provider! For my part I wish to thank my family for being understanding and forbearing, and Dr. Harry Edstrom, Assistant Dean of the Faculty of Medicine, Memorial University of Newfoundland, for the advice regarding the Homed program.

Introduction

You have either bought or are considering buying a C-64 (Commodore 64) computer. If you are in the latter category I feel I must tell you that reading a book about the machine will not tell you much on its own. First see if one of your friends has a computer—any kind will do—and will demonstrate some of the things that it can do. Don't expect to be able to follow what happens unless you are already familiar with computers. Then, buy your computer together with a copy of this book (and buy one for your friend as a friendly gesture) and go to it. Within a couple of hours you will be programming your machine, and within a couple of weeks, you will be fully in command of it. As in doing anything else that is worthwhile, practice makes perfect, but somehow I do not think you will need very much urging to play and practice on your C-64!

The Commodore 64 has been on the market for just over a year (at the time of writing) and has already achieved great popularity in home and school and in some cases, in business. It is a powerful machine with considerable potential for peripheral expansion.

The C-64 has been offered in a variety of packages designed to provide a flexible working system. I have seen it in configurations which consist of the machine itself with a disk drive, or with a modem. Some *software* (ready-to-use programs) has been included in both cases.

A computer is a better buy than a mere game machine. There is a growing awareness of this fact on the part of the general public. The interest in computers over the past four or five years has been nothing short of astounding. To many people, it might seem 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, cooking utility and income tax records, bank statements, and charge accounts—and often being blamed for the errors!

When you do discover a mistake, it is often a difficult and lengthy process to get matters rectified. Although the C-64 is not as big as the machines the large corporations use, it can perform the same operations. This means that you can have access to the same information as the organization

you may have to challenge and in as short a space of time.

For the young, the computer allows entry into a new, exciting, and private world. The Commodore line of computers is extensively used in schools, and perhaps this is as good a reason as any for a C-64 to be a part of the home environment. The C-64 is also remarkably inexpensive for what it can do and is well supported in both peripherals and software. With a C-64 on the premises, I can assure you from personal experience that there will be no need whatsoever for any urging to do homework! In addition, the computer can be used as a teaching device for the study of all manner of subjects in a personal and self-paced fashion. This subject, Computer Assisted Instruction, is an area by itself and will be covered in another book of mine to be published in the near future by TAB BOOKS Inc.

As a device for the home the C-64 will enhance and add concrete experience to knowledge acquired about computers, developing a versatility on the part of computer science students. Using the C-64 at home can broaden students' knowledge of the functioning of computers, the differences between various types of computers, the differences between varieties of BASIC, the most common means of communicating with computers, and the means of tackling old problems with new tools.

An author must have an aim. He or she must also make certain assumptions when setting out to write a book. My assumption is that you wish to know how to control all aspects of the C-64 and to be armed with sufficient information and skill to enable you to go further on your own, exploring the virtues of the machine either through your own experimenting or with the aid of the countless superb articles that appear in magazines. My aim, quite simply, is to provide you with the means of realizing that wish. I have tried to be as user-friendly as possible, as user-friendly as the C-64 is, in fact. I hope I have done this in a fashion which you will find congenial.

I have 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. There are lots of programs in the book; all of them are short sequences teaching the fundamentals of each feature of the computer and providing plenty of ideas for you to incorporate into your masterpieces. Each program can serve as the beginning of a much larger program—as a springboard, as it were.

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 with which they are unfamiliar. Such courses are futile and serve only to provide some teacher with a salary. In order to make the course look worthwhile, it is padded with superfluous information. There is often very little in the way of hands-on experience provided. Even organizations hiring computer personnel seem to think that a person who has had a course is a better buy than someone who has worked on their own. One hopes that they will learn one day.

After working through this book you will be able to stand on your own two feet, read, understand, and make use of material you find in magazines and other books. This book, therefore, is a teaching tool, but it will only teach if you try each of the programs, learning how to modify them, introducing them into larger schemes, and juxtaposing them in new ways. 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 at all! In many cases the documentation is so scant, and the explanation of the workings so thin (or so technical) that you have no means of modifying the program. I hope that you will, as a result of using my offering, be able to modify most programs you come across, or better still, write your own. There is money to be made at that too!

No book can treat such a subject exhaustively. There are aspects of the operation of the C-64 that I have seen fit to omit. If you become consumed with a burning desire to know these things, my book has done its job.

There really is no time limit to the learning process. Do not be tempted to compare your progress through the book with other folk you know who own and use a C-64. Take your time, digress, dip here and there if you wish, leave the book alone while you explore some facet of the machine. Explore, explore and experiment, dwell as long or as little as you wish at any point, and soon you will be an accomplished computer user.

Chapter 1

The basic C-64 computer looks like half a typewriter. Don't worry, you have most of what you need! When you bought the machine, it is likely that you also bought a Datasette, an item that is like a tape recorder except that it won't record music. The box the computer came in also contains a heavy black box-like thing, some additional wiring, and small TV switchbox.

SETTING UP THE C-64

The heavy black box is the power supply. One end plugs into the wall socket and the other into your computer on the right side of it just next to the power switch. Refer to the diagrams near the beginning of the book that should have come with your C-64, the *Commodore 64 User's Guide,* if you are in doubt about specific locations.

There is also a narrow black lead. One end of this plugs into the small round socket on the back of the computer and the other plugs into the TV switchbox. To attach the TV switchbox to your television, you first detach the antenna leads from the VHF screws. Then attach the leads from the TV switchbox to the screws. If your TV has a switch that allows you to choose between an internal and an external antenna, set the switch to external. Now attach the TV antenna leads to the screws on the TV switchbox.

Turn the TV to channel 3, set the switch on the back of the computer to channel 3 (that is, towards the cartridge slot), and then turn on the computer. If you see nothing but ordinary TV shows, make sure you have set the TV switchbox to COMPUTER. If there is still nothing, switch to channel 4 on both the TV and the computer. When you've got things right, you will see a statement in light blue set on a darkish blue background to the effect that you are in touch with the C-64 computer. The picture may not be very clear, in which case fiddle with the tuning dial until you get a clear picture.

It is possible that you may have to adjust the horizontal control on your TV to bring the picture down a little. A small amount of time and effort spent in producing a good image will save a lot of

epithets later. The color, tint, brightness, and contrast controls should all be adjusted to give you a good clear picture. When all is ready, that is, the computer is displaying the word READY and the cursor is gently flashing underneath it, you can begin.

USING THE KEYBOARD

The keyboard is your means of communicating with the computer. The TV set is the computer's means of communicating with you by displaying what you type in, thus allowing you to verify your input and by showing you the results of the work the computer performs using the instructions you give it. The communication is in three dimensions, as it were: numbers and letters, collectively known as *alphanumeric characters* color, and sound. In addition to the alphanumeric characters, the C-64 can provide images of a variety of *graphic symbols* circles, hearts, straight and curved lines, and half-blocks. There are eight foundation colors. More can be obtained but they are shades of the fundamental eight. The sound is reproduced via your TV speaker. The C-64 has such a range of goodies that the sound can be modified in the fashion of a very good synthesizer.

The keyboard behaves in much the same way as a typewriter keyboard. Type the numbers 1 to 5 on the keyboard, and you will see them appear on the screen. Now press one or the other of the shift keys (it doesn't matter which you press; they both perform the same function), and then enter the same numbers again. This time you will see !"#$% appear. Now release the shift key, and type the letters Q,W,E,R, and T. On the screen will appear QWERT in uppercase characters.

If you press the shift key and type the same letters again, you will see a group of strange symbols. Look at the symbols carefully, and you will see that they correspond to those printed on the right of the front of each key.

To the left of the left-hand shift key you will find a key with the C= logo on it. Press this key instead of the shift and then type QWERT again. Now you have some more graphics symbols—those on the left-front of the keys.

Now press C= and the shift key at the same time. You have reached the lowercase letters. The keyboard now behaves exactly like that on a typewriter; pressing the shift key reaches the uppercase letters. This mode is called the *text mode*. Experiment with all this for a few minutes until you feel comfortable with it and can use the various features without too much thought.

By now the screen is getting full, so full in fact that the top line disappears and the new material comes in at the bottom of the screen. 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 on the screen is the light blue cursor winking on and off.

Just to make sure that you have all the colors (there is no possibility of you *not* having them as it happens, try the following: press the key marked CTR (upper left), and then press each of the numbers 1 through 8 in turn. The cursor will change color with each number. Leave the display at whatever color you have reached and type a few letters. If you find that the result is illegible, change the color again. Some combinations are rather difficult to read, but experimentation will show you which they are. There have been adverse comments about the color smear on the C-64. When the background color is discussed, you will see how to avoid some of the problem. Keep changing the color as many times as you like; even have each letter a different color if you wish. Then try pressing the CTRL key and the 9. Now the display is reversed: the color appears as the background for the characters you press. Press the CTRL key and the 10 to return to normal. When you tire of experimenting, clear the screen with the shift and CLR HOME keys.

Now press the C= key and the numbers 1 to 8 once more. Now you have access to more colors. Actually there is no point in going beyond number 7 although number 8 and the C= key will give you a dirty white. C= 7 returns you to the original color combination. Another way of getting back to the original combination is to press the key marked RUN/STOP on the left hand side and marked RESTORE on the right-hand side. Immediately the

screen returns to normal with just the word READY and the flashing cursor in the top left corner.

There are just a few more things you can experiment with before you actually write a program. At the right-hand edge of the keyboard, you will note a key with two horizontal arrows on it. It is right next to a key with two vertical arrows. These keys are cursor controls. Press the vertical arrow key. The cursor moves down. Now press the shift key and the same key. The arrow moves up. Now try the same thing with the horizontal key. First press it with the shift key and then without the shift key. Note what happens when you keep the cursor controls down. There is an automatic repeat on these keys.

Consider one more key: the INST DEL key. Put a few characters on the screen and then press the INST DEL key. The characters are erased. This key also has a repeat feature; very useful for deleting whole lines rapidly. Put some characters on the screen, and then move the cursor back to the left until it is in the middle of the group. Now press the shift and the INST DEL keys. A gap appears in the line. Correction of a misspelled word is quite easy with these controls. Missing letters can also be inserted.

A few moments spent playing with these controls will save a great deal of frustration later on, for it is inevitable that you will wish to alter programs—and not just to correct spelling and other mistakes.

CAPSULE REVIEW

All cables and leads that come with the C-64 are designed to fit into one spot and one spot only. It is wise to make all connections before plugging into the ac supply, checking that all is in order first.

Channel selection is made by means of a switch on the back of the C-64. (Users familiar with the VIC-20 will note that the RF modulator is inside the machine, not a separate box). Tint, color, contrast brightness and fine tuning controls must be used to produce the best image. Try to use the channel upon which there is no local TV broadcasting for best results.

In its normal state the computer prints uppercase letters on the screen. The shift keys produce punctuation and modifier signs from the numeral keys, but graphic symbols from the letter keys. The graphic symbols reached are those printed on the right-front of the keys.

The C= and shift keys pressed together produce lowercase letters. Pressing them once reaches this mode, known as text mode, and the normal use of the shift key thereafter reaches uppercase letters as with a normal typewriter. Pressing the C= and shift keys once more returns the computer to its normal mode.

The CLR HOME key operates the cursor and returns it to the top left corner of the screen without erasing material. Used in conjunction with the shift key, it erases screen material.

The C= key used with the letter keys reaches the left-most graphic symbols.

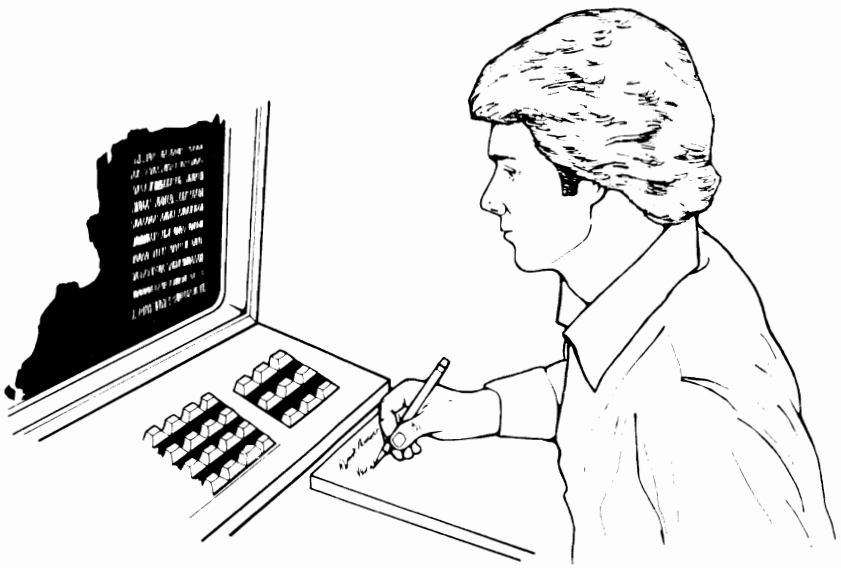
The six colors and black and white are reached by pressing the key CTRL and the numbers 1 to 8. Shades are reached by pressing the C= key and numbers 1 to 7.

The INST/DEL key removes material from the screen when used alone and when used in conjunction with the cursor and shift keys, can be used to insert material by creating as many spaces as desired.

The two cursor controls are found at the right bottom corner of the keyboard. The cursor will move down or right when the appropriate key is used alone; and up or left when used with the shift key and the appropriate key.

The cursor controls, the INST/DEL key and the space bar operate in the repeat mode when pressed and 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 according to precise plan. A few of the events and actions are not announced but must nonetheless be planned for lest the concert become a mess. Events such as raising the curtain, dimming the lights, and getting the performer to walk on stage at the right moment, sit down, stand up, take a bow, and walk off are all prepared even though they are taken for granted by stage crew and audience alike.

Computers are relatively dumb creatures. Everything they do must be prearranged. Some of the prearrangement has been carried out by the people who make the Commodore 64. That is good news for it means that your job is made a lot simpler. The machine will carry out your prearrangements to the letter, as long as you present them in the proper fashion.

In much the same way that different sections of a show are itemized and numbered on separate lines (we even call them numbers), each activity the computer indulges in is placed on a separate numbered line. Sometimes a number in a show will consist of a series of actions all listed on one line. It is possible to list more than one activity per line on the C-64 too, but you will have to wait a little before you do that. It has become the habit of programmers to number program lines at intervals of ten so that if a new line is needed, it can be inserted without your having to renumber all the rest of the lines.

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. They are important. Check to be sure that everything is exactly as printed here and then press the return key (lower right of the keyboard). Your

line is now installed firmly in the computer's memory, and the cursor is blinking on and off silently at the beginning of the next line.

Now type this line:

```
20 INPUT Z$        ($ is SHIFTed 4 Press the return key)
```

And now enter this final line:

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

Note that there is a gap after the word HELLO and before the final quotes. There is also a semicolon. You'll see the effects of writing the line this way when you run the program.

Now clear the screen (SHIFT and CLR HOME). Then type the word **RUN** and press the return key. Do what the computer tells you to do and then press the return key.

Your ultra-polite C-64 has not only accepted your name but has labeled it Z$ an then printed it in place of Z$ after the word **HELLO**. The word **READY** indicates that the computer is waiting for you to do it all over again or to do something else. If you type **RUN** once more the computer will indeed do it all over again. Just verify for yourself that Z$ can mean anything to the computer by entering another name. Wait a moment before calling in the family, though (unless they happen to be peeping over your shoulder—not a good plan for there is a great temptation for them to give advice!), for you need time to clean up that screen a little and learn something new: a time-saver!

Enter the following lines:

```
22 ?    (Always but always, press the return key after every line!)
23 ?
24 ?
```

This may not seem to be all that dramatic, three lines each with a question mark. Now type another new line:

```
5? [SHIFT/CLR HOME]
```

(What you will see is a reverse heart—don't worry!)

Now run the program.

The screen is cleared automatically and the **HELLO** and the name are further down the screen than before. The question marks had something to do with it. Type the word **LIST** and press the return key.

Your complete program is there, and the lines are in the correct order even though you typed them in the order 10,20,30,22,23,24,5 ! In addition, the question marks on lines 22, 23, and 24 have disappeared and the word PRINT is there instead. The question mark is a short-hand entry for the print command. The heart symbol, when placed in quotes, does the job of clearing the screen. Each print statement moves the text down the screen one line.

Now type 23 and 24 pressing the return key after each number. List the program again, and you will see that those lines are gone.

Now, using the cursor controls, bring the cursor up to line 22, move the cursor to the space after the question mark and type a colon. Then type another question mark and another colon, followed by a third question mark. Press the return key while the cursor is still on line 22. Then run the program. It behaves exactly as it did when lines 23 and 24 were in it. What you have done is to present three commands on one line. The question mark only works as a print command after a line number. Do not worry that all your question marks will come out as the word PRINT!

Now you are going to change some of the program so that the computer will perform new tricks. Add these lines.

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

```
80IFZ$="GRACE"THENPRINT"HI SIS"
```

(note the lack of spaces between words)

I have used the names of members of my family. You can use those of your family or of friends.

Run the program, and enter one of the names you have chosen. Somehow or other the computer is able to distinguish between each of the names, but only those in the program! Any other names are ignored. Now you can call in the family while you read the explanation.

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

If you wish you can enter the following lines:

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

The computer will put the lines in the right order, as you can verify if you list the program. You can use a bit of shorthand at this point. Clear the screen and then type L followed by shifted I. What you will see on the screen is the L followed by a small quarter-circle. Press the return key, and the program will be listed for you. Clearly, the L shifted I is shorthand for LIST. It's use saves a lot of time. Practice it!

Now run the program. It will now accept any name which is not stored as a string but it will simply say hello. The names are stored in the memory, and each time you enter something the computer checks to see IF it matches what it already "knows." IF the name does match THEN a specific response is made. IF not, THEN something else happens. It is as simple as that!

While the family is busy playing with your masterpiece, surreptitiously press the C= and shift keys. Be gratified by the oohs and ahs as the C-64 displays lowercase.

You will no doubt get a lot of questions like: "How do you get the colors?" "Will it talk?" "What's this button over here for?" Perhaps the wisest responses for the moment would be, in order: "Like this" (Press the CTRL key and one of the numbers), "No," and "Because it would be in the way over there."

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, run stop and restore keys.

SAVING AND RELOADING 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 fast forward (FFWD) button until you are sure there is magnetic tape in front of the head. You will, of course, have plugged the Datasette into the computer first, making sure that the plug is not upside down.

Before you can save a program, you must give it a name. While it is possible to save a program without a name, it is unwise because both the computer and you will have great difficulty finding it again! So, let us call this program Names. Type SAVE "NAMES" and press the return key.

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 two 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 footage counter reading for that program (it is a very good idea to make a note of that!), type the words LOAD"NAMES", and press the return key.

The computer responds with:

```
PRESS PLAY ON TAPE
```

You do that and then see:

```
OK
SEARCHING FOR NAMES
```

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

At risk of seeming to labor the point, let me urge you to make a note of the tape footage counter reading and the name of the program in the precise form in which it was saved. You cannot save a program with the name in the form N A M E S and then expect to load it by typing NAMES without gaps. It won't work!

There will be times when you want to be sure that a program has indeed been saved correctly. Rewind the tape to the correct tape footage reading and then enter VERIFY "NAMES" or whatever name you have given to your program. Press the play button and the C-64 will check the program on tape against the program in memory.

It is a very good plan to save portions of a long program from time to time. Do not leave a program in the middle to go do something else without saving what you have done. There could be a storm or malfunction that causes a power failure, and you will lose all your work that has not been saved. Alternatively, someone may come and play with the computer, not knowing that you are in the middle of a program. You are likely to have a mess on your hands!

DOING CALCULATONS

Now that you have saved your small Names program, with a view to future modification, it is time to do something else. Type the word NEW (and press the return key that's the last time I shall print that: take it for granted from now on). The program is now erased from the computer's memory.

Now, without putting in line numbers, enter the following:

```
X=10:Y=5:?X+Y
```

The C-64 responds instantly with the result, 15.

Now do the same thing on individual lines—still without line numbers but press the return key after each entry.

```
X=10
Y=5
?X*Y
```

(* is found just to the left of the restore key and is understood by computers to mean multiplication).

The C-64 responds with 50. Now enter ?X/Y and you will see the result, 2.

You are using the machine in what is known as *direct* or *calculator* mode. You can change the results merely by typing in new values for X and Y using the equals sign. It would be more efficient to use a program designed to accept new values for X and Y than to constantly enter new values in the direct mode. The format is similar to the one used for the Names program. You just use numbers instead.

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

```
10  INPUT "VALUE FOR X PLEASE";X
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 C-64 prints the result. Before you can use the program (such as it is) 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 waits patiently for new members.

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 time. Note that there is no need to put in the question mark, because the input statement does that automatically.

You can shorten the math program even more in the following way:

```
10  INPUT" VALUES FOR X,Y AND Z"; X,Y,Z
```

Remove line 20 by typing 20 and then pressing the return key. This time the computer will keep on displaying question marks until you have entered all of the values requested. Try your hand at doing something with the Z value that you have introduced; perhaps using +, /, or * in various ways. If you have a problem just remove the Z with the cursor and delete keys.

So far you can only alter the type of function you wish performed upon the numbers by changing a line. It is rather a nuisance to have to list the program and then operate the cursor controls just to get a different function.

This is where that wonderful device you are playing with shows its power, for all you need to do is write a program that allows you to select the numbers you wish to work upon and the kind of work you wish to have done.

The following is one way to do this: first leave line 10 as it is, you can input more digits later; then type the following lines:

```
30  INPUT "+,−,*,/";A$
40  IF A$="+" THEN 100
50  IF A$="−"THEN200
60  IFA$="*"THEN300
70IFA$="/"THEN400
100?X+Y
200  ?X−Y
300?X*Y
400?X/Y
410  END
```

There is just one more thing you must do before you run the program. As you perfect the program you will learn a little trick about line writing that will save you quite a bit of time.

If you were to run the program as it is you would get back four results from the computer. You see, after the computer has printed the result on line 100, it will continue to line 200 and print that result, and then go to 300 and print that result, and so on. You don't exactly want that. You have line 410, which tells the computer to stop. All you need to do is duplicate that instruction at line 110,210, and 310. You could easily enter those lines by typing in the appropriate line numbers and lines, but the trick I am to show you will be extraordinarily useful in the future when you have complex lines to duplicate.

Bring the cursor up to the 4 in line 410 and then type a 1 instead. Press the return key. Then bring the cursor up to the 1 in what is now 110 and change it to a 2. Again press the return key. Do it all over again except type 3 and press the return key. Now list the program. Hey Presto! The entire program is there including the new lines you needed, not only the line numbers but also the end instruction. Now you can run the program.

Lines 40 to 70 might look a little odd. You will have little difficulty recalling that the IF-THEN statement requires some action resulting from its use. The action in this case is to move to a particular line number. The full expression in this case should be IF-THEN-GOTO, but the GOTO can be dropped in the variety of BASIC used by the C-64. It saves memory and typing time!

It would be nice to be able to go back to the

beginning of the program without having to type RUN each time, wouldn't it? You know how to do this by changing the end statements into GOTO 10 or even GOTO 5, and then performing a "[SHIFT/CLR HOME]"at line 5. Simply type 110 GOTO 5 and do the same for each of the other end lines. Now the program recycles itself without help from you.

REPEATING A PROCESS

It often happens that you need a process to take place more than once. Sometimes you need to use the process 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 that form part of the BASIC language. Rather 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. Properly speaking a loop is not a routine or a subroutine, but it is jolly useful all the same and will serve as a reasonable introduction to the concept of routine. The vital ingredient in a loop is the for-next cluster.

The computer is told to perform some operation or other *for* so many times. Each time it performs the operation, it checks to see if it has done so the required number of times. If it hasn't, it performs, the *next* time. It's rather like those games where a player is told to take three turns in a row. He does so while the other players keep tabs on him. If he tries to take too many turns, he is told to stop. For three turns, he throws the dice; or, in BASIC parlance: FOR turns=3 throw dice:NEXT.

Here is a simple example:

```
10  FOR I=TO10 (in other words, ten turns
    called I)
20  ?I
30  NEXT
```

That's all the program consists of, but it does a surprising amount of work. Try changing the values of the two numbers in line 10; try, for example, changing the 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 by using the cursor. Try changing each of the numbers, but don't make the value of I too large or else you will not see what is happening.

The C-64 can count backwards very rapidly too. Change line 10 to read:

10 FORI=40 TO1STEP–2 (That's minus 2, by the way . . and you can leave out the spaces).

Again experiment with the numbers.

When you have grown tired of playing with this 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  FOR I=ATOB
40  ?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.

When folk are learning something new, they not only catch on to the obvious information that is being presented but also pick up an ancilliary points that have not been thoroughly discussed. One of these points I have indeed touched upon but not dealt with. The point concerns what is known as *crunching*. As mentioned, multiple statements can be made on single lines on the C-64. To confirm your suspicions about the format of this, the rule to follow is: statement, colon, statement, colon, statement; for as many statements you fit on a line.

You will soon discover that there is a limit to the number of characters that can follow a line number. The limit is 80, or two screen lines. Any more than that and the screen will give you an error message. In such a case, merely remove the extra material and place it in a new line.

Crunching lines saves memory, is easier to type, is often easier to read, and allows the program to run faster. Most of the programs in this book are printed in *open* form in order to provide you with the greater flexibility in the early stages. You are free to crunch the lines.

To return to the study of the loop program: Would it work with a list of names? Sure it would. Let's work up a new program for this in small stages.

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

This very short program will run very sweetly by itself. However, you are limited to ten names unless you alter the listing. The necessary alteration involves the use of a new BASIC word; DIM, which stands for dimension.

A whole series, pile, or collection of things that are nearly the same is called an *array*. Names are the same, speaking collectively. They are labels for things or people. So a collection of names would be called an array of names. Before you can use an array you must dimension it, in other words, you must declare how many names you are going to use. The format is DIM(X) where X is the quantity of names. Let's try a simple DIM.

```
10  DIM A$(30)
20  INPUT"HOW MANY NAMES";N
30  FOR I=1TON:INPUT A$(I):NEXT I
    (That's 1 TO N, by the way . . . not one ton!)
40  FOR K=1TON:PRINT A$(K):NEXT K
```

Line ten allows a limit of 30 names. It could be more or less of course, but thirty will do for the time being. Line 20 asks how many names you wish to enter on this run of the program. You can change it next time around if you wish. Line 30 is the for-next loop that says: keep counting the value of I from 1 to the number of names you wish to enter; store each name as A$(1), A$(2), A$(3) and so on up to the limit set by N. Line 40 says to do the same as line 30 but to print out the names instead of saving them.

If you want to enter and print more than 30 names, you must change line 10. It is possible to write a program that will allow you to dimension your array each time you run the program, but before you do that, I intend to introduce a *bug* into the program. Bugs are inevitable! Every programmer will find them, produce them, and with a bit of thought, get rid of them. Bugs can stop your program cold! They must be exterminated. You cannot deal with bugs in other programs until you have examined a few programs and solved the problems.

Change line 10 to read DIMA$(N). It would appear that the DIM statement is now under the control of the number you enter on line 20. Run the program. The computer shows the request quite nicely, and you can indeed enter the first name. No sooner do you press return, however, than the harrowing message

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

appears on the screen. List the program. Now you can see what the problem is. You have used the variable N as a *subscript* (that is what the number in parenthesis in a DIM statement is called) before you have assigned a value to it—an action that is not allowed!

To correct the situation, use the cursor to change the line number for line 10 so that it becomes line 25. Be sure to erase line 10 by typing 10 and pressing lthe return key before you run the program. Now everything runs well.

"All right," you say, "I must put in the names in alphabetical order in order for them to appear in alphabetical order". In this program you certainly do have to do that. But you have a computer and computers are delightful creatures. You can program them to do almost anything—even sort things into the correct order. Sorting things into order is something that the computer does so well that it deserves a chapter all to itself.

CAPSULE REVIEW

Line numbers are usually assigned using intervals of ten, at least in the early stages of a program. Each line consists of one or more *statements* or directions to the computer. Each program line can contain as many as 80 characters (2 screen lines). Thus more than one statement can be placed in one line, as long as the statements are separated by colons.

Whatever you place between quotation marks will appear as text or output to the screen. Statements or commands will not be displayed when the program is run.

Strings are denoted by a letter (or two or three) and a $ sign.

The input statement allows you to take part in the program activity. Whatever you type on the keyboard in response to prompt an input will appear on the screen and be used by the program, except under error circumstances.

The print statement can be used alone to generate blank lines on the screen so that presentation of material is neater.

Statements can be entered in shorthand form: PRINT can be represented merely by the ? mark and LIST by L and SHIFT I.

User input can be evaluated by the program. The simplest method is to use the if-then construction followed by some other statement indicating some form of action.

Programs are saved to tape by entering the word SAVE followed by the program name in quotation marks. The computer prompts you with the appropriate instructions and indicates when the process is complete.

A program is transferred from tape to the computer by entering the command LOAD; followed by the program name in quotation marks. Again, the computer prompts you with the appropriate instructions for operation of the tape recorder and indicates when the process is complete by displaying the word READY.

It is wise to label programs with a short name, being careful to avoid odd spaces and punctuation marks. Make a note of the program name in the exact form under which it was saved and the tape footage counter reading on both the cassette and its container. The C-64 will search through an entire tape for the program you wish to load, but the process can be irksome with a 90 minute tape.

Programs can be erased from the computer's memory by entering the BASIC word NEW or by switching off the computer. The word NEW cannot be used within the body of a program or else the entire program will erase itself.

The C-64 can be used in what is known as the *direct mode*. In this mode, no line numbers are necessary. Multiple instructions can be entered as one line and will be executed immediately. Only simple programs can be entered and used in this fashion.

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

The computer can be programmed to evaluate a term and act accordingly. The if-then construction is one means of causing an evaluation and consequent action to be performed.

The computer can be caused to repeat a given activity any number of times, even for ever. This condition of perpetuity is, however, undesirable in most cases and is due to an error in the program. The only solution is to press the run/stop and restore keys or, if things are in a very bad state, to switch the computer off and start again. This latter procedure destroys utterly any program that has been so far entered and thus all the work that has been done so far. It is therefore a good plan to save sections of an incomplete program from time to time.

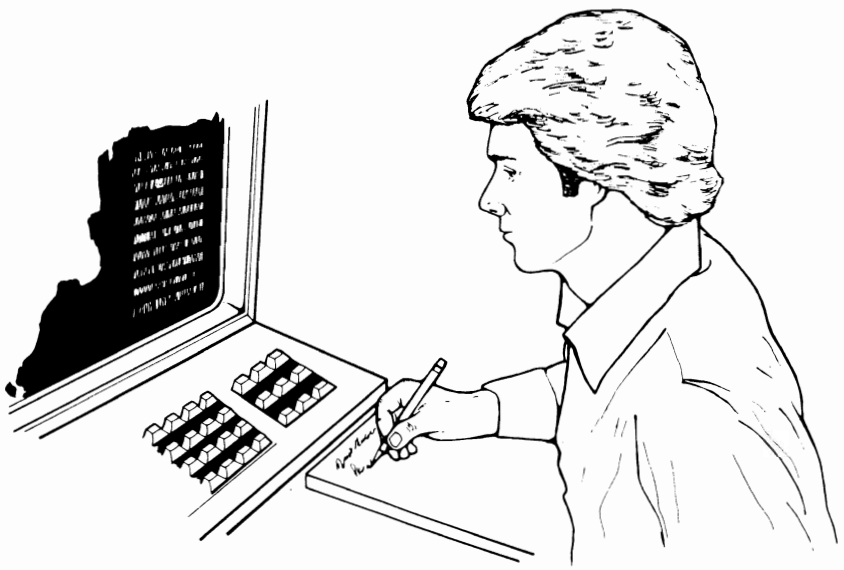
For-next loops allow for the repetition of action under the control of the programmer. The word FOR is followed by a variable label that is then given a series of values from one numerical point to another. Both points can be specified at that moment or can be subject to user input. They can also be defined in another area of the program where

some other event that produces a desired value has taken place.

The counting in a for-next loop can be either consecutive or by steps of any numerical size. The steps are defined by the step statement. The count can also be made in reverse, or decrementally. The format for decrementing is.

An array can be composed of FORI=N TOZSTEP–X strings or it can be a numerical array. An array of either type can be assigned to a variable name and can use the same variable name repeatedly because each item is assigned a subscript 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$(X) where X indicates the number of items that can be called A$ (or A if the DIM statement deals with numerical values). The number X can either be declared within the statement itself or called from elsewhere in the program. In the latter case, the value of X must still be declared before the DIM statement is reached, or else an error condition will exist.

Chapter 3



I have had to make certain decisions about the order in which the material of this book is to be presented. There is, however, no really fixed hierarchical structure to what activities you indulge in from now on. It is likely that this is not the only text on computers that you will have read and you may indeed already have some knowledge of BASIC from work with another machine.

You could be anxious to get on and explore other features of the C-64 such as color and sound. You are quite free to do so, returning, if you need to, to this chapter on sorting techniques and a variety of ancilliary matters.

SORTING ROUTINES

Type in the following program (or load it from the disk or tape if you have it):

```
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 NEXT
130 FORI=0TON+1:PRINTA$(I):NEXT
```

When you run this program for the first time, indicate how many names you wish to use. The computer will not allow you to enter more than this number. Then, for test purposes, put in names that begin with the same initial. Don't put too many in so that you avoid typing errors. See how nicely the program sorts the names for you?

Before going on to the explanation, let's tidy up this program 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 key twice. Now enter ? and a colon and press re-

turn. Bring the cursor below the word READY and then run the program once more.

Line 5 might seem to be the odd one here. It performs the same function as ? "SHIFT/CLR HOME". Each character, indeed each activity the computer indulges in is represented by a code. You will find a list of them in the user's manual that came with the machine. CHR$(147) is the one that clears the screen. I find myself using that more than the heart-in-quotes version. You could put another in at line 125 to get rid of the names you entered at first, although you might wish to demonstrate that the computer has indeed sorted the names into alphabetical order.

The first five lines in this program are fairly obvious and are little more than you have seen in previous programs. Line 50 has a curious feature. Try changing the value of TAB inside the parenthesis and see what happens. Try 50 to start with. TAB, far from having anything to do with either the publisher of this book or diet drinks, works just like the tabulator setting on a typewriter. If I press the tab key on my typewriter, the carriage moves across very quickly to a point that I have prearranged. The tab function on the computer does precisely the same thing, except on the screen. Now try 250. The maximum number you can use is 255. Now try adding TAB to line 130 by inserting it after the print statement: Just put it in without punctuation, like this:

```
PRINTTAB (45)A$(I):
```

Now the names are presented neatly near the center of the screen instead of crouching up against the edge.

Lines 60 to 120 do the actual sorting beginning with a loop that has one or two escape hatches. These escape points are there to be used when certain conditions are met.

Everything a computer does is by number. Each character has a value known as its ASCII (American Standard Code for Information Interchange) value. The CHR$ function is one way of using this value. The letter A is obviously assigned a lower value than the letter J, and so the computer merely sorts through what it thinks are numbers. I told you computers are dumb!

Line 70 says that if the second name you put in is of greater or equal value compared to the value of the first name, the program should go to line 120.

Line 120 tells the computer to go back to line 60 and go through the process over again with the next pair of names. If the second name is not of greater or equal value to the first, a new variable called B$, which is equal to the second name, is set up. Then there is a little swapping around that serves to keep that name aside until it is needed, in much the same way you would do it if you were sorting a set of filing cards, holding the Z's between two fingers and the T's between two others. When all the sorting has been done the computer prints the whole thing out for you.

If you have the patience, enter 50 or so names, or even make use of the program to sort your record collection (increase the value of the DIM statement, however). You can see that the computer takes a long time to do all that.

Actually, it is not the computer that is taking the time . . . it is the program, or rather the algorithm being used in the program, which takes all the time. There are faster methods than this one, which is called a Bubble Sort by the way, and I have printed a group of them in Appendix A.

If you would like to follow what is going on while the machine works, then enter the following lines:

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

This will print out the result of each shift back and forth between lines 60 and 120.

SUBROUTINES

Some time ago, in the second chapter in fact, I mentioned the word *subroutine*. I am going to deal further with the subroutine now. The for-next construction is a closed loop. The action will continue to recycle for the number of times you have specified, even when the action might include a little shuffle in the middle as in the name-sorting program.

A proper subroutine is a little more controllable than that for it is a section of the program that is brought into play only for those occasions when it is needed. You do not have to use it a fixed number of times, but you can call it at will.

Let's return to our first program, the one in which you put in your name and some sort of response comes out. This program will announce whether the name entered is a family name or not. This is not a tremendously useful program but one that will provide some fun at a party and teach you something new at the same time.

```
5 PRINTCHR$(147)
10 INPUT"WHAT IS YOUR NAME";Z$
20 PRINT"THANKYOU ";Z$
30 IFZ$="JOHN"THENGOSUB500
40 IFZ$="KAREN"THENGOSUB600
50 IFZ$="CAITLIN"THENGOSUB700
60 PRINT"DO I KNOW YOU ";Z$;"?"
100 STOP
500 PRINT"YOU ARE FAMILY--GET
    BACK TO THE TYPEWRITER"
510 RETURN
600 PRINT"SING ME A SONG THERE'S
    A GOOD GIRL!!! ";Z$
610 RETURN
700 "I THINK YOU'RE SUPER!! ":Z$
710 RETURN
```

This trite little program demonstrates the power of the GOSUB command. This command sends the computer hunting for the specified line, causes it to perform whatever activity is indicated there and then returns it to the line that follows the GOSUB command. Do not confuse the RETURN associated with a GOSUB with the key marked RETURN. You type the word in your program: it is no good just to press the key!

I do not recommend saving this program. Perhaps you might like to try something else a little more interesting?

A DEMONSTRATION PROGRAM

The program in Fig. 3-1 is fairly long. You have the choice of typing it in full, examining the program carefully to see what it does, or you could load it directly from the tape or disk. It is called Demo.

The machine introduces itself and calls for responses from the user. A menu of possibilities is listed and the user selects from these by entering the appropriate number. This is a fairly common way of selecting activities. You will find menus that call different sections by name or by means of coded abbreviations. When you write a more complex program than this, you might wish to try variations in menu style.

The new statement is ON-GOTO. Quite simply, you can avoid a great slew of if-then statements by using ON-GOTO. The operation is quite clear from the program . . . but I'll explain it anyway! On receiving a specific digit as input the computer checks to see which of the new program lines it is to go to. If the input is 1, the computer will go to the first program line listed; If the input is a 2, it will go to the second line listed, and so on.

Note the stop instruction which prevents the program from running on to the next part, thus making nonsense of the program.

Figure 3-2 shows one part of the program. The complete program is printed in Appendix F, program 1. One curious use of the FOR-NEXT loop might be noticed:

```
4060 FORL=1TO10000
4070 NEXTL.
```

At first sight it might seem as if a good deal of the program is missing. All this sequence does is to hold the text on the screen for a certain period of time without anything else appearing on the screen. It gives the user time to read the material. You will find programs that ask the user to press the space bar to continue, thus allowing for varying rates of reading. I find it a nuisance to have to keep on pressing the bar. It is easier to take a guess at how long a user will take to read a given amount of material. There is a rule that you can filch from the days of silent movies it says to time the reading of the text slowly and then add 15%. It works.

Note the choices the user has with regard to doing something over or going on to something new in lines 4220-4255. Some of the features you may not understand now will become clear, either later

```
10  "Hello. My name is Commo. 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  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
```

Fig. 3-1. The beginning of a demonstration program.

on when they are explained or as you run the program.

As I remarked in the introduction to this book, my aim is for you to stand on your own feet. It is likely that you will have texts on BASIC that you will have worked. Your knowledge of what is going on in this program is likely to be greater in such a case than I have presumed. The program is not crunched. Should you wish to expand the program to include other clever features at some point, it might be a good plan to convert it to multistatement form, but be careful of GOTOs as the program flow can become quite confusing.

CAPSULE REVIEW

An array may be regarded as a sequence of things that have a label or variable name in common. The word *computer* is a label that can be applied to a variety of devices that differ in appearance and in ability. The word computer is a natural language variable, and the set of computers that is to be found on the market is an array. A BASIC variable can be one, two, or three letters (or even more except that the computer will ignore too many characters and just count the first two) or a combination of a letter and a numeral. Thus A, AA, AB, A1, and A2 are all valid variable names.

CBM BASIC recognizes 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; and string variable, denoted by one or two characters followed by the $ sign. The first two variable types deal only with digits; the third can deal with

text input and output whether this includes digits or not.

CBM BASIC allows arrays of up to eleven elements (Ø to 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(Ø.) An array that requires more than eleven elements must be declared prior to use by means of the dimension

```
4000  ?CHR$(147)
4020  ?"YOU MAY ENTER UP TO 20 NAMES."
4030  ?:?"MEMBERS OF YOUR FAMILY; FRIENDS OR ANY OTHER
      NAMES."
4040  ?:?"WAIT FOR THE ? BEFORE TYPING, AND PRESS THE RETURN
      KEY AFTER EACH NAME."
4050  ?"YOU WILL HAVE TO TELL THE COMPUTER HOW MANY NAMES."
4060  FOR L=1 TO 10000
4070  NEXT L
4080  ?:?:? "PRESS ANY KEY TO CONTINUE"
4085  GET A$:IF A$="" THEN 4085
4090  ?CHR$(147)
4095  ?"GOOD."
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
```

Fig. 3-2. A possible subroutine for a demonstration program.

(DIM) statement. An array may not be redimensioned during the course of a program.

The DIM statement states the number of elements in an array in the form DIM(20). 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 and spaces in any single string does not exceed 255.

The computer can actually only recognize numbers and those only in the form of zeros and ones (binary notation). Programming a computer by direct manipulation of this binary code is a long and arduous process. Mercifully the internal brain of the computer can do most of the work of translating from *BASIC* into binary. All you have to do is figure out the appropriate pieces of BASIC to use in order to produce a workable program.

BASIC, Beginners All purpose Symbolic Instruction Code has sets of instructions that stand for many activities indulged in by the computer in much the same way that natural language has words, like *run* or *play the piano*, which cover a whole range of complex muscle movements.

Every character that can be accessed from the keyboard and each activity that can be controlled from the keyboard, such as carriage return, has a numerical value called its ASCII value. These values can be used to enable the computer to sort a variety of items into various hierarchical orders.

Subroutines are rather like miniature programs. 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 computer to go to one specific spot within the program and causing it to remain there until a new and separate command is given. The GOSUB-RETURN cluster requires that the main body of the program be rejoined immediately following the spot from which the jump was made. You cannot cause the computer to return to another spot.

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 *many* driver 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.

PRINT CHR$(n) is a function which allows for the generation of activity on the screen. This activity may be in the form of the presentation of a specific character or in the form of a change or activity. ?CHR$(156), for example does nothing more than to change the character color to purple. ?CHR$(147) causes the screen to be cleared. It is possible to enter anything in the form of CHR$ functions, yet that would be tedious. CHR$ should be reserved for those occasions when it would otherwise be difficult or impossible to include commands in quotes. Changes from upper to lowercase, the presentation of material in different colors, carriage returns, reverse on and off, and all cursor control can be operated from within a program by use of the CHR$ function.

The FOR-NEXT loop can be used to control the timing of the appearance of material on the screen so that it appears in a clear form and at a nonthreatening rate. The form of such a loop can be

FORI=1TO N:NEXT

where N is a number proportional to the amount of time you wish the material to remain on the screen undisturbed.

The input statement allows the user to enter material. The response made by the user will depend upon the type of variable. INPUT A, for example, will cause the computer to wait until a numerical has been entered. INPUT A$ requires a string.

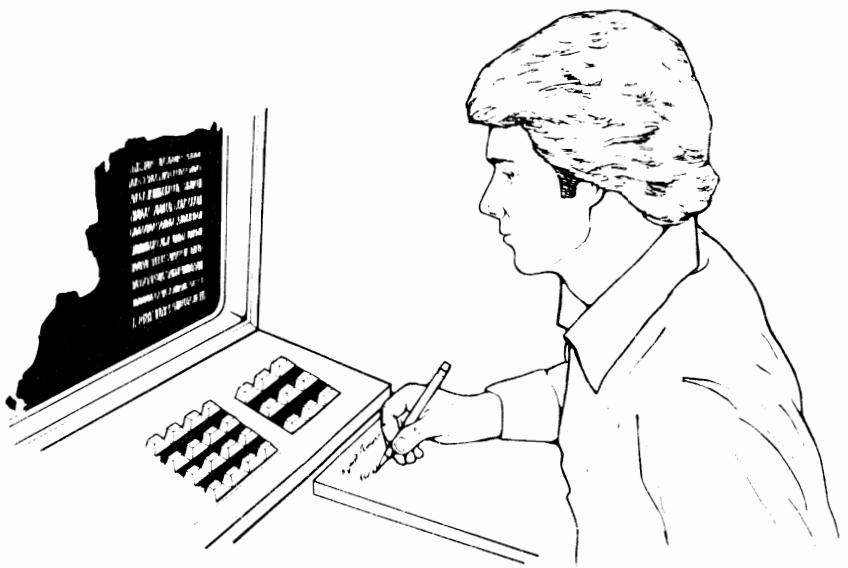
However, a five or six digit number (or any number of digits) can also be a string and so it is possible to respond to the question WHAT IS YOUR NAME? with 1234567, in which case the user must tolerate being so labeled for the rest of the program! It is possible to use this feature to provide security for the identities of users in such cases as exam marks and report cards.

A program uses a memory space roughly in proportion to the amount of characters used, although DIM statement have a nasty habit of using up memory just by being there!

The C-64 has a feature which conserves memory. The command PRINT would seem to use up five bytes. However, the C-64 tokenizes the command into two bytes, thus saving lots of memory in large programs.

You can check on the amount of memory left at any point in program production by typing (in direct mode) ?FRE(0). You may be very surprised to find that a negative number appears, perhaps −26472. The C-64 presents negative quantities when the memory remaining exceeds 32K.

Chapter 4



Moving things around the screen and using the many colors available on the C-64 are very jolly things to do. On first examination, the activity might seem to be a little complicated, but it isn't all that bad.

MOVING AN OBJECT ON-SCREEN

You have to remember that in order to show anything in a particular spot on the screen, you have to make two pokes. The first poke is to the screen address and the second to the color address. Screen addresses run from 1024, which is the top left corner, to 2023 which is the bottom right corner. Color addresses run from 55296 to 56296 and each has a one-to-one correspondence with the screen address.

You poke the value of a character into the screen address. These values are the ones found in Appendix E of your C-64 User's Guide, not the ones found in Appendix F. If you poke 1024 with value 81 (a ball, more or less) a filled-in circle will be placed at the top left corner. The problem is that you cannot see it until you also poke a color to the color address which corresponds to that spot.

In direct mode, therefore, type POKE 1024, 81:POKE 55296,3 and you will see a cyan circle in the top left corner. Change the value of the color by bringing up the cursor, moving it along to the number after the comma in the color address, and typing the new number over the top. When you press the return key, you will see the color of the circle change.

Now you can have some fun! Still in direct mode enter the following:

```
FORI=1TO40:POKE1424+I,81:POKE55696
+I,8:NEXT
```

When you press the return key, the small ball will zip across the screen leaving a row of itself behind. Change the value poked to the color address (the digit after the comma) and do it all over again. The ball-row will change color.

Now bring the cursor up to the line again and

create a gap of 14 spaces before the NEXT. Insert the following: POKE1424+I,32:

This time the ball will zip across the screen without leaving a trace of itself. Let's put that in the form of a proper program with line numbers and all.

```
1 REM BALLR-L
5 PRINTCHR$(147)
10 FORI=1TO20:POKE1024+I*38,81:
   POKE55296+I*38,8:POKE1024+I*38,
   32
100 NEXT
110 GOTO5
```

So far the ball merely travels straight across the screen. Making it move diagonally is fairly easy and involves putting in a small computation (which computers are good at, so they say). If you multiply I by a given amount that is less than the numbers of positions available on a screen line, the ball will travel in one direction. What happens when you use a number greater than 40? Try it and see.

It is a nuisance to have to keep on changing your program listing each time you want to see the effect of a new value. Let's work up a program that makes alterations subject to user input, for after all, this is the way good programs that deal with numbers or text work. One or two juicy bits in the form of values to be poked into the border and background color addresses will be added to the program. The speed with which the ball travels will be subject to a step instruction but do not make it too large until you have seen what small values can do.

```
1 REM BALLDIAG
4 PRINTCHR$(147)
5 PRINT"COLORS A & B: STEP SIZE C"
6 PRINT"A IS BORDER; B IS
  BACKGROUND"
7 INPUTA,B
8 INPUTC
9 PRINTCHR$(147)
10 POKE53280,A:POKE53281,B
12 FORX=1TO10
15 FORI=1024TO2023STEPC
20 POKEI,81
30 FORD=1TO42:NEXT
40 POKEI,32
50 NEXT
60 NEXT
70 GOTO1
```

You can play with the values here for a long time, gradually becoming familiar with the colors as well as the behavior of the ball. You could change the color of the ball too by adding POKE 646, and some value. You could add the value as a user input. Label it D and add D (plus the user prompt on line 6) to line 8.

The next program is even fancier. It is a variation on that found on page 65 of the *Commodore 64 User's Guide*. It adds an opportunity for you to decide which colors you are going to use. Note how you set values for X and Y at the start of the program and then increment them during the course of the program. You also set values for DX and YX. X and Y refer to horizontal and vertical positions of the ball; DX and DY are used to cause horizontal and vertical movement of the ball.

```
1 REM BALLC
2 PRINTCHR$(147)
5 INPUT"BORDER,BACKGROUND &
  CHARACTER COLORS";A,B,C
10 PRINTCHR$(147):POKE53280,A:
   POKE53281,B:POKE646,C
20 X=1:Y=1
30 DX=1:DY=1
40 POKE1024+X+40*Y,81
50 FORD=1TO5:NEXT
60 POKE1024+X+40*Y,32
70 X=X+DX
80 IFX<=0ORX>=39THENDX=-DX
90 Y=Y+DY
100 IFY<=0ORY>=24THENDY=-DY
110 GOTO40
```

By adding the appropriate lines you can produce a companion image.

```
1 REMBALLC
2 PRINTCHR$(147)
5 INPUT"BORDER,BACKGROUND &
  CHARACTER COLORS";A,B,C
10 PRINTCHR$(147):POKE53280,A:
   POKE53281,B:POKE646,C
20 X=1:Y=1
```

```
30 DX=1:DY=1
40 POKE1024+X+40*Y,81
45 POKE1024+X+5+40*Y,87
50 FORD=1TO5:NEXT
60 POKE1024+X+40*Y,32
65 POKE1024+X+5+40*Y,32
70 X=X+DX
80 IFX<=0ORX>=39THENDX=-DX
90 Y=Y+DY
100 IFY<=0ORY>=24THENDY=-DY
110 GOTO40
```

There is one small problem with the companion graphic character in as much as it tends to move off the screen to the right at times and then appear on the other side. The reason is obvious: the value of DX and DY can not be greater than 40 or 24, but the horizontal position for this character starts not at 1 but at five. This means that when this character, a circle, moves to the left, it meets a barrier five screen addresses away from the left side of the screen. When it moves to the right there is no barrier at that side of the screen, but one five screen addresses farther to the right. As the screen is only 40 columns wide, the circle moves merrily onto the left side of the screen!

There is a neat trick to be learned from this. You can limit the size of the area in which the figure can move simply by altering the values in lines 80 and 100. Try using 29 in line 80 and 14 in line 100. As if by magic, the ball (if you delete lines 45 and 65, you remove the other figure and help with the explanation!) now confines its activities to one area of the screen. Try other values and see what you get.

It is possible to change the left side of the area on the screen where the ball moves by increasing the values on lines 40 and 60 to 1034 and reducing the value in line 80 to 19, thereby forcing the ball to operate in a narrow band in roughly the center of the screen. Play with this for a while altering the program as you wish.

USING COLOR

The addresses 53280 and 53281 bear some attention. They refer to the border and background colors presented on the screen. In direct mode enter the following: POKE53280,1. The border turns a very pretty shade. Now, using the cursor controls (this feature whereby you can waltz all over the screen is called *full screen editing* by the way: not all computers have it), change the value after the comma to produce other colors.

Now, after the previous poke, enter POKE 53281, plus some value or other. Now you will see the background color change too.

The program below will run through all the combinations possible. It will take quite a while to run. You might consider writing a short program that allows you to check various combinations before using them in a program. You will find that some combinations of background and character colors will not work too well. Experiment to find those which work.

```
1 REM:BACKGROUND AND
  BORDER COLORS
5 PRINTCHR$(147)
10 FORBA=0TO15
20 FORBO=0TO15
30 POKE53280,BA
35 PRINT:PRINT:PRINTBA;"BA",BO;"BO"
40 POKE53281,BO
50 FORX=1TO1000:NEXT
60 NEXTBO:NEXTBA
```

The lines below are an example of a user-defined color program, including the text-color, which is controlled by POKE 646.

```
5 PRINTCHR$(147)
10 PRINT"ENTER VALUES FOR
   BACKGROUND, BORDER AND CHARACTER
   COLOR IN THAT ORDER"
20 INPUTA,B,C
30 POKE53281,A:POKE53280,B:
   POKE646,C
40 PRINT"THERE YOU ARE!!!!"
50 FORD=1TO3000:NEXT
60 GOTO5
```

If you buy some of the Commodore Educational software (actually you don't have to buy it. It is in the public domain which means that all you pay for is the disk), you will find that background and border colors change to enhance the effect and draw attention to the text, a neat use!

The thing to watch in your own programs is over-frequent use of color change. If you do too much of it, you will merely unsettle the user, and the program will become a trial. Do be careful too of rapid color changes. It has been known to cause problems for folk who suffer from epilepsy.

TRICKS WITH TEXT AND TITLES

In addition to making the various colors presentable on the screen, it is useful to be able to put pretty borders around titles or around any textual material that we wish to stand out. Look at the following simple program.

```
5 PRINTCHR$(147)
6 POKE53281,5:POKE646,2
7 PRINT:PRINT
10 FORI=1TO20:PRINTTAB(8)"*";:
   NEXT
20 FORI=1TO8:PRINTTAB(8)"*":PRINT
   TAB(28)"*":NEXT
30 FORI=1TO20:PRINTTAB(8)"*";:
   NEXT
```

By using a couple of simple for-next loops, you can make a border roughly in the center of the screen. Putting something in the middle of the pattern could be something of a problem unless we reorganize the program a little. Try the following as one solution:

```
1 REM TITLE
5 PRINTCHR$(147)
6 POKE53281,0:POKE646,5
7 PRINT:PRINT
10 FORI=1TO20:PRINTTAB(8)"*";:NEXT
20 PRINTTAB(8)"*";TAB(28)"*"
25 PRINTTAB(10)"THIS IS THE"
30 PRINTTAB(10)"WAY TO PUT TEXT"
35 PRINTTAB(10)"IN THE TITLE PAGE"
100 FORI=1TO22:PRINTTAB(8)"*";:
    NEXT
```

Full screen editing was mentioned a little while back. There may have been times when you found the idiosyncracies of the C-64 editing facilities to be a trifle irksome, particularly when you have placed something in quotes and then tried to edit the material by means of the cursor controls. Instead of doing what you expect the screen begins to show reversed characters which are difficult to erase.

After an odd number of quote marks (") the cursor controls cease to operate as direct screen manipulators. Instead, the action which would normally take place becomes part of the string (that which is within the quotes) and then performs that action or series of actions when the program is run.

An example is in order:

```
1 REM:CRSR CONTROL
5 PRINTCHR$(147)
10 PRINT"D■I■A■G■O■N■A■L■L■Y
   Y■O■U■R■S"
```

Line 10 shows inverse Q's after each letter of the word DIAGONALLY. These are reached simply by entering a cursor-down after each letter. Likewise with the word YOURS, which shows reversed or blank balls. These are reached by cursor up characters (press the shift key and the vertical cursor key).

When you run the program the two words run at an angle to one another. Try changing the program so that each letter falls on each of two lines alternately. Then try putting in a series of cursor-down commands at the beginning so that the text appears low down on the screen.

Playing with these controls for a while will be repaid by the understanding of a variety of tricks that will enhance the presentation of your programs.

Moving titles are a neat thing to try. The following little program does this very neatly:

```
1 REM:SCREEN MANIPULATION
5 PRINTCHR$(147)
10 FORI=30TO10STEP-1
20 PRINTTAB(I)"HELLO";
25 FORD=1TO50:NEXT
30 PRINTCHR$(147)
40 NEXT
```

The duration loop can be altered to make the word move more slowly. Alternatively you could

introduce another duration or pause loop to hold the CHR$(147) (CLR HOME) for the same amount of time as the word, thus making it blink more.

Talking of blinking, you could remove the movement altogether and make the word flash on and off the screen in the same place each time. See what you can do with that on your own.

A surprise for guests at a party would be to ask them to enter their name and then animate it across the screen, as in the following program.

```
1 REM:SCREEN MANIPULATION
  FIG IV -10
5 PRINTCHR$(147)
6 INPUT"NAME PLEASE";A$
7 PRINTCHR$(147)
10 FORI=25TO10STEP-1
20 PRINTTAB(I)"HELLO ";A$;
25 FORD=1TO200:NEXT
30 PRINTCHR$(147)
40 NEXT
50 GOTO5
```

Alter the program to produce a different background color and then overlay the text in a suitable, contrasting (and legible!) shade.

Consider now the four routines below. They are simple variations upon the theme that we have been discussing. Type in the last program. Bring up the cursor to the print statement in line 20 and insert a space after the quotation marks. Press the CTRL and 9 keys. Enter the line by pressing return and then run the program. The result is reverse characters (but then you knew that already because 9 is the RVS ON key. There is no need to enter a reverse-off command; the carriage return does that automatically.

```
1 REM:SCREEN MANIPULATION
5 PRINTCHR$(147)
10 FORI=255TO10STEP-40
20 PRINTTAB(I)"HELLO";
25 FORD=1TO200:NEXT
30 PRINTCHR$(147)
40 NEXT
```

```
1 REM:SCREEN MANIPULATION
5 PRINTCHR$(147)
8 FORJ=1TO16:PRINT
10 FORI=255TO10STEP-40
20 PRINTTAB(I)"HELLO";
25 FORD=1TO200:NEXT
30 PRINTCHR$(147)
40 NEXT
```

```
1 REM:SCREEN MANIPULATION
5 PRINTCHR$(147)
10 FORI=255TO10STEP-1
20 PRINTTAB(I)"HELLO";
25 FORD=1TO200:NEXT
30 PRINTCHR$(147)
40 NEXT
```

```
1 REM:SCREEN MANIPULATION
5 PRINTCHR$(147)
6 INPUT"NAME PLEASE";A$
7 PRINTCHR$(147)
10 FORI=25TO10STEP-1
20 PRINTTAB(I)" HELLO ";A$;
25 FORD=1TO200:NEXT
30 PRINTCHR$(147)
40 NEXT
50 GOTO5
```

You will recall that I mentioned the fact that each keyboard function was given a numerical value and that it was described by use of the CHR$ function? Well . . create a gap after the TAB instruction on line 20 and insert CHR$(14). The result is lower-case output. Any of the commands can be entered by calling them by their CHR$ codes. Often this is more convenient than using the CTRL in quote mode.

Now for a surprise: carefully create a space in the spot where you can see the reversed R in line 20. Press the return key and then go back to the space with the cursor controls. Now press the CTRL and RVS/ON keys. The cursor should continue to flash in that same spot, but without printing a reverse R. Now press N. Press the return key and then the program. You can see that the word HELLO and the name appears in lowercase, but not in inverse characters. Pressing the shift key and N for any following material will put the character set back to the normal upper case condition.

BAR CHARTS

Many advertisements for computers depict them with the most wonderful business graphics: bar charts and all! The C-64 can do bar charts. A simple way to do them is to use character codes to indicate the output to the screen. Such displays seem to delight office managers out of all proportion to their worth, and it always pays to keep your boss happy, so let's explore it for a while.

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

Clear the screen by pressing the shift and CLR HOME keys and then run the program. Don't worry about the strange line numbering for the moment. Enter a value less than 40 for X for the moment. The checkerboard symbol (CHR$(166)) runs down the side of the screen.

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

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 40 for X, say 45. Now you can see that the whole line is filled, and the row of symbols spills over into the next line too. Try other values for X. How much is required to fill the screen?

Add the following line:

```
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 two alternating symbols.

If you change the value of CHR$(18) to 144 you can turn the whole thing black. CHR$(144) is nothing more than the code for the color black, which you would normally reach via the CTRL key and the appropriate number, or with a POKE646,0.

Let's change the program to read:

```
100  ?CHR$(18):FORI=1TOX:?A$;:NEXTI
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 display into normal color balance by means of 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. Let's 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$(144):FORI=1TOX:?A$;:NEXT
110  ?CHR$(31):FORI=1TOY:?Z$;:NEXT
120  ?CHR$(156):FORI=1TOZ:?Y$;:NEXT
```

Note that we have already assigned CHR$ to variables in lines 30 and 40.

A variation on the program could exist as follows:

```
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, then type:

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

and change the CHR$ code in lines 100 and 120 to 18 and add:

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

Now run the program.

This program, as long as you can remember that the lines represent lemons, oranges and nuts in that order is fine. However, a graphic display should make things self-evident . . . even to your

boss! You need some means of indicating what each line is supposed to mean. The simplest approach is the best . . . merely printout the name of the item! So that you can allow for quite a number of different objects, it would be a good plan to DIMension the array. The completed program is shown in Fig. 4-1. Now your boss knows what the pretty picture is supposed to mean!

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, tridlegroinge and grundget sales, the ratings you give to pin-ups and movie stars or whatever takes your fancy. The program will be much the same, whatever you wish to represent.

So far you have managed to produce a horizontal bar graph. What about a vertical one? It is a little more difficult, and there are limits to the number of items and the quantity of each that can be represented on the screen. There are only 25 lines available in all, and in the program in Fig. 4-2, nevertheless, a start.

For each item, a letter of the alphabet appears. Leave a couple of lines at the bottom of the screen and begin with the address 1906 on line 120. Now run the program.

It is important to ensure that there is sufficient contrast between adjacent colors, or else 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 ask it to deal with too many items in one go. The best approach is to put the whole sequence in a subroutine somewhere at the end of the program and call upon it for just a few items at a time.

CAPSULE REVIEW

Animation is possible on the C-64 by making use of the memory-mapped screen. Memory map-

```
1 REM:BAR CHART
2 PRINTCHR$(147)
4 DIMA$(20)
5 A$(1)="LEMONS"
6 A$(2)="ORANGES"
7 A$(3)="NUTS"
20 X$=CHR$(18)+CHR$(160)
30 Y$=CHR$(18)+CHR$(166)
40 Z$=CHR$(18)+CHR$(162)
50 INPUT"HOW MANY LEMONS";X
60 INPUT"HOW MANY ORANGES";Y
70 INPUT"HOW MANY NUTS";Z
100 PRINTCHR$(144):FORI=1TOX:PRINTX$;:NEXT:PRINTCHR$(5);:
    PRINTA$(1);
105 PRINTCHR$(28)
110 PRINTCHR$(28):FORI=1TOY:PRINTY$;:NEXT:PRINTCHR$(5);:
    PRINTA$(2);
115 PRINTCHR$(31)
120 PRINTCHR$(156):FORI=1TOZ:PRINTZ$;:NEXT:PRINTCHR$(5);:
    PRINTA$(3);
130 POKE646,14
```

Fig. 4-1. The Bar Chart program.

```
5 PRINT"□"
10 P=1906:C=56178
20 PRINT"HOW MANY ITEMS DO YOU WISH TO DISPLAY?"
30 INPUTA
35 PRINT"□"
37 POKE53281,0
50 XX=0
60 FORW=1TOA
70 INPUT N
100 XX=XX+3
110 FORI=1TON:POKEP,224:POKEC,2+XX:P=P-40:C=C-40:NEXTI
120 P=1906+XX:C=56178+XX
130 NEXTW
140 END
```

Fig. 4-2. The Vertical Bar Chart program.

ping is the term used for a condition in which each possible character placement on the screen is located in a particular spot in memory. There are 1000 such spots on the C-64 screen, your TV or monitor, running from 1024 to 2023.

Color memory for each spot runs from 55296 to 56295. Each address must be poked; the first with the value of the character desired and the second with the color desired.

Simple movement up, down, and across the screen can be achieved by means of for-next loops. Diagonal movement must take into account the fact that the screen displays 40 columns. The area of the screen in which a particular object is made to move can be changed at will by altering the start address of the desired movement.

Poking the appropriate address with the value for a space (32) will erase the object from the spot in which it was previously visible.

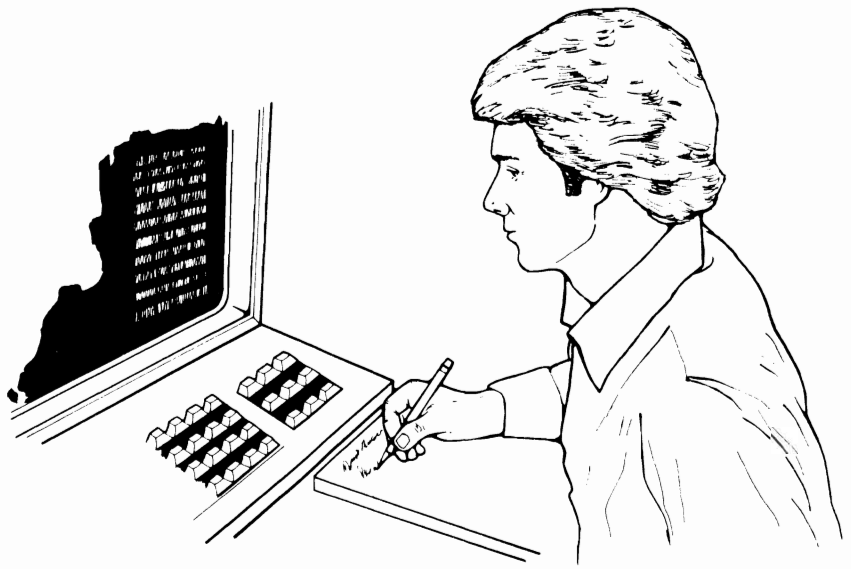
The color of text appearing on the screen can be altered almost ad infinitum by using address 646. Poking this address with a number one less than those shown on the keys for any given color will change the text to the desired color.

Enhancement of screen presentation can be achieved by poking the addresses for background and border colors 53281 and 53280, with the desired values.

The C-64 has full screen editing features; this means that lines may be altered in suit, without having to be brought to a specific spot on the screen. After an odd number of quotes the cursor controls no longer perform their normal functions but rather present a series of inverse characters on the screen. When the program is run these characters are read as if they were the real movements of the cursor and are presented in action form, having the same effect on the positioning of the text as they would normally in editing mode.

All cursor and color controls, as well as those for changing to lower and upper case, have ASCII code values. The action required can be achieved by use of CHR$ and the appropriate value enclosed in parenthesis. The CHR$ codes make for relatively easy production of graphic display of numeric material.

Chapter 5



I have touched on the subject of the DIM function, the means whereby arrays can be dimensioned in order that we do not run out of variable labels. Time spent on this topic will be well repaid and you can have a bit of fun at the same time.

USING ARRAYS

The problem you are going to deal with is rather like the old contention that if you give a hundred monkeys a typewriter each and enough time, they will eventually come up with the works of Shakespeare or Goethe! Hmnn!

The following is the first part of the program:

```
1 REMRNDWORDS
5 PRINTCHR$(147)
10 DIMA$(20)
20 GOSUB501
30 FORD=1TO1000:NEXT
40 PRINTA$(1)+A$(3)+A$(10)+A$
   (1)+A$(4)
45 C=0
70 J=INT(RND(1)*20)+1
102 PRINT
105 C=C+1
110 FORD=1TO2000:NEXT
125 IFC>=25THENEND
130 PRINTA$(J);
140 GOTO70
```

So far there is no line 501 to go to so there is no point in running the program yet. Line 70 uses that fascinating creature: RND. Note the format for this. RND generates, more or less at random, a number between 0 and 1. The expression INT(RND(1) *20)+1 multiplies that number by twenty, which is the number to which A$ has been dimensioned, finds the integer of the result, and adds one. The number so generated is then "translated" into the variable J. A new array called Z$ appears on the scene. This is nothing more than the array A$(J). Line 105 is a counter that increments C by one each time round. A trap is set up at line 115 to make sure that the value of C can be no greater than 5, so that no more than five versions of A$(J) are printed. All you need to do now is put in the various items of the DIMensioned array. Figure 5-1 shows the completed program.

```
1 REMRNDWORDS
5 PRINTCHR$(147)
10 DIMA$(20)
20 GOSUB501
30 FORD=1TO1000:NEXT
40 PRINTA$(1)+A$(3)+A$(10)+
   A$(1)+A$(4)
45 C=0
70 J=INT(RND(1)*20)+1
102 PRINT
105 C=C+1
110 FORD=1TO2000:NEXT
125 IFC>=25THENEND
130 PRINTA$(J);
140 GOTO70
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
```

Fig. 5-1. A simple random word program.

The computer prints mostly garbage. The computer cannot of itself recognize the various parts of speech that when used according to the standards of our language, can make sense. What you need to do is reorganize the program in a slightly different way so that the machine will appear to be able to distinguish between a definite article, a noun, an adjective, and so on. I use the word appear because that is all it is . . . an illusion! Figure 5-2 shows the new version.

Now you can see that whole sentences are being produced by the C-64. Not many of the sentences make all that much sense, but then, computers are not noted for their innate intelligence!

What is needed in order to clean this program up a little is a means of selecting each item at random instead of using the same random number for all of them.

Note that the line 70 deals only with a random number between 1 and 6, even though you have seven items in some of the dimensioned arrays. Perhaps this error will spur you on to expand the program and revamp it to your own delights.

You will notice that the same sentence often occurs more than once. This is caused by the use of the same random number for all items in each sentence. The best way to get around this would be to introduce a neat subroutine that selects a different number for each array before printing out the result. You can do that, but for the time being I'll show you a quick and nasty way of doing it. It will not look very professional but it will do the trick.

Bring the cursor up to line 70 and change the line number to 131. Then change the 131 to 133. Remember to press the return key each time. Change the line number to each of the odd numbers through 139, remembering to change the existing 139 to 140 and 140 to 141. The reorganized lines look as follows:

```
129 REM FIG V-1D
130 Z$=A$(J):PRINTZ$;
131 J=INT(RND(1)*6)+1
132 Z$=B$(J):PRINTZ$;
133 J=INT(RND(1)*6)+1
134 Z$=C$(J):PRINTZ$;
136 Z$=D$(J):PRINTZ$;
137 J=INT(RND(1)*6)+1
138 Z$=A$(J):PRINTZ$;
139 J=INT(RND(1)*6)+1
140 Z$=B$(J):PRINTZ$;
```

```
1 REMRNDWORDS:FIG V-1C
5 PRINTCHR$(147)
10 DIMA$(20)
20 GOSUB501
30 FORD=1TO1000:NEXT
40 PRINTA$(J)+B$(J)+C$(J)+D$
   (J)+A$(J)+B$(J)
45 C=0
70 J=INT(RND(1)*6)+1
102 PRINT
105 C=C+1
110 FORD=1TO2000:NEXT
125 IFC>=25THENEND
130 Z$=A$(J):PRINTZ$;
132 Z$=B$(J):PRINTZ$;
134 Z$=C$(J):PRINTZ$;
136 Z$=D$(J):PRINTZ$;
138 Z$=A$(J):PRINTZ$;
139 Z$=B$(J):PRINTZ$;
140 GOTO70
501 A$(1)="THE "
502 A$(2)="ANY "
503 A$(2)="SOME "
504 A$(4)="MANY "
505 A$(5)="NO "
506 A$(6)="A "
507 B$(1)="CAT "
508 B$(2)="ELEPHANT "
509 B$(3)="RAIN "
510 B$(4)="HOUSE "
511 B$(5)="PIANO "
512 B$(6)="DOG "
513 C$(1)="MEOWS "
514 C$(2)="BARKS "
515 C$(3)="RUNS "
516 C$(4)="GROWLS "
517 C$(5)="SCRATCHES "
518 C$(6)="EATS "
519 C$(7)="BITES "
520 D$(1)="FROM "
521 D$(2)="TO "
522 D$(3)="AROUND "
523 D$(4)="UNDER "
524 D$(5)="AT "
525 D$(6)="BEYOND "
600 RETURN
```

Fig. 5-2. A more complex random word program.

The program is saved on the tape or disk under the name RNDWRD3.

Figure 5-3 shows a short selection of what my computer printed for me. It is not very exciting, and you must forgive such solecisms as "a elephant"... but the computer has not yet been taught much grammar!

Incidentally, the lines used to cause a printout are as follows:

```
129 OPEN 1,4:CMD1
```

This opens a channel to the printer, then:

```
130 Z$=A$(J):PRINT#1,Z$
```

Lines 132 through 140 follow the same format as line 130 and the whole thing is finished off with:

```
141 CLOSE1,4
```

Should consider it at all worth it, you could expand this program ad infinitum (or ad nauseam, according to taste) by adding words to the various arrays, increasing the DIMs by the appropriate amount, and altering the value of the RND; the part to change there is the digit after the *.

With a little imagination, you could produce a fair quantity of so-called poetry. It will have much the same value as so-called computer-generated music, but don't let that worry you. Treated with the right level of levity, the problem will not be too serious, and you can give your friends a good laugh at the same time.

Talking about a laugh, let's try a different sort of program, the one that is shown in Fig. 5-4.

This is a rather long program that illustrates another use of the DIM statement. All that happens is that the user is asked words that are various parts of speech, seemingly to no purpose. At length the computer spins a story around the words so entered. The result is surprising and even amusing.

There is always a temptation to use rude words but this should be avoided for the result is funnier when ordinary words are used.

You can write as many stories as you like to fit the required DIM statements. It would be quite funny to make use of the user input a number of times, each time producing a different tale. The input section would be required only once in your program. The stories could begin at 500, 600, 700 and so on and the C-64 could then select a tale at random 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 strings, each labeled with a variable according to whether it be a verb, noun, or other part of speech, much as in our first program in this chapter. 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 unfamiliar with computers for hours.

An extension of this could be in the form which an erstwhile colleague of mine implemented in 1967. In this program, Anthony Barton arranged for the user to "write" a play. The user was able to select the type of plot, the characters, and the mood via prompts during the course of the program.

If there was too much of one type of mood, say mayhem, the user would be told quite sternly that the mood would, instead, be sweetness and light. Sometimes the computer would interject with the words:"No! That character has had too much to say already. It is now the turn of the ------!"

Such a program looks very much like artificial

```
MANY RAIN EATS BEYOND A RAIN


MANY HOUSE RUNS AROUND SOME HOUSE
A PIANO RUNS AROUND THE PIANO
THE HOUSE GROWLS UNDER MANY CAT
NO DOG EATS BEYOND THE HOUSE
DOG BARKS TO THE DOG
RAIN RUNS AROUND THE PIANO
A RAIN BARKS TO A ELEPHANT
NO CAT EATS BEYOND NO PIANO
A HOUSE SCRATCHES AT THE ELEPHANT
DOG GROWLS UNDER THE RAIN
MANY PIANO GROWLS UNDER NO HOUSE
THE ELEPHANT BARKS TO THE HOUSE
NO RAIN SCRATCHES AT A DOG
A HOUSE RUNS AROUND A DOG
SOME RAIN EATS BEYOND A ELEPHANT
MANY RAIN BARKS TO NO RAIN
A RAIN BARKS TO MANY HOUSE
SOME RAIN RUNS AROUND MANY RAIN
A ELEPHANT GROWLS UNDER MANY CAT
A CAT EATS BEYOND MANY CAT
SOME ELEPHANT BARKS TO A PIANO
SOME RAIN GROWLS UNDER THE PIANO
HOUSE GROWLS UNDER MANY HOUSE
SOME HOUSE SCRATCHES AT MANY RAIN
```

Fig. 5-3. Sample results from the random word program.

```
5 PRINTCHR$(147)
6 REM:WORDS
10 PRINT"EACH TIME YOU SEE THE QUESTION MARK"
20 PRINT"TYPE IN THE PART OF SPEECH ASKED FOR"
30 FORD=1TO3500:NEXT
40 PRINTCHR$(147)
50 DIMA$(20)
60 INPUT"NOUN";A$(1)
70 INPUT"VERB INFINITIVE";A$(2)
80 INPUT"ADVERB";A$(3)
90 INPUT"LIVING THING";A$(4)
100 INPUT"AN EXCLAMATION";A$(5)
110 INPUT"ADJECIVE";A$(6)
120 INPUT"PLURAL NOUN";A$(7)
130 INPUT"FAMOUS CHAIN STORE";A$(8)
140 INPUT"ADVERB";A$(9)
150 INPUT"PART OF BODY";A$(10)
160 INPUT"PART PF BODY";A$(11)
170 INPUT"VERB:INTRANSITIVE, PAST TENSE";A$(12)
200 INPUT"NOUN";A$(15)
210 INPUT"VERB:INTRANSITIVE, INFINITIVE";A$(16)
220 INPUT"VERB:INTRANSIVE, PAST TENSE";A$(17)
230 INPUT"PLACE-NOUN";A$(18)
240 INPUT"VERB:INTRANSIVE, PAST TENSE";A$(19)
250 FORD=1TO2000:NEXT
260 PRINTCHR$(147)
270 PRINT"ONCE UPON A TIME THERE WAS A ";A$(1)
280 PRINT"WHO USED TO LIKE TO ";A$(2);"."
290 PRINT"ONE DAY, THE ";A$(1);" WAS DOING THIS   WHEN ";A$(3);" A ";
300 PRINTA$(4);" STROLLED BY."
310 FORD=1TO5000:NEXT
320 PRINTA$(5);"! SAID THE ";A$(4);". I HAVE NEVER SEEN SUCH ";A$(6);"
    ";A$(7);
330 PRINT" BEFORE. WHERE DID YOU GET THEM?"
340 FORD=1TO6500:NEXT
350 PRINTCHR$(147)
360 PRINT"THEY WERE ON SALE AT ";A$(8)
370 PRINT"WAS THE REPLY. 'THEY WERE VERY CHEAP.'"
380 PRINT"THE ";A$(4);"'S ";A$(10);" BRIGHTENED, HIS ";A$(11);
385 PRINT" WIDENED AND HE ";A$(12)
390 FORD=1TO3000:NEXT
400 PRINT"I AM TRULY AMAZED. I MUST GO AND GET"
410 PRINT"SOME ";A$(7);". THEY WILL HELP MY ";A$(15);" "A$(16);"."
420 PRINT"I WILL COME WITH YOU AND SEE IF THEY FIT"
430 PRINT"SAID THE ";A$(1);
440 FORD=1TO3000:NEXT
450 PRINT". AS THEY ";A$(17);" INTO THE ";A$(18);
460 PRINT" THEY ";A$(19);"!"
470 END
```

Fig. 5-4. The Words party game program.

intelligence at first-sight. It is nothing of the kind, of course. AI programs require much more sophisticated techniques than those you have currently at your disposal. They also take a long time to write!

For the moment, assuming that you have finished playing with the previous program, you will be introduced to a very simple sequence to give you 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 units this is very sound 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. Too slavish a reliance on flowcharts inhibits the growth of the mind.

A SORTING PROGRAM

The first thing to do is to define the problem. You are going to produce a program that allows guests at a party to enter their names. The names will be sorted and then retrieved from storage for whatever purpose you wish. It could be that you would like 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 the 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 do 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 activate the printout when you are ready.

The first part of our program is already done! You have the very thing in the Names program in Chapter 2. You should increase the value of the line numbers, however, just to give yourself some room to work. Figure 5-5 shows the completed program.

This program is essentially the same as the previous name-sort program with one small addition that could prove interesting. There is a small security device in there that will prevent anyone unfamiliar with computers from finding out the contents of the list. Anyone with half a notion of how to list a program will have the password discovered in two seconds. There is a way round this: use the alternate graphics set. This should slow up the neighborhood whizzkid for a while, particularly if you make the password a mixture of digits and letters.

You do not have to use a name, of course. The password could be a birthdate, 070438, or vital statistics such as 362236, or even a telephone number.

An experienced individual will crack whatever code you care to think of, but with most folk, the program will be quite secure. By the time your acquaintances know what you know now, you will be that much further ahead!

The program is meant to be saved to tape or disk after you have stored all the names. Simply enter the appropriate command for the device. The next day when you wish to view the file once more, you load the program with the saved names and run it. To your great annoyance, the program does not list the names.

You know the stuff is there but you cannot get at it! The problem lies with the command you used. The BASIC command run clears out any variables—and all the names that were entered are stored as variables—with the grand result that you seem to have lost all the names!

They are still on tape. Make a mental note to enter **GOTO 130** instead of **RUN**. Then you will be able to enter the password and see the list, assuming you can remember the password!

Programs such as this one are touted as being useful for storing all sorts of information that would normally be kept on little cards. I have never found such a program useful for such things, but for sorting lists—that is another question.

A larger file program is printed in Chapter 12 and is also to be found on the disk or tape under the name File.

```
1 REM:PASSWORD FIG V-3
5 PRINTCHR$(147)
10 INPUT"HOW MANY NAMES";N
20 DIMA$(20)
22 PRINT"THANKYOU":FORD=1TO2000:NEXT
23 PRINTCHR$(147)
25 PRINT"PLEASE ENTER NAMES ONE AT A TIME"
30 FORI=1TON
40 INPUTA$(I):NEXT
45 PRINTCHR$(147)
50 PRINTTAB(8)"NOW SORTING"
60 FORI=1TON-1
70 IFA$(I+1)>=A$(I)THEN120
80 B$=A$(I+1)
90 A$(I+1)=A$(I)
100 A$(I)=B$
110 GOTO60
120 NEXT
125 PRINTCHR$(147)
130 PRINT"TO SEE THE LIST TYPE IN THE PASSWORD"
140 INPUTX$
150 IFX$<>"JIM"THEN300
160 PRINT"THANKYOU":FORD=1TO1500:NEXT
230 FORI=0TON+1:PRINTA$(I):FORD=1TO2000:NEXT:NEXT
299 END
300 PRINT"WRONG PASSWORD":GOTO130
```

Fig. 5-5. The Names program.

For your amusement I have included the bare bones of a program called Talk in Appendix B. It too makes extensive use of DIM statements, but it is not intended to be taken seriously! I provided only the bare bones so that you can extend it if you wish. A glance at the listing will make the principle of the program quite obvious.

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 Ø, is assigned to a variable. A counter is then placed within the body of the repeated material. If there is to be a limit to the total number of times the routine is to be used, a limiting statement, usually in the form IF X = N THEN ------. Screen pokes can be incremented using either the same variable or some other. The same rules apply.

If variable names were confined to just one letter then the computer would be restricted to dealing with 26 numeral and 26 string arrays. Variables can be combinations of letters or letters and numerals in the form AA, DB, A2$. Another method of increasing the number of items, with the added advantage of being able to cluster objects of the same type, is to form an array using the DIM statement.

The format is DIMA(X), where A can be any letter or pair, and X is any quantity up to the limit of RAM available. X need not be directly declared in the DIM statement but may be deduced from elsewhere in the program. For example:

```
10 INPUT"HOW MANY ";X
20 DIM$(X)
```

A$ may then be numbered in sequence and a variety of items assigned. Each value for A$ may then be called at will.

String arrays may be *concatenated* or linked and the result printed to the screen or other device. Concatenation may be the result of a fixed statement: A$(1)+A$(3) and so forth; or it may be the result of a 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 is the total number of strings from which you wish to make selection. For example, the dimensioned array might contain 20 items; you may then 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 the value of 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!)

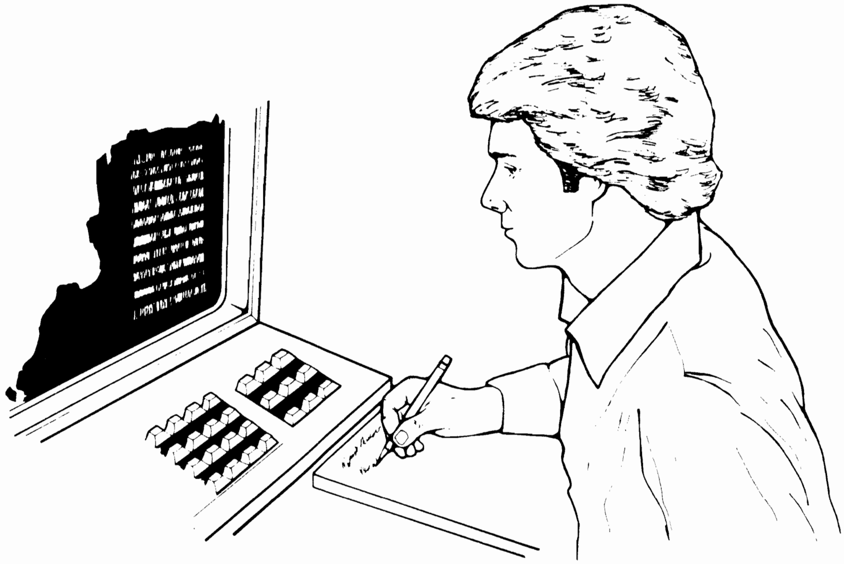
There is a great possibility that the computer will select the same number twice or even three times in a row. This may or may not be inconvenient, depending on the nature of your program. With a large array of numbers to choose from, the likelihood of repetitive selection is lessened but not totally avoided.

The run command erases any variables that have been stored. Thus any data or input material will be lost. If the program has been saved, the data will still be stored on the tape or disk. The program is avoided by use of the GOTO command, followed by a line number after that in which the variables have been declared.

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

Program development is best accomplished by preparing a plan of some kind, even a flowchart; however, it is a good idea to develop good programming practices at the keyboard in much the same way that an expert musician extemporises at his keyboard. There is little point in the possession of a powerful and creative machine such as a computer if you confine your thought processes to the most elementary variety. Be creative!

Chapter 6



This chapter is going to start you off with color. From there it is not such a large jump to animation and sprites.

MORE WAYS TO USE COLOR

Let's look at a way to put text (that's words etc.) on the screen in various colors. Type the following short program in using direct mode.

```
FORI=1TO 16:POKE646,I: PRINT"HELLO";
:FORD=1TO300:NEXT:NEXT
```

Make sure you include the semicolon after "HELLO". Now press the return key and see what happens.

As you can see, only a few of the colors sit well on the background blue. What you are doing here is simply poking various values into the character color address. It doesn't matter what you enter as the string; it could be a series of graphic symbols retrieved from the alternate set. Let's put that in a proper program:

```
5 PRINTCHR$(147)
10 FORI=1TO16
20 POKE646,I
30 PRINT"♥‾_| | |⌒┌⌐L";
40 FORD=1TO200:NEXT
100 NEXT
110 GOTO10
```

Now you can smarten the program up a little with a few more tricks. Before you do so, however, try the following in direct mode.

```
POKE 646,1
```

Then change the value poked into 646; any number between 0 and 15 will do. Run the program, but do not erase what is already on the screen; just press the run/stop key to halt the program. Watch what happens. Each time a value is poked, the color is changed.

There is an address which controls the background color: 53281. Try it in direct mode:

```
POKE 53281,X
```

where X can be any value between 0 and 15. As soon as you have admired that, continue poking 646 with various values as before. Do all this in direct mode. Then add the following line to your program (Press the run/stop and restore keys if you can't see what you are typing.)

50 POKE53281,I

And it might be a good idea to increase the value of the duration loop lest you become mesmerised by the display. Some of the combinations are very pretty. Notice that in some cases there appears to be nothing but colored background. This is because the characters or symbols are the same color as the background and therefore do not show up. Now add this:

60POKE 53280,I

This time the border has changed too; in fact the whole screen is the same color each time. If you wished to provide a different color border, you could do one of two things: either make one of the variables, I, equal to I+2 or some other number, or put in another for next loop with a high value greater than 15 or step the value by 2 or a greater number. Your program might look like this:

```
1 REM:BOCHARBACKCOLOR
5 PRINTCHR$(147)
10 FORI=1TO16
20 POKE646,I
30 PRINT"■¯_I II◠⌐⁄L";
40 FORD=1TO450:NEXT
50 POKE53281,I
60 POKE53280,I+2
100 NEXT
110 GOTO10
```

Good! You seem to be getting somewhere. Now let's get even fancier.

```
5 PRINTCHR$(147)
10 POKE646,6
15 PRINTTAB(10)"\\\\\\\\\\\\ //////
   //////"
16 PRINTTAB(10)"////////////\\\\\\
   \\\\\\"
20 POKE53281,7
25 FORD=1TO300:NEXT
30 POKE53281,6
35 FORD=1TO300:NEXT
37 PRINTCHR$(147)
40 GOTO10
```

This time the background changes and causes the symbols to disappear. The alternate version of that is as follows:

```
5 PRINTCHR$(147)
10 POKE646,6
15 PRINTTAB(10)"\\\\\\\\\\\\ //////
   //////"
16 PRINTTAB(10)"////////////\\\\\\
   \\\\\\"
20 POKE53281,7
25 FORD=1TO300:NEXT
30 POKE646,7
35 FORD=1TO300:NEXT
37 PRINTCHR$(147)
40 GOTO10
```

And now for something just a little different:

```
5 PRINTCHR$(147)
10 FORI=10TO255STEP40
12 POKE646,6
15 PRINTTAB(I)"\ /"
20 POKE53281,7
25 FORD=1TO300:NEXT
30 POKE646,7
35 FORD=1TO300:NEXT
37 PRINTCHR$(147)
40 NEXT
```

A very subtle change in line 10 produces yet another version. You are, of course free to change the background color and the character color to whatever you wish. Do it for the practice, for if you just type in the programs exactly as I have produced them here you will not really learn to control your C-64.

```
5 PRINTCHR$(147)
10 FORI=10TO255STEP41
```

```
12 POKE646,6
15 PRINTTAB(I)"\"
20 POKE53281,7
25 FORD=1TO50:NEXT
30 POKE646,7
35 FORD=1TO50:NEXT
37 PRINTCHR$(147)
40 NEXT
```

You are going to practice poking a couple of other addresses. These refer to screen and color memory. The screen is made up of 1000 possible positions in which symbols can be placed. The positions are numbered from 1024 to 2023. By poking the code of a symbol into one of these positions, you are able to place anything on the screen in just that exact spot you wish. However! If you try a plain vanilla poke to one of these addresses, you cannot see the result! You must also poke a value that represents a color into the color memory map. Like the screen map, there are 1000 possible positions. However (here you go again!) the addresses are different. If they were not, you would be in a frightful muddle. The start address for the top-left corner of the color memory map is 55296. This corresponds to the screen map address 1024. Got that? Actually, there are two grids printed in the C-64 *User's Guide* on pages 63 and 64. A swift look at them will make everything quite clear.
A short program which demonstrates this follows:

```
5 PRINTCHR$(147)
10 FORI=1TO200
12 POKE646,6
15 POKE1024+I,78
16 POKE55296+I,3
20 POKE53281,7
25 FORD=1TO50:NEXT
30 POKE646,7
35 FORD=1TO20:NEXT
37 PRINTCHR$(147)
40 NEXT
```

And now let's make something crawl diagonally down the screen.

```
5 PRINTCHR$(147)
10 FORI=1TO20
12 POKE646,6
14 POKE1024+(I*41),85
18 POKE55296+(I*41),3
20 POKE53281,7
21 FORD=1TO20:NEXT
25 POKE1064+(I*41),74
27 POKE55296+(I*41),3
30 POKE646,7
34 POKE53281,7
35 FORD=1TO40:NEXT
37 PRINTCHR$(147)
40 NEXT
50 GOTO5
```

There is enough material in those few programs to keep you happily learning and out of mischief for quite a long while; however, I expect you would like to get on with some of the fancier stuff.

THE CREATION AND USE OF SPRITES

Fancier stuff on the C-64 is fancy indeed! Sprites are it! In order to deal with sprites we must take a short look at the construction of a symbol on the screen. Each character is made up of dots in an 8 by 8 rectangle.

If you draw a design in that grid by placing a dot in each square, you have the basis for the produc-

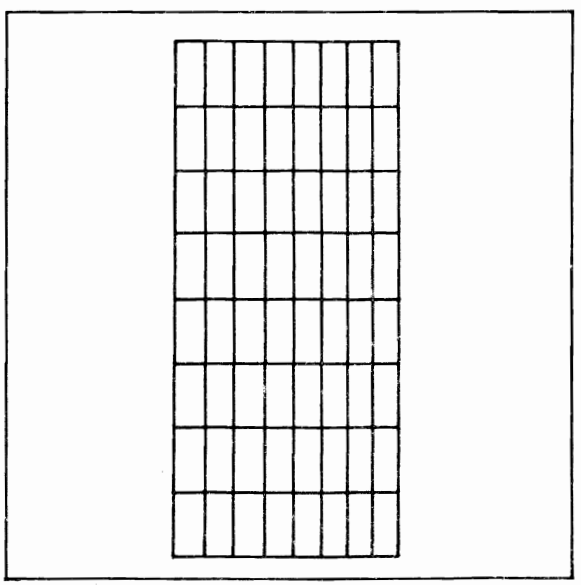


Fig. 6-1. The pattern of pixels in a character.

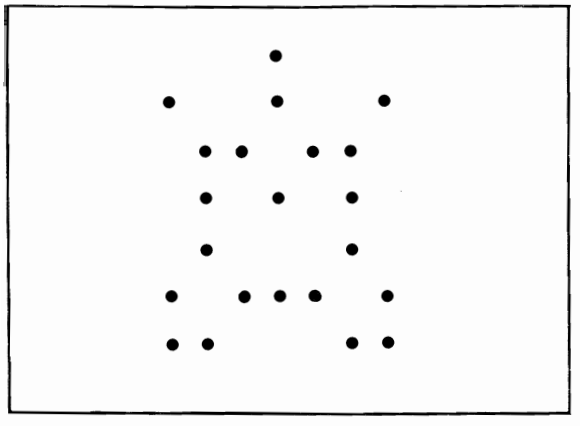


Fig. 6-2. A Character design.

tion of sprites. For example, consider a pattern such as the one shown in Fig. 5-2.

Don't worry about what it is, for I could not tell you! The important thing is that it can be reproduced on the computer by a rather clever method. Each dot is represented by a 1 and each blank by a 0. You thus arrive at a pattern like the one in Fig. 6-3. The left and the bottom edges are left clear so that the new design does not run into another character.

The next thing to do is to translate each line into a decimal number. This is not difficult, but it is time-consuming. Reading from right to left (like Hebrew or Arabic), each position is double the value of the previous one as shown in Fig. 6-4. The right-most position represents 1s; the next position 2s, the next 4s and so on starting with the top line of the graphic, you add each value, as follows:
0+0+0+8+0+0+0+0= 8 (Remember you are reading the row from right to left)
1+0+0+8+0+0+64+0=73

And so on. You then do a little bit of magic with the decimal number and a read-data cluster, and your graphic appears on the screen. Don't worry about that magic for the time being; For the moment you must concern yourself with using what you have learned to create sprites.

As you can see, the process of turning a grid pattern into a decimal number takes quite a bit of time. There is an easier way! Figure 6-5 is a short program that does the conversion for you!

All you have to do is type in eight zeros and ones, and the result is given in decimal. Write the result down one side of your graphic grid and then enter the numbers into a data line when you are ready (later, when you have learned how to do it).

For those of you with a printer, the version in Fig. 6-6 prints the results, saving you the bother of writing it down.

Note that it would not be all that difficult to modify this program so that the results could be stored on file on a disk or tape. The main program that would use the information could then load down the appropriate graphics data when it was needed.

The little program is going to save you a great deal of time, for sprites use rather more than just eight lines—in fact there are 21, each with three parts for a total of 63 sets of binary numbers that have to be translated into decimal! Why do it by hand when you have a computer?

First you have to produce a grid 24 squares across and 21 down. Next, do exactly the same as for user-defined characters put in the dots. Figure 6-7 is an example.

It may not be very clear to you at first, but Fig. 6-12 is a steam locomotive of uncertain vintage, let alone parentage. Some of the design is smoke! Three of these locomotives are necessary to create an animated picture. The other two should differ slightly in the pattern of smoke emanating from the smokestack. Working from left to right, read the zeros and one's in groups of eight across the whole

```
00001000
01001001
00110110
00101010
00100010
01011101
01100011
00000000
```

Fig. 6-3. The pattern of zeros and ones that form the character design.

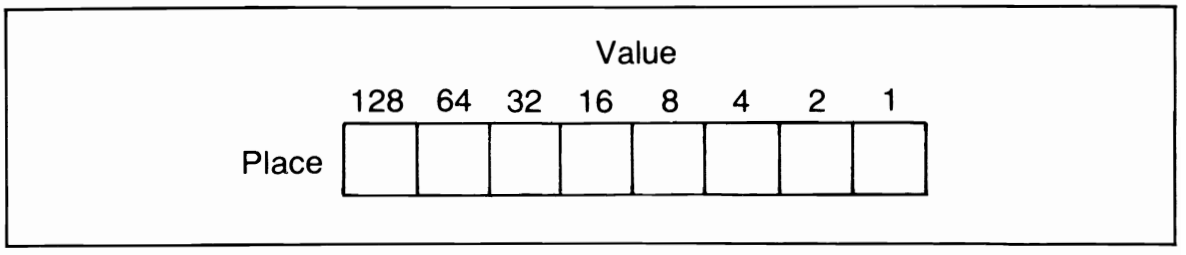


Fig. 6-4. The values assigned to the bits.

width of the graphic. That is, find the decimal number for each group of eight zeros and ones. Do not take the first eight across and then the next eight down, or else your loco will appear quite strange. Read from left to right straight across the page, and then take the next line. You will get three decimal numbers for each of the twenty one lines giving 63 decimal numbers in all. Place these in data statements at the end of your program. Better still, type or load in the program in Fig. 6-8 in its entirety and then examine it.

Essentially this is a straightforward program. Some of the addresses may be unfamiliar at this point for they refer to the sound (Did you turn the sound up while the train ran?) The reverse E is obtained by holding down the CTRL key and then

```
5 REM BINARY-DECIMAL CONVERSION PROGRAM
10 INPUT"BINARY NUMBER (8 BITS)";X$
20 IFLEN(X$)<>8THENPRINT"EIGHT(8) BITS":GOTO10
30 Z=0:Y=0
40 FORI=8TO1STEP-1:Y=Y+1
50 Z=Z+VAL(MID$(X$,Y,1))*2↑(I-1)
60 NEXTI
70 PRINTX$;"= ";Z
80 GOTO10
```

Fig. 6-5. Binary to Decimal Converter program, S on-screen version.

```
5 REMBINARY TO DECIMAL CONVERTER
6 OPEN1,4
10 INPUT"ENTER 8-BIT BINARY NUMBER";A$
12 IFLEN(A$)<>8THENPRINT"8 BITS PLEASE...":GOTO10
15 TL=0:C=0
20 FORX=8TO1STEP-1:C=C+1
30 TL=TL+VAL(MID$(A$,C,1))*2↑(X-1)
40 NEXTX
50 PRINTA$;"BINARY ";"=";TL;" DECIMAL"
55 PRINT#1,TL;" DECIMAL"
60 GOTO10
```

Fig. 6-6. Binary to Decimal Converter program, printer version.

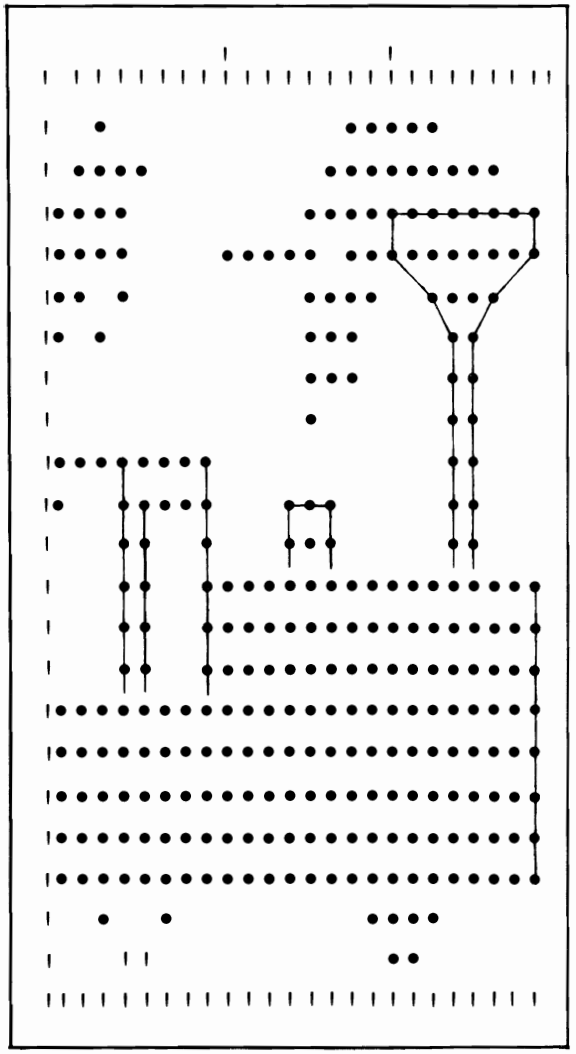


Fig. 6-7. One of the sprite designs for the locomotive program.

pressing 2. This causes the text to be printed in white and is an alternative method to poking 646. You could liven the program up a little by adding line 8 as follows:

```
8POKE53281,H+2
```

You could add whatever value after H you wish.

Again for fun (to amuse yourself or passing children) remove line 37, but not by just typing the number and pressing the return key! Rather, bring up the cursor to line 37, and then, by creating a gap, turn it into 370. Then add line **360 END**. Now you can erase 37. You will see why in a minute. The train has been reduced in size.

Now bring the cursor back to line 370 and erase from it each reference to V+23 and the associated value. This leaves three sets of POKEV +29. Now erase the zero from line 370, turning it back once more into 37. Now line 37 is back in the program. The train has been miraculously elongated!

Put the POKEV+23 statements back in again (they follow the same format as V+29 with the same values 1,2,3) and then erase the V+29 statements. This time you have a tall skinny loco. Reestablish line 370 as it was, and then put it back in the program as line 37. Each time you run this program (save it on tape or disk) you can amaze folk merely by entering 37 and pressing the return key, thus reducing the size of the engine. The speed of the train is controlled by the T loop at the end of line 80. Increase the value of the upper figure to slow the train down.

Lines 1 and 2 are self-explanatory. Line 5 sets S at the beginning address of the sound chip 54272. S+24 is the volume; S and S+1 are the high and low bytes of the first voice and set the pitch. S+5 and S+6 are the registers for attack/delay and sustain/release.

S+7 and S+8 are voice 2 registers; S+12 and S+13 are voice 2 envelope generators. You'll learn more about sound in Chapter 7.

In line 15, V is the starting address of the video chip. It also controls sprite 0's horizontal or X position. It is convenient to take the starting address of a particular facility and merely add the appropriate number to obtain the required address; thus you can use POKEV+21 for the sprite enable register. Poking the value 1 into this register turns on one sprite officially, sprite 0. The data statements must be put somewhere in RAM in order for the computer to be able to use them. Lines 20, 25, and 30 place the information for the three images that make up the loco, with its moving smoke, in locations 12288 to 12350; 12352 to 12414; and

```
1 INPUT"■CHOOSE A NUMBER FOR COLOR";H
2 INPUT"AND TYPE YOUR NAME□";Z$
5 S=54272:POKES+24,15:POKES,220:POKES+1,68:POKES+5,80:POKES+6,195
10 POKES+7,120:POKES+8,100:POKES+12,148:POKES+13,195
15 PRINT"□":V=53248:POKEV+21,1
20 FORS1=12288TO12350:READQ1:POKES1,Q1:NEXT
25 FORS2=12352TO12414:READQ2:POKES2,Q2:NEXT
30 FORS3=12416TO12478:READQ3:POKES3,Q3:NEXT
35 POKEV+39,H-1:POKEV+1,68
37 POKEV+23,1:POKEV+29,1:POKEV+23,2:POKEV+29,2:POKEV+23,3:POKEV+29,3
40 PRINTTAB(160)"■THE ";Z$;" EXPRESS"
45 P=192
50 FORX=0TO347STEP2
55 RX=INT(X/256):LX=X-RX*256
60 POKEV,LX:POKEV+16,RX
70 IFP=192THENGOSUB200
75 IFP=193THENGOSUB300
80 POKE2040,P:FORT=1TO10:NEXT
85 P=P+1:IFP>194THENP=192
90 NEXT
100 DATA0,3,224,3,223,240,7,225,255,7,251,255,7,240,60,0,0,24,0,0,24
101 DATA0,0,24,255,0,24,159,28,24,25,28,24,25,255,255,25,255,255
102 DATA31,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255
103 DATA60,1,224,24,0,192
104 DATA48,3,224,111,220,240,95,220,255,87,231,255,79,92,60,203,224,24
105 DATA235,224,24,120,0,24,255,0,24,191,28,24,121,28,24,25,255,255
106 DATA25,255,255,31,255,255,255,255,255,255,255,255,255,255,255
107 DATA255,255,255,255,255,255,60,1,224,24,0,192
108 DATA32,3,224,120,7,252,248,15,255,248,11,255,216,15,60,160,30,24,0,30,24
109 DATA0,8,24,255,0,24,159,28,24,25,28,24,25,255,255,25,255,255,31,255,255
110 DATA255,255,255,255,255,255,255,255,255,255,255,255,255,255,255
111 DATA60,1,224,24,0,192
200 POKES+4,129:POKES+4,128:RETURN
300 POKES+11,129:POKES+11,128:RETURN
```

Fig. 6-8. The animated locomotive program.

12416 to 12478. S1 is the first image; S2, the second; and S3 the third. The data is read and then poked into each set of memory locations.

V+39 is the address that controls the sprite color; V+1 is the address that controls the Y or vertical position of the sprite.

V+23 and V+29 are the addresses that allow you to expand the sprite in vertical and horizontal directions respectively. You will recall that you played with line 37 a short while ago. Personally I prefer the loco to be expanded horizontally. It looks quite good.

Line 40 is obvious. Line 60 moves the sprite along using the values derived in lines 50 and 55. Line 80 pokes location 2040 with the value P. This tells the computer where in memory to look for the sprite data. Line 80 also gives the duration of the halt part of the movement. Line 85 increments the value of P and also sets a limit. Line 90 ends the for-next loop begun in line 50, which controls the movement across the screen.

Lines 200 and 300 concern the sound, which is dealt with more fully in Chapter 7. For the moment let's stick to the sprites! You might like to add line 199 END to prevent the appearance of an error message. The program is worth studying for much can be learned about animation from it.

Line 55 needs a little more explanation at this point. The matter will become clearer after you have seen a few more programmed examples, however. The line dictates the movement of the sprites across the screen. As you know, an 8 bit binary number (a byte) can have a maximum value (all 1's) of 255. Thus you can represent a total of 256 consecutive numbers (0 to 255 = 256) by means of the various configurations of a byte. You can therefore use each of the numbers to refer to a horizontal position on the screen.

However, there are more than 256 horizontal positions across a screen. If you were limited to just those 256 spots, your sprites would be confined to just a portion of the screen and never reach the right hand edge! The formula RX=INT(X/256) generates a value for RX, which is a convenient mnemonic for Right-side X coordinate or Right-side horizontal parameter.

Address 53264 helps control the X position of all sprites to enable them to reach the right hand edge of the screen. If you poke this address (POKE V+16 in the program) with the value generated for RX (0 or 1) you can allow the sprite to move beyond position 256 across the screen. LX controls the Left hand side, but then that is obvious by extrapolation. In line 45 P is set at 192 to point sprite 0 to memory that begins at 12288. (Do you remember line 20? Note that 192 multiplied by 64 equals 12288.) Line 85 increments P by 1 to set the pointer to the memory at location 12352. It is then obvious what happens when P = 194, that is P+2. You do not want P to go beyond 194, and so you should set a limit at line 85. Clearly, you could build data for more sprites, in which case you could allow the pointer P to go beyond 194.

You also use P to control what happens to the sound. This is a neat trick for when P=192 you use the waveform control for the first voice; when it is 193, you use the waveform control for the second voice.

When you run the program you will probably get an error message, unless you put in line 199 END to get rid of it.

Now it might appear to you that you have used three sprites, indeed, I might have given that impression. The fact is that you have actually used only one, but have used three sets of data to animate it. This is useful information for it allows you to realize that although the C-64 seems to have only 8 sprites available, in reality, with clever programming, you have access to an almost unlimited number, although they cannot all appear on the screen at the same moment. By recycling the data, the 8 sprites can become what you will! For example, let us suppose that you have designed a game in which alien Boozle-Bugs attack your space station and drink your life juices. After a certain amount of imbibing, the B-B's turn into Twirdle-Groindges. Clearly the sprite that has, up to now, been a Boozle-Bug, with the particular set of data to be so, can now become a T-G. If your B-B's do something strange while they move, such as poke out their tongue, then this, as well as the transformation, can be accomplished with one sprite that accesses as

many sets of data as necessary. Very clever!

Obviously a whole collection of similar objects can access the same data. Only the placement on the screen will differ.

Let's take things one at a time, altering various factors until we can become totally expert.

Let's begin with a short program that puts a white square on the screen.

```
5 REM SPRITES
10 PRINTCHR$(147)
20 POKE2040,192
30 FORS=12288TO12350+62:POKES,
   255:NEXT
40 V=53248
50 POKEV+21,1:POKEV+39,1:POKEV,
   24:POKEV+1,100
```

Line 10 is obvious; it clears the screen. Line 20 sets the sprite pointer found at 2040. The value in this address indicates the location from which sprite 0 gets its data. Line 30 puts sprite zero into 63 bytes of RAM starting at address 12288. Line 40 sets V at the start of the video chip. From now on you can use V+X where X is all the other addresses you might need. Doing it like this not only saves a lot of effort on your part but also saves memory. Although the C-64 has quite a lot more memory to play with than many other computers of comparable price, you should learn to conserve it.

Line 50 turns on sprite 0. Hold it! If V+21,1 turns on sprite 0, why not type V+21,0? Well . . . it has to do with the funny way programmers use numbers. 0 is just as valid a number as any other, and so it is convenient to start counting at zero. In addition, to turn on sprite 1, 2 is poked to address V+21; for sprite 2, a 4 is poked to V+21, for sprite 3, an 8, for sprite 4, a 16, and so on. See Fig. 6-9.

Now perhaps you can see that the sprites are labeled or numbered according to the decimal equivalent of binary numbers! Thus sprite 7 would be V+21,128.

Taking all this a little further, it is possible to turn on any combination of sprites by adding the decimal values together. For example: Sprites 0 and 2 would be V+21,5; Sprites 3 and 7 would be V+21,136; sprites 3,5 and 6 would be V+21, 8+32+64=104. All sprites together would be V+21,255, and no sprites on at all would be V+21,0.

V+39 concerns color. In this case you have chosen white. Now, in direct mode type POKEV +39,3. Do not clear the screen first. The square turns green. Bring the cursor back up to your command, move it along to the 3, type 14, and press the return key. Then type in other numbers. What number makes the sprite disappear? Type in that number (it is 6) and then type LIST. You can see that part of the program listing is missing just where the bottom of the sprite should be, proving that it is still there. Change the color again.

V,24 dictates where the sprite will appear on the screen horizontally run the program (to get rid of the listing) and then, in direct mode enter POKEV,28. The sprite moved. Now add the following line:

```
60 FORI=1to255:POKEV,I:NEXT
```

See how beautifully the sprite moves across the screen? Poke a new value for color (that's V+39, you'll recall) and run the program.

What's this? The sprite is still white! Ah! we must change the color in the program. Do that with a neat bit of editing (you can always change it later) and then run the program.

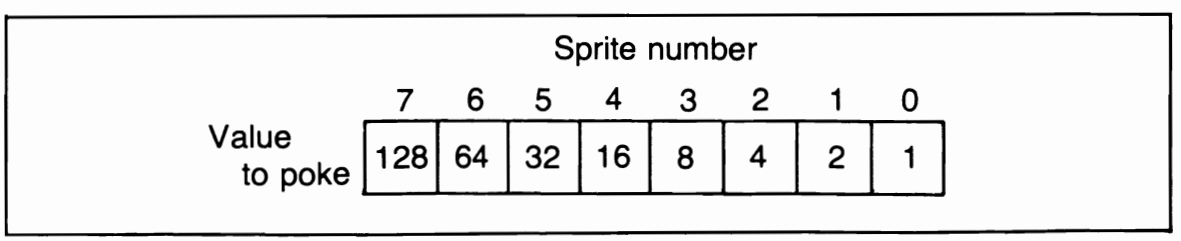


Fig. 6-9. The values to poke to enable or expand individual sprites.

Now, save the present program if you wish, and then, either by typing in afresh or by editing enter the following modification:

```
5 REM SPRITES MOVING
10 PRINTCHR$(147)
20 POKE2040,192
30 FORS=12288TO12350+62:POKES,
   255:NEXT
40 V=53248
50 POKEV+21,1:POKEV+39,1:POKEV+1,
   100
60 FORI=1TO255:POKEV,I
70 FORX=1TO15:POKEV+39,X:FORD=1
   TO10:NEXT:NEXT
80 NEXT
```

This program moves the sprite quite slowly across the screen but flashes the color changes very rapidly. How would you move the sprite rapidly but change the color slowly? Try the following which simply swaps the for-next loops around.

```
5 REM SPRITES MOVING
  DIAGONALLY
10 PRINTCHR$(147)
20 POKE2040,192
30 FORS=12288TO12350+62:POKES,
   255:NEXT
40 V=53248
50 POKEV+21,1:POKEV+39,1
60 FORX=1TO15:POKEV+39,X:
   FORD=1TO20:NEXT
65 POKE53281,X+2
70 FORI=1TO255:POKEV,I:POKEV+1,
   I:NEXT
80 NEXT
```

This program also changes something else. You will have noticed that the Y parameter or vertical placement of the sprite was set at 100. In the current version I have made V+1 subject to the changing value of I, in just the same way as the X or horizontal parameter. This brings the sprite diagonally down the screen.

Now I will put in a new variable: Z=255. Add it to line 40. Then change V+1 on line 70 to read: POKE V+1, Z–I. Now the sprite moves diagonally *up* the screen.

What about making the sprite move in the opposite direction, that is from right to left. That's easy, for you will recall that you have this marvelous feature whereby the computer can count backwards making use of the step command.

```
5 REM SPRITES MOVING
  DIAGONALLY
10 PRINTCHR$(147)
20 POKE2040,192
30 FORS=12288TO12350+62:POKES,
   255:NEXT
40 V=53248:Z=255
50 POKEV+21,1:POKEV+39,1
60 FORX=1TO15:POKEV+39,X:
   FORD=1TO20:NEXT
65 POKE53281,X+2
70 FORI=255TO1STEP-1:POKEV,
   I:POKEV+1,I:NEXT
80 NEXT
```

There you are! Perhaps you might like to produce a longer program that makes the sprite go through its paces one at a time.

You will notice that the sprite seems to appear from off-screen. The value of I controls the coordinates of the top left-hand corner of the sprite.

The next three programs introduce the use of two and three sprites at one time, each doing something different.

```
5 REM SPRITES MOVING
  DIAGONALLY
10 PRINTCHR$(147)
20 POKE2040,192:POKE2041,193
30 FORS1=12288TO12350+62:POKES1,
   255:NEXT
32 FORS2=12352TO12414+62:POKES2,
   255:NEXT
40 V=53248:Z=255
50 POKEV+21,3:POKEV+39,1
60 FORX=1TO15:POKEV+39,X:POKEV+40,
   X+1:FORD=1TO20:NEXT
65 POKE53281,X+2
70 FORI=255TO1STEP-1:POKEV,
   I:POKEV+1,I
72 POKEV+2,Z-I:POKEV+3,Z-I:NEXT
80 NEXT
```

```
5 REM SPRITES MOVING
  DIAGONALLY
10 PRINTCHR$(147)
20 POKE2040,192:POKE2041,193:
  POKE2042,194
30 FORS1=12288TO12350+62:POKES1,
  255:NEXT
32 FORS2=12352TO12414+62:POKES2,
  255:NEXT
34 FORS3=12416TO12478+62:POKES3,
  255:NEXT
40 V=53248:Z=255
50 POKEV+21,7:POKEV+29,4:
  POKEV+23,2
60 FORX=1TO15:POKEV+39,X-1:
  POKEV+40,X+1:POKEV+41,X+2:
  FORD=1TO20:NEXTD
65 POKE53281,X+3
66 FORA=1TO255:POKEV+4,A:POKEV+5,
  100
67 NEXTA
70 FORI=255TO1STEP-1:POKEV,
  I:POKEV+1,I
74 POKEV+2,Z-I:POKEV+3,Z-I
76 NEXTI
79 NEXTX
```

```
5 REM SPRITES MOVING
  DIAGONALLY
10 PRINTCHR$(147)
20 POKE2040,192:POKE2041,193:
  POKE2042,194
30 FORS1=12288TO12350+62:POKES1,
  255:NEXT
32 FORS2=12352TO12414+62:POKES2,
  255:NEXT
34 FORS3=12416TO12478+62:POKES3,
  255:NEXT
40 V=53248:Z=255
50 POKEV+21,7:POKEV+29,4:
  POKEV+23,2
60 FORX=1TO15:POKEV+39,X-1:
  POKEV+40,X+1:POKEV+41,X+2
65 POKE53281,X+3
66 FORA=1TO255STEP10:POKEV+4,A:
  POKEV+5,100
70 FORI=255TO1STEP-1:POKEV,I:
  POKEV+1,I
74 POKEV+2,Z-I:POKEV+3,Z-I
76 NEXTI
77 NEXTA
79 NEXTX
```

Now the problem of sprites going all the way across the screen really ought to be tackled. To do this, let's go back to a simple program that you have had before, or at least a version of it.

```
5 REMSPRITE MOVING
8 X=0
10 PRINTCHR$(147)
20 POKE2040,192
30 FORS=12288TO12350+62:POKES,
  255:NEXT
40 V=53248
50 POKEV+21,1:POKEV+39,X:POKEV+1,
  100
70 FORI=1TO347
72 R=INT(I/256):L=I-R*256
75 POKEV,L:POKEV+16,R
80 NEXT
85 POKE53281,X+2
90 X=X+1
100 GOTO20
```

The important lines here are 70,72 and 75. 70 gives a higher value, up to 347. Line 72 gives a value for R (either 0 or 1) which is then poked into location V+16 which controls the right side of the screen. L is given a value that enables the sprite to start at the left hand edge if the value of I is less than 256. To make the sprite move faster, merely add a step value to line 70. Try various values for the step. Some of them may make the sprite move too quickly, but don't forget that when there are a number of sprites moving at once, the entire action is slowed down. Being able to move some of them quickly speeds up the action in games, without having to resort to machine code (although without question that is the superior way to go).

Each sprite can be expanded along the two dimensions, either individually or in combination. The vertical expansion control is at V+23. Poke that address with the number of the sprite (or sprites) you wish to enlarge; for example, `62 POKEV+23,1` will increase the vertical size of sprite 0. Try it.

This feature is extraordinarily useful, particularly when used in combination with the horizontal expansion feature which is found at V+29. Put

POKE V+29,1 on line 62 to see the sprite become wider. The train program displays this feature.

You could design a series of characters to teach a young child his letters. The data could be kept in data lines; each sprite from 0 to 7 (8 sprites in all) can call up different sets of data and thus be used for more than one letter (there should be no problem here for a young child would not be using words of more than four or five letters). The letters could be doubled in size in the initial stages and then reduced as the child becomes accustomed to them. Letters could be moved across the screen. The size could be changed along each dimension in much the same fashion as on children's TV shows.

The advantages of the expansion feature for games is obvious; as, indeed, are its advantages for the emphasis of information in business charts and diagrams.

Try the following:

```
5 REM SPRITES EXPANSION
10 PRINTCHR$(147)
20 POKE2040,192:POKE2041,193
30 FORS1=12288TO12350+62:POKES1,
   255:NEXT
32 FORS2=12352TO12414+62:POKES2,
   255:NEXT
40 V=53248:Z=255
50 POKEV+21,3:POKEV+39,1
60 FORX=1TO15:POKEV+39,X:POKEV+40,
   X+1
62 POKEV+23,1:POKEV+29,2
65 POKE53281,X+2
70 FORI=255TO1STEP-1:POKEV,I:
   POKEV+1,I
73 POKEV+2,Z-I:POKEV+3,Z-I
75 NEXT
80 NEXT
```

You will have noticed that one square passes in front of the other. There is a rule here; Sprite 0 has priority over all other sprites; it will thus pass in front of any other. Sprite 7 has no priority and will thus pass behind all others. The lower the sprite number the greater the priority. Bear this in mind as you write your masterpiece program. Address V+30 detects sprite-to-sprite collision. Add this line to your program and see what happens.

IF PEEK (V+30) AND 1=2 THEN POKE V+21,0

The chapter began with color . . . and it will end with color: Multicolor!

Look at the listing below.

```
5 REM SPRITES MULTICOLOR
10 PRINTCHR$(147)
20 POKE2040,192:POKE2041,193
30 FORS1=12288TO12350+62:POKES1,
   255:NEXT
32 FORS2=12352TO12414+62:POKES2,
   255:NEXT
40 V=53248:Z=255
50 POKEV+21,3:POKEV+28,1:POKEV+37,
   5:POKEV+38,8
52 FORI=12288TO12414+62STEP2:
   POKEI,19:NEXT
60 FORX=1TO15:POKEV+39,X:POKEV+40,
   X+1
65 POKE53281,X+2
70 FORI=255TO1STEP-1:POKEV,I:
   POKEV+1,I
73 POKEV+2,Z-I:POKEV+3,Z-I
75 NEXT
80 NEXT
```

Compare it with the previous listing and you will note that three more pokes have been added to line 50 and that a for-next loop has appeared at line 52.

The additional pokes do the following: V+28 turns on the multicolor mode. V+37 sets Multicolor 1; that is the first of two colors you chose for your sprite. V+38 chooses the second color.

The loop says: take the data at addresses 12288 to 12414+62 but leave out every second one (that's what the STEP2 instruction does, as you will already know). Now poke the right hand bits a group of eight with their binary values converted into decimal, thus:

128	64	32	16	8	4	2	1	
				1	1	1	1	= 15

The effect is a checkered square. Now try 27, then 65, then 85. Then try the same values with the

STEP removed. Now put the STEP back and try various values there: 3 for example, or 5, or 4, or 9, or 17, or what you will. Notice that the sprite that moves downwards shows the background color of the screen through the gaps. The upward-moving sprite is in two colors, which are fixed. Do you know why?

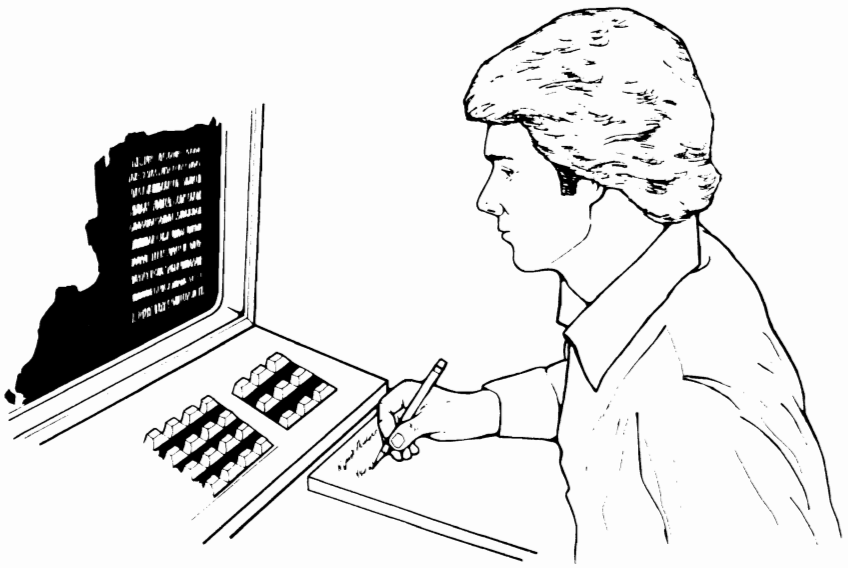
The reason is that you have set V+28 to 1, that is, only one sprite is set to multicolor mode. The other merely has gaps in it through which the screen color shows, making it look like a multicolor sprite. This is a very useful trick particularly if you are writing a program testing visual perception. (I am thinking here of the use of graphics in the field of psychology.)

To deal with both sprites you must add their values together: 1 (for sprite 0) and 2 (for sprite 1) = 3. Just to assure yourself that you are in control, change the value poked to V+28 to a 2. This will now control the second sprite. To control both, as was said, POKEV+28,3. Both are now black and green. To make them change colors you must POKE V+37 and V+38 with X (from line 60).

Just for fun, here is an amended version of the sprite program, in a more holey fashion!

```
5 REMSPRITE HOLEY
8 X=0
10 PRINTCHR$(147)
20 POKE2040,192
30 FORA=12290TO12290+15STEP3:
   POKEA,5:NEXTA
32 FORA=12289TO12289+20STEP2:
   POKEA,195:NEXTA
34 FORA=12309TO12309+23STEP2:
   POKEA,65:NEXTA
40 V=53248
50 POKEV+21,1:POKEV+39,X:POKEV+1,
   100
70 FORI=1TO347
72 R=INT(I/256):L=I-R*256
75 POKEV,L:POKEV+16,R
80 NEXT
85 POKE53281,X+2
90 X=X+1
100 GOTO20
```

Chapter 7



Let's get one fact straight from the start: the sound features on the Commodore 64 are such that to do the subject proper justice, a whole book is needed. This chapter is going to do little more than skim the surface; however, with the aid of the program, which I have called Music, you will have an opportunity to explore each of the aspects.

The program listing shown in Fig. 7-1 is rather long and might seem too daunting to type in at one session. I have therefore organized it in sections.

Each time you see a line which reads S=54272 you will know that a new section of the program has begun. For example, there is a section, quite self-contained, which runs from line 200 to 280; likewise from 300 to 390. In fact, each section starts at a hundred line or just after and finishes before the next even hundred.

PRODUCING SOUND ON THE C-64

Producing sound on the C-64 can be as simple or as complex as you like. I am going to take things fairly simply and work up a gradual head of steam.

Sound production is controlled by poking values into a series of addresses that begin at 54272 and finish at 54296. In order to keep things reasonably clear, I will set a variable, S, to equal the first address and then merely add values to S for the various addresses.

Begin by typing in, in direct mode, the following:

```
S=54272:POKES,75:POKES+1,34:POKES+
24,15
```

Turn up the sound on your TV and then press the return key. The note you hear is being produced by two values poked into S and S+1. These addresses are the low and high byte frequency controls for voice 1. S+24 controls the volume, which in this case is set at full, the range being from 0 to 15. Now enter the following, still in direct mode:

```
POKES+24,8
```

The volume is reduced.

Fig. 7-1. Music program.

```
  5 PRINTCHR$(147)
 10 PRINTTAB(2)"THIS PROGRAM TELLS ABOUT THE SOUND"
 15 PRINT"ABILITIES OF THE COMMODORE 64 COMPUTER"
 20 FORD=1TO5000:NEXT
 25 PRINTCHR$(147)
 30 PRINT:PRINT"THE MEMORY LOCATIONS FOR SOUND ACCESS   BEGIN
    AT 54272"
 35 FORD=1TO3800:NEXT
 40 PRINT"VALUES ARE POKED INTO THESE ADDRESSES   TO ALTER
    VARIOUS ASPECTS OF SOUND"
 42 FORD=1TO4500:NEXT
 45 PRINT"THE RESULT IS NOT JUST A 'BEEP' BUT A"
 50 PRINT"FULLY CONTROLLABLE MUSICAL INSTRUMENT"
 55 FORD=1TO5500:NEXT
 60 PRINT:PRINT"THERE ARE 24 CONSECUTIVE LOCATIONS THAT
    WE CAN USE"
 65 PRINT"HOWEVER, TO MAKE IT EASY, WE WILL JUST  SET A
    VARIABLE 'S' TO 54272"
 70 PRINT"AND THEN PUT 'S+X', WHERE X IS EQUAL TO A NUMBER
    BETWEEN 1 AND 24"
 75 FORD=1TO9500:NEXT
 80 PRINT"IN THIS WAY WE WILL SAVE OURSELVES THE  TROUBLE OF
    HAVING TO RECALL LARGE"
 85 PRINT"NUMBERS"
 90 FORD=1TO5500:NEXT
 95 PRINTCHR$(147)
100 PRINT:PRINT"LET'S BEGIN"
101 FORD=1TO300:NEXT
110 PRINT:PRINT"THERE ARE THREE VOICES AVAILABLE ON     THE
    COMMODORE 64"
115 PRINT:PRINT"IN ADDITION TO THESE THERE IS A WHITE   NOISE
    GENERATOR WHICH IS USED "
120 PRINT"MAINLY FOR SOUND EFFECTS"
122 FORD=1TO3500:NEXT
125 PRINT"FIRST LET US LOOK AT A SHORT SECTION OF A PROGRAM
    LISTING"
130 FORD=1TO4000:NEXT
132 LIST2000-2015
140 PRINTCHR$(147)
145 PRINT"NOW LET US HEAR THE RESULT OF ALL THAT"
150 PRINT"MAKE SURE THAT YOU HAVE THE SOUND TURNED UP ON YOUR
    TV SET!"
155 FORD=1TO3000:NEXT
160 PRINT"HERE WE GO"
165 FORD=1TO2000:NEXT
200 S=54272:REM START ADDRESS
```

```
201 FORC=STOS+24:POKEC,0:NEXT
202 X=0
203 POKES+5,16:POKES+6,32
205 POKES+24,12:REM SETS VOLUME (COULD  BE FROM 0-15)
207 POKES+1,34:POKES,75
208 X=X+1
209 POKES+4,33:REM S+4 IS THE CONTROL SETTING FOR VOICE 1
211 FORD=1TO90:NEXT:REM THIS IS THE DURATION OF THE NOTE
212 POKES+4,32:REM THIS STOPS THE TRIANGLE WAVEFORM
213 FORD=1TO50:NEXT
214 IFX=<6THEN207
220 PRINTCHR$(147)
225 PRINT"NOW LET US CHANGE EACH OF THE VARIABLES"
230 PRINT"ATTACK NUMBERS ARE 128,64,32,16"
232 PRINT"DELAY NUMBERS ARE 8,4,2,1"
233 PRINT"EACH RANGE READS FROM THE HIGHEST RATE TO THE LOWEST"
235 PRINT"FOR EXAMPLE: 128 WOULD MEAN THAT THE  ATTACK RATE
    WOULD BE HIGH BUT THE"
237 PRINT"DECAY RATE WOULD BE ZERO"
239 PRINT"BY COMBINING NUMBERS FROM BOTH RANGES  BOTH ATTACK
    AND DECAY ARE SET"
240 PRINT"THUS 136 WOULD MEAN THAT BOTH HIGH      ATTACK AND
    DECAY RATES ARE SET"
241 PRINT"THE NUMBER 129 WOULD MEAN THAT THE HIGH ATTACK RATE
    AND THE LOW DECAY"
243 PRINT"RATE ARE SET"
245 PRINT"THE SAME SYSTEM IS USED FOR SUSTAIN/RELEASE"
247 PRINT"VOLUME RANGES BETWEEN 0 AND 15"
249 PRINT"WAVEFORM SETTINGS ARE 17,33 AND 65 AND 129"
250 INPUT"ATTACK/DELAY";A
260 INPUT"SUSTAIN/RELEASE";B
270 INPUT"VOLUME";V
280 INPUT"WAVEFORM";W
300 S=54272:REM START ADDRESS
301 FORC=STOS+24:POKEC,0:NEXT
303 POKES+5,A:POKES+6,B
305 POKES+24,V:REM SETS VOLUME (COULD  BE FROM 0-15)
307 POKES+1,34:POKES,75
309 POKES+4,W:REM S+4 IS THE CONTROL SETTING FOR VOICE 1
311 FORD=1TO90:NEXT:REM THIS IS THE DURATION OF THE NOTE
312 POKES+4,32:REM THIS STOPS THE TRIANGLE WAVEFORM
313 FORD=1TO50:NEXT
320 PRINTCHR$(147)
325 PRINT"NOW LET US CHANGE EACH OF THE VARIABLES"
330 INPUT"TRY AGAIN";A$
332 IFA$="Y"THEN220
335 PRINTCHR$(147):PRINT:PRINT"OK. LET'S TRY SOMETHING ELSE"
340 PRINT"SO FAR WE HAVE USED ONLY ONE NOTE"
```

```
345 PRINT"LET'S TRY CHANGING PITCH TOGETHER WITH  THE OTHER
    VARIABLES"
350 INPUT"ATTACK/DELAY";A
360 INPUT"SUSTAIN/RELEASE";B
370 INPUT"VOLUME";V
380 INPUT"WAVEFORM";W
382 PRINT"EACH PITCH MUST BE SET WITH TWO NUMBERS"
383 PRINT"THE FIRST IS THE HIGH SETTING AND THE   SECOND THE
    LOW"
384 PRINT"      C  C#  D    D# E  F  F#  G   G#   A    A#  B
    C  C#"
385 PRINT:PRINT"HIGH: 34,36, 38, 40,43, 45, 48,51, 54,  57,
    61, 64, 68, 72"
387 PRINT:PRINT"LOW:  75.85.126.200.52.198.127.97.111.  172.
    126.188.149.169"
388 PRINT"CHOOSE A CORRESPONDING PAIR"
390 INPUT"PITCH";H,L
400 S=54272:REM START ADDRESS
401 FORC=STOS+24:POKEC,0:NEXT
403 POKES+5,A:POKES+6,B
405 POKES+24,V:REM SETS VOLUME (COULD  BE FROM 0-15)
407 POKES+1,H:POKES,L
409 POKES+4,W:REM S+4 IS THE CONTROL SETTING FOR VOICE 1
411 FORD=1TO90:NEXT:REM THIS IS THE DURATION OF THE NOTE
412 POKES+4,32:REM THIS STOPS THE TRIANGLE WAVEFORM
413 FORD=1TO50:NEXT
430 INPUT"TRY AGAIN";A$
432 IFA$="Y"THEN350
435 PRINTCHR$(147):PRINT:PRINT"OK. LET'S TRY SOMETHING ELSE"
440 INPUT"FIRST ATTACK/DELAY";A
441 INPUT"NOW THE SUSTAIN/RELEASE";B
442 INPUT"NOW THE VOLUME";V
443 INPUT"AND THE WAVEFORM  17,33,65";W
444 PRINT"AND NOW THE NOTE: CHOOSE FROM THE FOLLOWING:"
445 INPUT"C C# D D# E F F# G G# A A# B C";N$
450 IFN$="C"THENH=34:L=75
451 IFN$="C#"THENH=36:L=85
452 IFN$="D"THENH=38:L=126
453 IFN$="D#"THENH=40:L=200
454 IFN$="E"THENH=43:L=52
455 IFN$="F"THENH=45:L=198
456 IFN$="F#"THENH=48:L=127
457 IFN$="G"THENH=51:L=97
458 IFN$="G#"THENH=54:L=111
459 IFN$="A"THENH=57:L=172
460 IFN$="A#"THENH=61:L=126
461 IFN$="B"THENH=64:L=188
```

```
462 IFN$="C"THENH=72:L=169
500 S=54272:REM START ADDRESS
501 FORC=STOS+24:POKEC,0:NEXT
503 POKES+5,A:POKES+6,B
505 POKES+24,V
507 POKES+1,H:POKES,L
509 POKES+4,W
511 FORD=1TO90:NEXT
512 POKES+4,32
513 FORD=1TO50:NEXT
520 POKES+24,0
530 INPUT"TRY AGAIN";A$
532 IFA$="Y"THEN440
535 PRINTCHR$(147):PRINT:PRINT"OK. LET'S TRY SOMETHING ELSE"
540 INPUT"FIRST ATTACK/DELAY";A
541 INPUT"NOW THE SUSTAIN/RELEASE";B
542 INPUT"NOW THE VOLUME";V
543 INPUT"AND THE WAVEFORM  17,33,65";W
544 PRINT"AND NOW THE NOTE: CHOOSE FROM THE FOLLOWING:"
545 INPUT"C C# D D# E F F# G G# A A# B C";N$
550 IFN$="C"THENH=34:L=75
551 IFN$="C#"THENH=36:L=85
552 IFN$="D"THENH=38:L=126
553 IFN$="D#"THENH=40:L=200
554 IFN$="E"THENH=43:L=52
555 IFN$="F"THENH=45:L=198
556 IFN$="F#"THENH=48:L=127
557 IFN$="G"THENH=51:L=97
558 IFN$="G#"THENH=54:L=111
559 IFN$="A"THENH=57:L=172
560 IFN$="A#"THENH=61:L=126
561 IFN$="B"THENH=64:L=188
562 IFN$="C"THENH=72:L=169
610 S=54272:REM START ADDRESS
611 FORC=STOS+24:POKEC,0:NEXT
612 POKES+4,32
613 POKES+5,A:POKES+6,B
615 POKES+24,V
617 POKES+1,H:POKES,L
618 POKES+14,117:POKES+18,16
624 POKES+4,W
625 FORD=1TO90:NEXT
626 POKES+4,32
627 FORD=1TO50:NEXT
628 POKES+24,0
630 INPUT"TRY AGAIN";A$
632 IFA$="Y"THEN540
```

```
650 PRINTCHR$(147)
655 PRINT"A MORE COMPLEX APPROACH INCREASES THE   INTEREST OF
    THE SOUND"
660 PRINT"VIBRATO CAN BE ADDED BY USE OF     THE  OSCILLATOR"
670 FORD=1TO3000:NEXT
680 PRINT"FOR EXAMPLE: THE FOLLOWING PROGRAMME    SHOWS A
    SOUND EFFECT"
700 S=54272:FORC=0TO24:POKES+C,0:NEXT
705 POKES+14,6:POKES+18,16
710 POKES+3,7:POKES+24,143
720 POKES+6,240:POKES+4,65
730 F=10814
740 FORD=1TO200:F0=F+PEEK(S+27)*8.5
750 H=INT(F0/256):L=F0-H*256
760 POKES+0,L:POKES+1,H
770 NEXT
780 POKES+24,0
800 PRINT"OR TRY THIS ONE:"
802 FORD=1TO2000:NEXT
805 PRINTCHR$(147)
810 S=54272:FORC=0TO24:POKES+C,0:NEXT
820 POKES+1,130:POKES+5,9
825 POKES+6,1
830 POKES+15,8:POKES+24,225
840 FORC=1TO12:POKES+4,21
845 PRINTC;
850 FORD=1TO1800:NEXT:POKES+4,20
860 FORD=1TO1200:NEXT:NEXT
865 PRINTCHR$(147):PRINT:PRINT
870 PRINT"HERE'S ANOTHER ONE!"
880 FORD=1TO1000:NEXT
889 V=15
890 FORCH=1TO150
895 READA
900 S=54272:POKES+24,V:POKES,220:POKES+1,68:POKES+5,80:
    POKES+6,193
905 POKES+7,120:POKES+8,100:POKES+12,148:POKES+13,195
910 V=V-.10
911 FORD=1TOA:NEXT
912 DATA200,190,190,190,180,180,170,170,160,160,150,145,140,
    130,120,110,105,100
913 DATA98,96,94,93,90,88,86,84,81,80,78,73,70,65,61,57,53,
    48,45,41,36,32,29,25
914 DATA23,22,21,20,20,20,20,20,20,20,20,20,20,20,20,20,20,
    20,20,20,20,20,20,20
915 DATA11,11,11,11,11,11,11,11,10,10,10,10,10,10,10,10,10,
    10,10,10,10,10,10,10
```

```
 916 DATA10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,
     10,10,10,10,10,10,10,10
 917 DATA10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,
     10,10,10,10,10,10,10,10
 918 DATA10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,
     10,10,10,10,10,10,10,10
 920 POKES+4,129:POKES+4,128
 930 POKES+11,129:POKES+11,128
 935 NEXTCH
 940 PRINTCHR$(147)
 945 PRINT"THE NEXT SHORT PROGRAMME ILLUSTRATES WHAT HAPPENS
     WHEN RANDOM VALUES ";
 950 PRINT"ARE     POKED INTO BOTH THE VOICE AND THE A/D &
     S/R ADDRESSES"
 955 FORD=1TO3000:NEXT
 960 PRINT"THIS IS THE LISTING:-":LIST 1000-1040
1000 S=54272
1005 FORC=STOS+24:POKEC,0:NEXT
1010 FORI=1TO20
1012 Z=INT(RND(1)*255)+1
1013 FORV=1TO15STEP.25
1015 POKES+1,Z:POKES,Z/20
1018 POKES+5,Z:POKES+6,Z/3
1020 POKES+24,V
1021 NEXT
1025 POKES+4,17
1030 FORD=1TOZ*8:NEXT
1035 NEXT
1040 REM"*********TO HEAR THIS, TYPE     ●●GOTO1000●●● ******"
1050 PRINTCHR$(147)
1055 PRINT"THE NEXT SHORT PROGRAMME ADDS A SECOND   VOICE"
1060 FORD=1TO2000:NEXT
1062 PRINT"THE LISTING FOR THIS SECTION RUNS FROM  1050-1185"
1065 PRINT"TO SEE THIS PRESS RUN/STOP THEN 'LIST   1050-1185.
     TO CONTINUE TYPE GOTO"
1067 PRINT"1100"
1070 GETA$:IFA$=" "THENLIST1100-1185
1080 IFA$="C"GOTO1190
1100 S=54272
1105 FORC=STOS+24:POKEC,0:NEXT
1110 FORI=1TO20
1112 Z=INT(RND(1)*255)+1
1113 FORV=1TO15STEP.25
1115 POKES+1,Z:POKES,Z/10
1118 POKES+5,Z:POKES+6,Z/3
1125 POKES+4,17
1130 IFZ=10THENZ=100
```

```
1155 POKES+8,Z+8:POKES+7,Z/10
1158 POKES+12,Z+4:POKES+13,Z/5
1170 POKES+24,V
1175 POKES+11,17
1176 NEXT
1185 NEXT
1190 PRINTCHR$(147):PRINT"AND NOW FOR SOMETHING DIFFERENT"
1200 S=54272
1205 FORC=STOS+24:POKEC,0:NEXT
1210 FORI=1TO15
1212 Z=INT(RND(1)*255)+1
1213 FORV=1TO15STEP.25
1215 POKES+1,Z:POKES,Z/20
1218 POKES+5,Z:POKES+6,Z/3
1225 POKES+4,17
1230 IFZ=10THENZ=100
1255 POKES+8,Z+8:POKES+7,Z/6
1258 POKES+12,Z+10:POKES+13,Z/5
1260 POKES+14,Z+4:POKES+15,Z/4
1262 FORD=1TO10:NEXT
1265 POKES+16,Z+8:POKES+17,Z/2
1268 POKES+18,64
1270 POKES+24,V
1275 POKES+11,33
1276 NEXT
1280 FORD=1TOZ*9:NEXT
1285 NEXT
1290 PRINTCHR$(147):PRINT"EVEN MORE COMPLEX--THE LISTING RUNS
     FROM 1300-1385"
1300 S=54272
1305 FORC=STOS+24:POKEC,0:NEXT
1310 FORI=1TO12
1311 Y=INT(RND(1)*127.5)+1
1312 Z=INT(RND(1)*255)+1
1313 FORV=1TO15STEP.25
1314 IFZ<20THENZ=20
1315 POKES+1,Z:POKES,Z/20
1316 IFY<5THENY=5
1318 POKES+5,Y:POKES+6,Y/3
1325 POKES+4,17
1355 POKES+8,Z+8:POKES+7,Z/10
1358 POKES+12,Y+10:POKES+13,Y/5
1360 POKES+14,Z+4:POKES+15,Y/4
1362 FORD=1TOY:NEXT
1365 POKES+16,Y+8:POKES+17,Z/2
1368 POKES+18,64
1370 POKES+24,V
1375 POKES+11,33
```

```
1376 NEXT
1380 FORD=1TOZ*9:NEXT
1385 NEXT
1390 PRINTCHR$(147):PRINT"AND FOR MORE VARIETY---"
1400 S=54272
1405 FORC=S+24TOSSTEP-1:POKEC,0:NEXT
1406 FORI=1TO10
1410 Y=INT(RND(1)*127.5)-1
1412 Z=INT(RND(1)*255)-1
1415 POKES+1,Z:POKES,Z/20
1418 POKES+5,Y:POKES+6,Y/3
1425 POKES+4,17
1427 IFZ=<20THENZ=20
1430 FORX=1TOZ/5
1455 POKES+8,X:POKES+9,X/10
1456 POKES+21,Z-13
1457 POKES+22,X+2:POKES+23,Y/2
1458 POKES+12,Y+10:POKES+13,Y/5
1460 POKES+14,Z+4:POKES+15,Y/4
1461 NEXTX
1462 FORD=1TOY:NEXT
1465 POKES+16,Y+8:POKES+17,Z/2
1468 POKES+18,64
1470 POKES+24,10
1475 POKES+11,33
1476 NEXT
1480 FORD=1TOZ*9:NEXT
1490 POKES+24,0
1500 PRINTCHR$(147)
1505 PRINT"THE PROGRAMME FROM LINE 1000 WILL GIVE  A
     DIFFERENT RESULT EACH TIME"
1520 PRINT"LOOK AT THE LISTING OF THE PROGRAMME,   TAKE
     PORTIONS OF IT AND ALTER IT"
1530 PRINT"TO SEE THE VARIOUS EFFECTS"
1540 FORD=1TO10000:NEXT
1550 POKES+24,0
1600 S=54272
1605 FORC=S+24TOSSTEP-1:POKEC,0:NEXT
1606 FORI=1TO10
1610 FORX=1TO60:READA:NEXT
1615 POKES+1,A:POKES,A
1618 POKES+5,A:POKES+6,A
1625 POKES+4,17
1655 POKES+8,A:POKES+9,A
1656 POKES+21,A
1657 POKES+22,A+2:POKES+23,A
1658 POKES+12,A:POKES+13,A
1660 POKES+14,A+4:POKES+15,A/4
```

```
1662 FORD=1TOA:NEXT
1665 POKES+16,A+8:POKES+17,A/2
1668 POKES+18,64
1670 POKES+24,10
1675 POKES+11,33
1676 NEXT
1680 FORD=1TOA*4:NEXT
1690 POKES+24,0
1691 DATA30,200,40,45,211,231,124,154,11,16,23,14,37,95,65,
     47,43,78,156,235,154
1692 DATA47,35,125,231,145,197,27,59,96,75,68,199,200,201,
     234,105,112,56,24,35
1693 DATA30,200,40,45,211,231,124,154,11,16,23,14,37,95,65,
     47,43,78,156,235,154
1694 DATA47,35,125,231,145,197,27,59,96,75,68,199,200,201,
     234,105,112,56,24,35
1695 DATA30,200,40,45,211,231,124,154,11,16,23,14,37,95,65,
     47,43,78,156,235,154
1696 DATA47,35,125,231,145,197,27,59,96,75,68,199,200,201,
     234,105,112,56,24,35
1697 DATA30,200,40,45,211,231,124,154,11,16,23,14,37,95,65,
     47,43,78,156,235,154
1698 DATA47,35,125,231,145,197,27,59,96,75,68,199,200,201,
     234,105,112,56,24,35
1699 DATA30,200,40,45,211,231,124,154,11,16,23,14,37,95,65,
     47,43,78,156,235,154
1700 DATA47,35,125,231,145,197,27,59,96,75,68,199,200,201,
     234,105,112,56,24,35
1701 DATA47,35,125,231,145,197,27,59,96,75,68,199,200,201,
     234,105,112,56,24,35
1702 DATA30,200,40,45,211,231,124,154,11,16,23,14,37,95,65,
     47,43,78,156,235,154
1703 DATA47,35,125,231,145,197,27,59,96,75,68,199,200,201,
     234,105,112,56,24,35
1704 DATA30,200,40,45,211,231,124,154,11,16,23,14,37,95,65,
     47,43,78,156,235,154
1705 DATA47,35,125,231,145,197,27,59,96,75,68,199,200,201,
     234,105,112,56,24,35
1706 DATA47,35,125,231,145,197,27,59,96,75,68,199,200,201,
     234,105,112,56,24,35
1707 DATA47,35,125,231,145,197,27,59,96,75,68,199,200,201,
     234,105,112,56,24,35
1708 DATA47,35,125,231,145,197,27,59,96,75,68,199,200,201,
     234,105,112,56,24,35
1709 DATA47,35,125,231,145,197,27,59,96,75,68,199,200,201,
     234,105,112,56,24,35
1710 DATA47,35,125,231,145,197,27,59,96,75,68,199,200,201,
     234,105,112,56,24,35
```

```
1711 DATA47,35,125,231,145,197,27,59,96,75,68,199,200,201,
     234,105,112,56,24,35
1712 DATA47,35,125,231,145,197,27,59,96,75,68,199,200,201,
     234,105,112,56,24,35
1713 DATA47,35,125,231,145,197,27,59,96,75,68,199,200,201,
     234,105,112,56,24,35
1714 DATA47,35,125,231,145,197,27,59,96,75,68,199,200,201,
     234,105,112,56,24,35
1990 PRINTCHR$(147)
1999 END
2000 S=54272:REM START ADDRESS
2001 FORC=STOS+24:POKEC,0:NEXT
2002 REM LINE2001 CLEARS THE SOUND CHIP
2003 POKES+5,9:POKES+6,0
2004 REM LINE 2003 SETS ATTACK & DELAY - (S+5)
     SUSTAIN & RELEASE
2005 POKES+24,12:REM SETS VOLUME (COULD  BE FROM 0-15)
2006 REM  SETS NOTE FOR VOICE 1
2007 POKES+1,34:POKES,75
2008 REM S+1 SETS THE HIGH FREQUENCY AND S THE LOW
2009 POKES+4,17:REM S+4 IS THE CONTROL SETTING FOR VOICE 1
2010 REM SECOND NUMBER SETS WAVEFORM
2011 FORD=1TO50:NEXT:REM NOTE DURATION
2012 POKES+4,32:REM WAVEFORM STOPPED
2013 FORD=1TO50:NEXT
2014 GOTO2007:REM DO IT AGAIN
2015 REM  ******** TO CONTINUE THE PROGRAM
     TYPE 'GOTO140 '**************
```

Bring the cursor back up to the first line and then press the return key to hear full volume; then press the return key again, and the volume will be reduced to half once more.

Now bring the cursor back up to your first line, move it along to the value of S+1, and change the value to 85 or 99 and hear what happens.

Now, either by means of typing in a new line or by editing what you already have, produce the following single line program in direct mode:

```
S=54272:FORX=1TO10:FORI=1TO100:
POKES,I:POKES+1,I:POKES+24,12:NEXT:
NEXT
```

This program says to perform the sequence ten times; incrementing the value of I and thus moving the frequency upwards each time. The result, as you can hear, is a series of swoops.

Now change the FORI loop to read:

```
FORI=150TO1STEP-1.
```

Now the swoops are downwards.

Let's try something else:

```
10  S=54272
15  FORX=15TO1STEP-1
20  FORI=1TO150:POKES,I:POKES+1,I:
    POKES+24,X:NEXT:NEXT
```

You might wish to put each statement on a

different line for easier manipulation. You could try swapping the for-next loops around, in which case you will hear something quite different. Experiment with this program as much as you wish, changing the upper limit values of the FORI loop. Beyond a certain figure you will get an illegal quantity error. Then change the lower limit to start the sound at a higher pitch.

You have concerned yourself only with the voice 1 and the volume registers. The voice 2 registers are found at S+7 and S+8. Try adding them to your program. For example:

```
10S=54272
15FORX=15TO1STEP-1
20FORI=100TO250
30  POKES,I:POKES+1,IPOKES+24,X
35  POKES+7,I/2:POKES+8,I/2
40  NEXT:NEXT
```

When you have heard that enough times, change the value for the second voices (S+7 and S+8) to I/3, or even 3.5. Then add the pokes for voice 3 at S+14 and S+15.

```
36POKES+14,I/2.5:POKES+15,I/2.5
```

You will notice that the program runs a little slower each time, but there are three distinct pitches being generated at once.

That program will probably drive everyone mad, so let's change it quickly. Get rid of line 20 to start with, and then remove the unnecessary NEXT in line 40 (line 20 is the FOR that goes with it). Now you have a series of rapid taps. There is no pitch because there is no value for I being generated. Let's put one in . . . in fact let's put a series in!

```
12  I=10
20  I=I+5
```

And not put in: 39 FORD=1TO50:NEXT

```
10  S=54272
20  POKES,75:POKES+1,34:POKES+24,12
30  POKES+7,52:POKES+8,43
35  POKES+14,97:POKES+15,51
40  FORD=1TO2000:NEXT
100  POKES+24,0
```

There are other parameters that can be changed too. A note does not just sound. It has a beginning, a middle and an end. Each of these factors, the way the sound behaves at each point determines the character of the sound. You can control these factors on the C-64, just as in the very best music synthesizers.

There are two important features you must concern yourself with in regard to the sound: *attack/decay* and *sustain/release*. The first combination covers the very opening and closing of a sound; a violin being plucked (pizzicato) produces a very different sound from one which is being bowed (arco). The former displays a very rapid attack and decay rate. When played with a bow, the violin can begin very smoothly and the sound can be allowed to tail off gradually too. Of course, it is also possible to produce short, sharp sounds with a bow on a violin, but this would perhaps be obvious only to an expert.

Each voice has its own A/D address. For voice 1, this will be S+5, or 54277. To control the rates of attack and decay, you must set each of the eight bits in that location in much the same way that you set each bit in order to produce user-programmed characters. The difference is that you hear the result rather than see it. The four high bits attend to the attack and the four low ones to the decay. The four high bits are, in order 128,64,32 and 16; the four low bits ar 8,4,2 and 1. Let's try a short program, using only one voice, in which you can alter the A/D rates. I also added the sustain/release rates while I was about it, for the sake of completeness.

```
4  PRINTCHR$(147)
7  PRINT"SUSTAIN/RELEASE:128TO1"
8  PRINT"ATTACK/DECAY:128TO1"
9  INPUT A,B
10  V=54296:HF=54273:LF=54272
    :AT=54277:SR=54278:W=54276
20  POKEV,12:POKEHF,68:POKELF, 149
```

```
25  POKEAT,B,POKESR,A
30  FOR1=1TO1000:NEXT
40  POKEV,0
40  GOTO4
```

Now run the program putting in values given above; high values for A and the low ones for B. Not very dramatic, is it?

Perhaps you should add a little more spice to the mixture in the form of waveform. This is controlled at address 54276 or S+4. There are four settings here, which are:

```
17, 33, 65 AND 129
25 POKE W, 17
```

Add the above line and the difference is noticeable immediately. You have added some character to the affair. Perhaps you might like to be able to alter this at will too, in which case alter the program as follows:

```
9   INPUT A,B,C
27  POKEW,C
```

Note that it doesn't matter to the computer that C is used before A or B. Confine yourself to settings 17 and 33 for C for the moment, and stick to the same settings of A/D and S/R until you can hear differences.

The values for A/D and S/R can be combined. For example you can set S+5 at 130, which is 128+2 or the highest A setting and the second lowest D setting. You can combine more than two if you wish, for example: 128+32+16+8+4+1=189 Whatever you do, you cannot fail to notice the rounder, fuller sound of waveform setting 17 against the reedier, thinner sound of setting 33. Setting 129 produces noise and setting 65 seems to do nothing..... yet!

Now add some of the other fancy bits from the earlier examples to produce various effects. Meanwhile, just to set you wondering, give C the value 22 and see, or rather hear what you get! Then try 38. What you are doing here is combining another feature, or two bits, of address 54276 (S+4). But more of that later. The chief thing about it all is that the variety is close to infinite. It is possible to write a program that rings the changes through all the parameters, thus producing true computer generated sounds. Whether or not you call it music is up to you, personally I feel that it requires conscious effort and some considerable amount of ability to write music. There are already too many note-spinners in the world, many of them writing utter rubbish—although it occurs to me that perhaps the computer might be the means of putting the peddlers out of business!

You will find claims that the C-64 is capable of producing, or reproducing the sound of any instrument that exists. That may be true; however, the programming to do so would be immense. Suggestions that such and such a setting produces the sound of a harpsichord are little short of wildly imaginative. As a performer on that instrument and a builder of some two dozen of the beasts, I think I have a fair idea of what the sound should be. Of course, it is a little unfair to listen to the C-64 through a 90¢ speaker of the type found on most TV's!

The real value of the C-64 sound feature is the ability to produce sounds that have not been heard before, learning how to shape and control whole sound atmospheres. And sound effects for games are good fun too!

In case you wish to play with such a program as above without having to remind yourself about values, enter and save the program below (On the disk and tape it is called Notes.)

```
1 REM:NOTES
4 PRINTCHR$(147)
5 PRINT"DURATION: ANY REASONABLE
  FIGURE"
6 PRINT"WAVEFORM:17,33,65,129"
7 PRINT"SUSTAIN/RELEASE:128TO1"
8 PRINT"ATTACK/DECAY:128TO1"
9 INPUTA,B,C,D
10 V=54296:HF=54273:LF=54272:
   AT=54277:SR=54278:W=54276
20 POKEV,12:POKEHF,68:POKELF,149
25 POKEAT,D:POKESR,C:POKEW,B
30 FORI=1TOA:NEXT
```

```
40 POKEV,0
50 GOTO4
```

If you take a look at the program Music in Fig. 7-1 you will see a curious line FOR C=STOS+24:POKEC,0:NEXT. This line clears out the sound chip, wiping the slate clean for each program. The sound chip seems to have a nasty habit of holding on to what it has! Add the line to the above program.

Now perhaps you are ready to try your hand at changing pitch in a more regular fashion.

Look at the Music program, lines 300-390. Here you can see the values for the low and high bytes that must be entered for each note. There is only one octave given here. A full set of values can be found in the appendix in the *User's Guide*. If the noise gets on your nerves while you are typing in new sections of a program, just turn the sound down on the TV, or POKE 0 into S+24.

THE PRINCIPLES OF SOUND PRODUCTION

As you will know from your physics lessons in school, sound travels in waves. Waves are really nothing more than alternate compressions of air. The way the violin string vibrates determines the shape of the wave. A lute or guitar string, although appearing to vibrate in the same way as the violin string, actually behaves in quite a different way. This is partly due to the greater length of the guitar string and partly due to the fact that it is not held in such great tension as the violin string is. The fact that a guitar string is not bowed also has something to do with the resulting sound.

The sound chip on the Commodore 64 allows you to alter all the parameters of a sound such that any instrument can be reproduced. When a string vibrates it tends to divide itself a number of times. It not only vibrates along its whole length but also in halves, thirds, sixths, twelfths and so on. Some of the vibrations, the distance the portion of string moves from side to side (we call this *amplitude*), are larger than the others. Each vibration produces a different note. Thus an instrumental sound is not just one note but several sounding at the same time. Because the different notes sound at different volumes (a result of the amplitudes) all at the same time, the note becomes colored in a particular way. If you have an acoustic guitar (preferably of the classical type with nylon strings), you can hear how the sound can be changed by first of all plucking the string over the rose hole; then plucking it very near the bridge, producing a much thinner sound; and then plucking it well above the rose hole, producing a fuller and slightly duller sound.

The first plucking point produces the most variety in vibration. The sounds from the additional vibrations are called *harmonics*. The second plucking point, near the bridge produces more short-length vibrations than long ones. Thus the sound is richer in the higher harmonics than in the lower ones. The third plucking point produces more of the longer vibrations than the shorter, and thus produces more of the lower harmonics than the higher. The sound is not so thin.

An oboe is rich in the upper harmonics, as are the violin and the trumpet. The clarinet has fuller lower harmonics as does the French Horn; that's what gives the full rich sound to both those instruments. The piano is a dull instrument by comparison. The harpsichord is rich in harmonics: 8 or 9 of them. The clavichord, a type of early piano (and my favorite keyboard instrument both to make and play) is richest of all.

It is of great importance to be aware of the effect of volume when trying to imitate the sound of an instrument. An oboe is a good bit quieter than a trumpet, although a trumpet can play quietly. However, it is not enough just to reduce the volume for the quiet trumpet, as opposed to the loud one; you must also change the timbre or character of the sound a little; otherwise, the result will sound nothing like the instrument.

The reproduction of sounds, whether they be of conventional or unconventional instruments, is an art. The subject has not yet been fully explored. In fact, only a small start has been made. The Commodore 64 will, I am confident, do a great deal in fostering and furthering an interest in electronic music, by placing at everyone's fingertips a device which is, although inexpensive, very versatile indeed! The limits of the device are nothing more than those of the imagination.

In addition to the harmonic structure of a note, you must concern yourselves with the continuing behavior of the sound as it progresses from its beginning to its end. Take the piano, for example: A piano is a percussive instrument; the sound is produced by means of a hammer that strikes a taughtly drawn string (actually two or three strings). The note begins suddenly, quickly reaches peak volume and then continues, decaying all the time until the sound dies on its own or until the damper is allowed to fall on the string. The sound might be described as an initial boing followed by a ring.

The volume is varied by the speed at which the hammer strikes the string, and that is controlled by the speed that is used to strike the key.

An oboe, on the other hand, begins with a short sharp puff of air. However, instead of dying away, the sound remains at a constant level. The oboe, like any instrument which is blown or bowed, can produce sounds that can vary in volume during the course of the sound. A note can begin quietly and become louder! It can begin quietly, get louder, get softer, get louder again, and then finally die, all under the control of the player. Any attempt to reproduce the sound of a wind or bowed string instrument which does not include variable volume will be quite unsatisfactory, and cause the C-64 to behave no better than an electronic organ!

The total shape or behavior of a note is called the *envelope*. The envelope is controlled by the ASDR addresses. The harmonic structure is controlled by the filters, which are found at address locations 54293 to 54296, or S+21 to S+24. You will recognize S+24 as the volume control.

As each address contains or consists of one byte made up of 8 bits, a good deal of flexibility can be allowed. At address S+24, the lower four bits serve to control the volume in a range 0 to 15. You can see how this is done by working out the decimal equivalents of the binary number formed by the 8 bits:

00000000=0 decimal.....volume level is at 0
00000001=1 decimal.....volume level is at 1
00000010=2 decimal.....volume level is at 2
00000011=3 decimal.....volume level is at 3
00001111=15 decimal.....volume level is at 15 or full

The four upper bits control other features such as filter high and low pass, which allow either high or low harmonics to sound or be suppressed.

The short program sections found in Music can all be added to, making use of the various addresses from 54272 to 54296. A list of them and what each controls appear in the summary to this chapter.

Any value for any of the addresses can be retrieved from data lines which are read. Thus, pitch values, volume values, values for duration, A/D, S/R, filtering, and waveform can be stored in a small or large bank of data. A subroutine type program consisting only of the BASIC commands can be used over and over again with different sets of data. Its rather like the program itself being the stereo unit and the data lines being the record album.

CAPSULE REVIEW

Sound on the C-64 is controlled from address locations 54272 to 54296. There are three voices each of which reproduce 123 notes of the western chromatic scale. They can also, of course, produce pitches that fall in between those of the conventional western system and thus allow for music of other cultures such as Chinese or Indian.

The voices are obtained by poking the following addresses:

54272, 54273,	low and high byte frequency control	Voice 1
54279, 54280	low and high byte frequency control	Voice 2
54286, 54287	low and high byte frequency control	Voice 3

Envelope generators are found as follows:

54277	Attack/Decay	Voice 1
54278	Sustain/Release	Voice 1
54284	Attack/Decay	Voice 2
54285	Sus/Rel	Voice 2
54291	Att/Dec	Voice 3
54292	Sus/Rel	Voice 3
54276	Waveform	Voice 1

54283	Waveform	Voice 2
54290	Waveform	Voice 3
54293-54296	Filters/Volume	All voices.

It is easiest to set S=54272 and work the remaining address by adding numbers 1 to 24 for each. The values to be poked can be placed directly or be subject to variable control. They can also be randomly generated or retrieved from data lines.

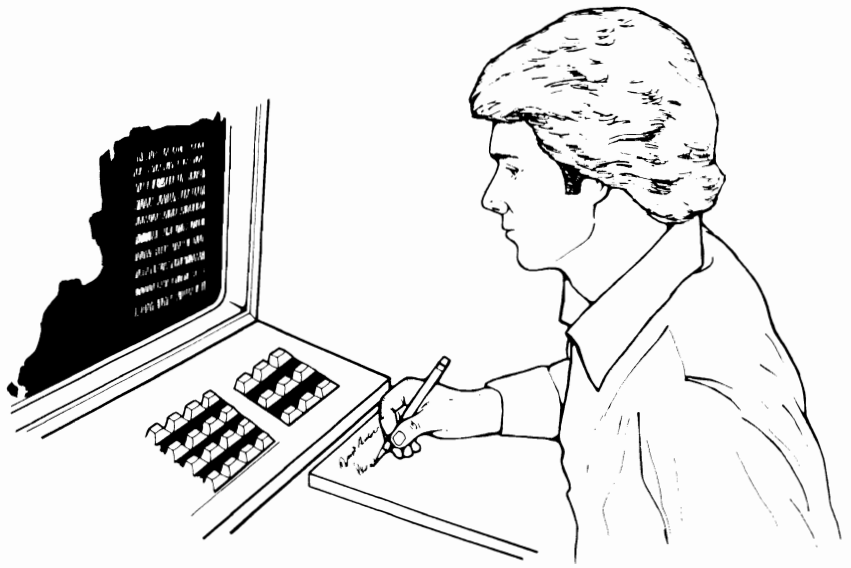
The values poked into the addresses refer to the various bits at those addresses, eight in each. The values are the decimal equivalents of the binary value of each bit. A/D and S/R values can be combined to produce complex structures.

The low and high byte values for pitch are poked into the voice addresses. They can be stored as combination values assigned to variables that correspond to the conventional names of notes. E.g. IFN$="C"THEN H=34:L=75 where H = high and L = low byte.

A phenomenon that you will observe is that of resultant tones. When two notes are sounded together, there is often a third note generated. If the third note corresponds to a harmonic found in the first two, the resultant tone is amplified. Clearly it is possible, though confoundedly complex, to produce four or even five voices even though only three are programmed. The principle has been used not only in electronic organs but also in pipe organs to produce notes lower than those otherwise obtainable from the longest pipe fitted in the organ.

Programming music, whether it be conventional or avant grade, requires patience and, contrary to popular statements, a good knowledge of music theory. I regret to have to say that it is not possible for an unskilled person to generate other than musical or tonal doodling except by accident; and formal music is far from being accidental!

Chapter 8



In the late sixties, I was developing a system for schools that would take advantage of the (then) coming computer revolution. One of the aspects that the system addressed was that of information retrieval or IR. The idea, as expressed at that time, was to make use of a system that would allow an individual to access whatever information he or she desired in much the same way that one shopped in a supermarket.

It was mooted that there was a browse mode, rather like walking between the rows of shelves. One could pause and examine items and then make use of those bits and pieces of information that one desired. All that was far off in the future and was thought to be too expensive to concern ourselves with at the time.

Now, however, you, as the owner/user of a C-64 have that power at your finger-tips—quite literally so.

USING THE VIC MODEM

The device that allows you access to this power is called a *modem*, which is short for modulator/demodulator. It is a device which is inserted into the user port at the back of the C-64 and attaches directly to your phone line. It translates signals from the computer, transmits them over the line so that you can talk to other computers, either those of your friends or those operated by information banks. Believe that the world is now on your TV screen as it never was before!

The modem can be purchased separately, although it is possible that you bought one as part of the various packages that Commodore has been marketing. The modem itself is a brown cartridge. Insert it with the name on top, in fact it is impossible to do otherwise! Into the modem you must insert the small phone adaptor. Into the adaptor you must insert the white cable. You will find two slots at the back of the phone adaptor. The white cable goes into the one marked TELCO. The cable from the phone is removed from the wall jack and inserted into the slot marked TELSET. Now insert the other end of the white cable into the wall jack.

On the phone adaptor you will find a switch. Make sure it is in the position marked V. Check that your phone gives you a dial tone! On the modem itself, you will find another switch. Move it to the position marked 0 (for originate).

In the box with the modem, you will find an instruction book, a packet of other papers and a tape. The tape contains a program that turns your computer into a terminal. One side of the tape is for the VIC-20 and the other for the C-64. Make sure you have inserted the tape correctly into the Datasette, if necessary, rewind it to the beginning and then type LOAD"".

If you have a disk drive it might be a good plan to put a copy of the terminal on disk. It is a good idea to have a back-up copy in any case.

When the screen shows ready, enter RUN. You will find that the characters are barely legible. Don't try to do anything about that at the moment, just enter F4 (press the shift and F3 keys). You will now see a quite legible menu. Baud rate, duplex, parity, stop bits, and word length might sound either rude or confusing to you, and you might think that you have to do a lot of study before going further, but you don't!

Baud rate will probably be set at 300, duplex at FULL, Parity at NONE, Stop bits at ONE, and word length at 8. If your screen does not show this, press the first letter of the item you wish to change. The item will change to inverse characters. Now move the cursor along to the setting required (the settings will revolve, when you get to the end, the cursor will move back to the first one, changing it into inverse characters).

When you have done this press N, for next page. There is no need to press the return key for this one.

A new menu appears. You may need to make one or two changes. Press L and press the return key. This now gives you a line feed and carriage return. Forget the next two options for the time being, but press F.

Now, before you go any further, press E to get back to BASIC. This puts you right back to the beginning requiring you to reload the terminal program, run it, and go through all the paces you have just performed. Don't be annoyed with me for making you do this. Rather, thank me for the practice. Even go to the length of doing it two or three times so that you can do it with confidence. In that way you will not have to refer to this section of the chapter too many times in the future!

Now press T. Again you will see a barely legible remark at the top of the screen. Now you can change something. Pressing the CTRL key, tap #8 five times. This will change the colors of the screen each time until you reach black. Keep on tapping if you wish, but try to bring yourself back to black. I have found that this produces a screen that is fairly easy on the eyes. Pressing the CTRL key and 1 changes the border color. Pressing the CTRL key and 5 changes the character color but will seem to have no effect whatsoever at first.

Now look for two items in the package: a light blue-grey pamphlet called VICMODEM Information Package and a CompuServe Information Service snappack. Open the pamphlet to page 3 and undo the snappack, removing the contents.

Your local telephone directory should give the number of either CompuServe, or failing that, DATAPAC. There should be three telephone numbers, one for each BAUD rate. Choose the number that corresponds to 300 baud. Dial it, and as soon as you hear the high pitched tone, move the switch on the phone adaptor, not the modem to D and hang up the phone.

Now refer to page 3 of the pamphlet. If you are connected to CompuServe, move directly to instruction 8. It is more than likely, however, that you will be connecting with DATAPAC and must therefore go through the whole procedure.

The next operation can be one of total frustration due to the totally inadequate instructions provided in the pamphlet, VIC Modem instruction book or any other sheet of paper provided!

First type a period and then press the return key. The response should be DATAPAC and some numbers. Next type in 2:1 and press the return key. Ignore both the word *set* that precedes 2:1 and the message that comes on the screen. It will say, (at least on mine it says): comma required before character data.

Now, any rational creature would think that this was an error message and that they had done something wildly wrong! Not so! Keep right on typing the next instructions!

Eventually you will reach the point where you are asked for your ID. Read it carefully from the bottom of the contents of the snappack. Do likewise with the password. Make sure you type everything exactly as you see it.

As you will observe as you scramble further inside the tangled web of the computer world, all is not well in the field of instruction writing. The majority of those who write about computers assume a fairly thorough knowledge on the part of new users. The truth is, of course, that nobody is born knowing about computers, that everybody must learn, and that everybody is entitled to information in as quick and as easy a fashion as possible. Of course, information services must ensure that unauthorized users cannot gain access, but that is not quite the same thing as creating utter confusion among those who would wish to make use of services. You would think these organizations would be glad of new customers, wouldn't you?

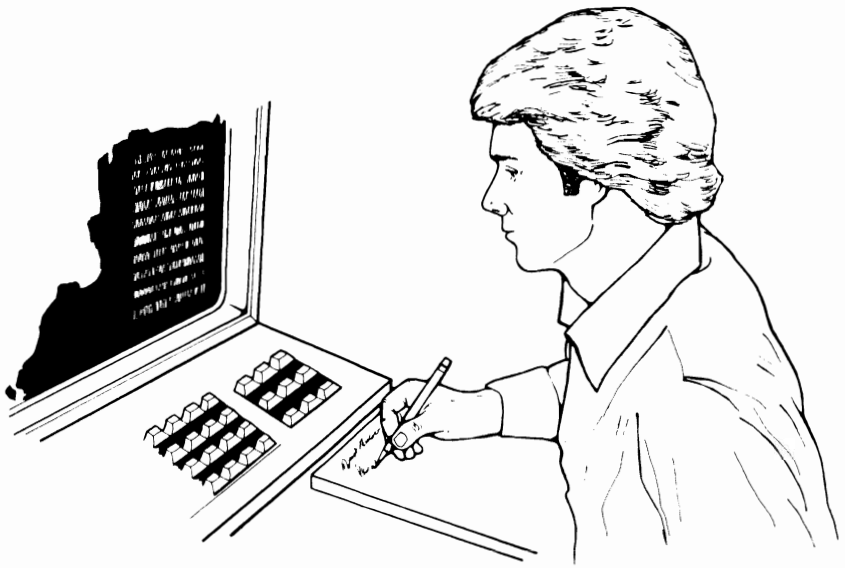
Once logged on, you can make your choice among the services. It is my recommendation that you type GO CBM at the first prompt. This will give you a good idea of what the service is all about, and it includes a Commodore users' survey that you might like to complete.

CompuServe also provides a variety of information, including a section on employment with CompuServe itself. You will note that the display stops from time to time. This is because the host computer only seems to be dealing with you as an individual. In fact it is dealing with a lot of users at one and the same time. It gives you a little bit of time, and then deals with someone else, then someone else, and eventually comes back to you. How long you have to wait will depend upon the number of users. Note that after your first free hour you will not be able to access any service until you have signed up with the service.

Dow Jones provides a thorough service that is mainly financial but includes weather and sports; General Videotex Corporation also provides a rounded service with access to an encyclopedia. In Canada, Comp-U-Store allows for a complete shopping service at home, no less! In addition you can access newspapers from other cities as soon as they appear on the street! Just imagine living in New York and being able to read the Guardian at 10 p.m. the day before it appears! (Nothing to do with relativity—just with the time difference!)

Returning for the moment to the CTRL 5 feature: you will notice that nothing seems to happen when you press that control. You will notice the difference as soon as the next set of characters appears on the screen. Play with it until you find a combination of background and character colors that is easy on your eyes. You can even change the combination while the service is in use, giving your eyes a rest from one single pattern. By changing the border to the same color as the screen, you can ease eye-fatigue considerably.

Chapter 9



There is no question about it! A printer is a very useful device. If your programs become longer than can be contained on one screen, a printer becomes almost essential. The ability to gaze at a complete printout of a program rather than just one screenful at a time is a boon.

At one time the only way to deal with a computer was via a printer. They were very slow, unless you used one of those very expensive line printers that could print a complete line at one time, and made a fair amount of noise. Nowadays printers are small but very flexible in their operation, and are becoming less expensive everyday.

The printer marketed by Commodore to match your C-64 carries the number VIC1525. This is important for there is also the 1515, which is designed to run only with the VIC-20 computer. The VIC1525 will run with either the VIC-20 or the C-64.

Commodore have chosen to make their printer machine-specific. The 1525 will only run with the two machines mentioned above. No other printer will run with either machine unless there is a special interface provided. An interface is the electronic circuitry that allows one device to understand the instructions of another.

It is possible to buy a *parallel* interface that will plug into the back of the C-64 and run a great variety of printers with varying degrees of efficiency. Using the 1525 is much easier, however, for there is no need to perform a great deal of programming in order to do the simplest tasks.

You will have noticed, no doubt, that there are a lot of machines on the market that look very much like the 1525 printer. Commodore has chosen a particular machine and put their label on it, as well as making it part of their intelligent peripheral series. The 1525 is the same machine as the Seikosha GP100 and, indeed, the latter printer will work very well with the C-64 (with the correct interface). A better machine is the Seikosha GP 250X. It has true lower-case descenders (the little bits that dangle below the line in letters such as g and y) and can be altered to produce German and

Swedish character sets without the need to use its dot addressable graphic feature.

USING THE PRINTER

Complete instructions on how to set up the printer are contained with the machine. Do follow the precautionary advice very closely or else the printer may be damaged before you experience the joy of using it.

The printer plugs into the special socket at the back of the C-64. There is only one socket that will match the printer cable. The plug can be inserted in only one way. The other end of the cable plugs into the printer. The printer is powered via the ac cable which is nonremovable. Do not switch any part of your equipment on until every detail has been attended to!

Insertion of the ribbon might seem a little awkward at first, but with care the two plastic housings will seat themselves neatly on the extreme ends of the carriage bar without either coming off or covering your hands with ink.

The control inside the machine which controls the intensity with which the print head strikes the paper also deals with paper thickness. The manual does not tell you this.

On the back of the printer, there is a small switch that allows you select the device number of the printer. You will recall that each peripheral has an address number; the disks can be either 8 or 9, for example. The device number for the printer can be either 4 or 5. It is conceivable that you might at one point in time have two printers and will therefore wish to indicate which machine should print at any point in time.

If you are using a disk drive, the printer plugs into that instead of directly into the C-64. Switch on the printer and the disk-drive before switching on the computer!

In the same way that you must send specific commands to the disk drive in order to get any action, so must you send specific instructions to the printer. The command is OPEN.

Normal operation requires that the various commands be sent from within a BASIC program and that is what you will try now. Enter the following on your C-64:

```
10 OPEN 1,4
20 CMD1
30 PRINT "PRINTER TEST"
40 LIST
```

Now type the command RUN. The printer will begin by producing the text PRINTER TEST and then follow it with a listing of the short program. You might at this point like to load a short program of yours from either tape or disk and then, in direct mode enter the following:

```
PRINT#1
```

and then press the return key. The printer will chatter out the message READY. Then enter:

```
OPEN1,4:CMD1:LIST
```

The result should be that fine program you wrote, neatly printed for the world to see and admire. When it is all over enter

```
PRINT#1
```

and the computer will respond with READY.

Quite simply, you have instructed the computer to tell the printer, which is device 4, to print the text and then list the program. The 1 after OPEN is an arbitrarily chosen "file" number between 0 and 255 to which all following statements must refer. The PRINT#1 is a message to the printer telling it that printing is over. The printer has a buffer in which the data being sent is stored. You must let the printer know that there is no more to come for the moment.

A more complete program would include the command CLOSE1, which closes the file called 1. Note that the number used in CMD and CLOSE must match the one in OPEN.

If you wish to have more than one set of data sent to the printer, each set must have its own file number. For example OPEN1,4:OPEN3,4 would

mean that two files, one labeled 1 and the other 3, are available for access. Should you then wish to send only the contents of file 1 you would enter CMD1. When all is finished you would enter CLOSE1. Then you would be free to use CMD3 and print the contents of that file. Do not forget to use PRINT# and the number of the file you wish to close in order to make sure that nothing is left in the printer buffer.

Now let us explore some of the versatility of the printer. Enter the following:

```
10 REM:PRINTER FIG IX-2
20 OPEN1,4
25 CMD1
30 PRINT#1,CHR$(14)"PRINTER TEST"
32 PRINT#1,"EXPANDED MODE"
35 PRINT#1,CHR$(15)
40 LIST
```

The printer will now produce the text PRINTER TEST EXPANDED MODE, but in double width characters. Note that it will not list the program this time for there is no direct command for it to do so. The commands have been concluded with PRINT#1, CHR$(15). Actually, all the CHR$(15) does is to change the mode back to normal size characters, but the PRINT# statement has closed off the other commands. If you want to see what happens when a file has not been closed, just enter LIST once more before you enter PRINT#1.

As you can see, many of the features of the printer can be called up using the correct CHR$ code. This is often quicker than entering the desired mode by using a symbol inside quotation marks. Expanded print is obtainable only by use of CHR$(14) as it happens, but some others, such as reverse mode, can be achieved by using both *quote mode entry* and the CHR$ code. For example:

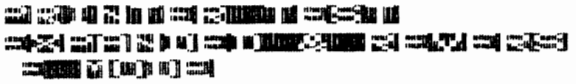
```
10 REM:PRINTER
20 OPEN1,4
25 CMD1
30 PRINT#1,CHR$(18)"PRINTER TEST"
32 PRINT#1,"EXPANDED MODE"
35 PRINT#1,CHR$(146)
37 CMD1
40 LIST
```

```
PRINTER TEST
EXPANDED MODE
```

Note here that only the first part of the text is printed in reverse mode. The printer reverts automatically to normal mode at the end of the line. To make both lines appear in reverse mode you must change line 32 to include CHR$(18) as in the previous line. CHR$(146) causes the printer to revert to normal mode too, but as you will have noticed the command is superfluous at this point. Its major use would be when there is a need to mix both reverse and normal text on the same line.

Now add the following program to your repertoire: The simplest way to do this is to type NEW and then bring the cursor up to each line of the program, pressing the return key for each line you wish to keep as it is. Alter the other lines according to the listing given here, and then run the program.

```
10 REM:PRINTER
20 OPEN1,4
25 CMD1
30 PRINT#1,CHR$(18)CHR$(14)
   "PRINTER TEST"
32 PRINT#1,CHR$(18)"EXPANDED &
   REVERSE MODE"
35 PRINT#1,CHR$(15)
37 CMD1:LIST
```



Reverse mode is a trifle difficult to read but is nonetheless eye-catching.

Now explore things a little further with the following program:

```
10 rem:Printer
20 oPen1,4,7
25 cmd1
30 Print#1,chr$(17)"Printer test"
32 Print#1,"cursor down mode
   (lower case"
37 cmd1:list

Printer test
cursor down mode (lower case
```

This time the whole thing is printed in lowercase. To revert to uppercase, or *cursor up* mode as it is called, you must give the command CHR$(145).

The printer can also allow for printing starting at a point other than the left-hand side of the sheet of paper:

```
10 rem:Printer
20 open1,4,7
25 cmd1
30 Print#1,chr$(16)"08Printer
   test"
32 Print#1,chr$(16)"23Print
   Position"
37 cmd1:list

Printer test
                Print Position
```

Note here that no command was given to the printer to revert to cursor up mode. The result is that the text and the program are both printed in lowercase letters. Note that the print position is changed by the command CHR$(16). The actual point at which printing is to start is indicated by the number, which must appear within the quotes.

Unless you provide instructions to the printer, it will continue to print material one line after the other. There are times when it is necessary or even just more pleasing to the eye to have a free line or a number of free blank lines between one portion of material and the next. The code for a linefeed is CHR$(10) preceded by a PRINT# statement.

The 1525 printer is what is known as a *graphic printer*. Quite clearly this has something to do with the ability to produce graphics of some sort. One of the most useful attributes of a *dot-matrix* printer is the ability to print *fonts* (types of print) other than those set automatically by the manufacturer. Some programming is required to do this, and a fair amount of preparation of the data the program will use is also needed.

Careful examination of any printout will disclose the fact that each character is made up of dots. The characters that are displayed on the screen are formed in the same way. Dots are arranged in patterns that outline the shapes of letters and numbers. It is possible to change these patterns so as to produce italic, Gothic, Russian, Hebrew—any character set in fact—by telling the printer to strike the ribbon in patterns we specify. The procedure is not very complicated, but does require some understanding of binary notation.

A piece of graph paper would be useful at this point, but failing that, draw a block seven squares wide and seven squares high. Place dots in the squares so that the new shape you want appears.

Let us say that you wish to produce one of those modified letter o's that is used in German. It is an ordinary lower case o with two little dots on top. The two dots are called an *umlaut*. It looks like this: ö.
First draw it on the graph paper.

```
 1  * *
 2
 4   ***
 8 *   *
16*     *
32 *   *
64  ***
```

Next number each row using binary code: 1,2, 4,8,16,32,64. Next, moving vertically down each column, add only those values where there are dots. In this example, there is only one dot in column one, and that corresponds to the binary number 16. Write that number under column one. The next column has two dots; at 8 and 32 add the two numbers and write the result under the second column. The next column, total 69 (1,4 and 64).

The whole row of columns yields the following totals:

16,40,69,68,69,40,16,0

Make sure you include the 0 for the last column. The printer must have that information for it indicates that no dots will be set for that column. Now it is necessary to add 128 to each of the numbers to produce 144,168,197,196,197,168,144,0.

This information is then placed in a data statement that is read by the program. This is what such a program looks like:

```
 5 REM:PRINTER DOT GRPAHICS
10 DATA144,168,197,196,197,168,
   144,0
20 OPEN1,4
30 PRINT#1,CHR$(8)
40 FORI=1TO8
50 READA
60 PRINT#1,CHR$(A);
70 NEXT
80 PRINT#1:PRINT#1
ö
ö
```

The umlaut ö is printed neatly, just once, on the page. An alternative method of dealing with the data line is to enter only the initial values derived before adding 128. Then, instead of requiring the computer to READ CHR$(A), you can ask it to READ CHR$(A+128), thus adding in the extra amount automatically.

Any special characters can be programmed in this way and called up at will. However, there has to be some means of telling the printer which character is wanted at what time. The following sequence, which is set with high numbers with the intention of using it as a subroutine, produces the umlaut ö each time the function key F1 is pressed.

```
1 REM:PRINTER GRAPHIC WITH
  FUNCTION KEY
63000 OPEN1,4
63001 GETA$:IFA$=""THEN63001
63002 IFA$="■"THEN63006
63003 PRINT#1,A$
63004 PRINTA$;
63005 GOTO63001
63006 DATA16,40,69,68,69,40,16,0
63007 PRINT#1,CHR$(8);
63008 FORI=1TO8
63009 READA
63010 PRINT#1,CHR$(A+128);
63011 NEXT:PRINT"*";
63012 PRINT#1,CHR$(15)
63013 RESTORE:GOTO63001
```

The first few lines open the file, check to see if a key is being pressed, and then check to see if F1 is being pressed. Line 63003 and 63004 print the result of GETA$ both to the screen and to the printer. If F1 is pressed then the program skips round the printing portion in order to read the data line. Line 63007 puts the printer in graphic mode, line 63008 begins a loop which reads each of the columns, 1 to 8. The screen shows the asterisk instead of the new character. Before returning to the earlier part of the program to start all over again, the data line is restored so that it can be reused.

There are eight function keys, so it is conceivable that eight different symbols could be programmed without resorting to complex programming techniques. All of the modified vowels for German, Swedish, French, and Spanish, as well as special consonant characters for those languages, are examples of the sort of thing dot graphics are useful for. An entire international phonetic character set could be produced as well as editors' marks or Greek.

Dot graphics can be used to produce bar charts in hard copy form. First let us look at a simple one and then work up a proper (albeit small) program that does something useful.

```
5 REM:BAR CHART PRINTER
10 A=128
20 OPEN1,4
30 INPUT"LENGTH";L
40 PRINT#1,CHR$(8);
60 PRINT#1,CHR$(26)CHR$(L)CHR$
   (255)
70 PRINT#1
```

Responding to the input from the question, the printer produces a neat row of blocks equivalent to the length required. All you need to do to produce a small bar-chart program is to place the printing portion in a subroutine and place an initial sequence at the beginning:

```
5 REM:BAR CHART PRINTER
10 INPUT"HOW MANY NUTS";L
20 GOSUB10000
30 INPUT"HOW MANY LEMONS";L
40 GOSUB10000
50 INPUT"HOW MANY ORANGES";L
60 GOSUB10000
9999 END
10000 OPEN1,4
10003 PRINT#1,CHR$(8);
10004 PRINT#1,CHR$(26)CHR$(L)CHR$
      (255)
```

```
10005 PRINT#1,L
10006 PRINT#1:CLOSE1,4
10007 RETURN
```

To make the program really useful, you should add a few lines that not only tell the user of the chart what each row refers to, whether nuts, oranges, or lemons, but also the value the line represents.

The various CHR$ codes that command the printer are as follows:

CHR$(8)	Graphic mode
CHR$(10)	Line feed after printing
CHR$(13)	Carriage return
CHR$(14)	Expanded or double-width mode
CHR$(15)	Standard character mode (after some other mode)
CHR$(16)	Position: point at which printing starts
CHR$(17)	Cursor down mode (lower case)
CHR$(18)	Reverse image mode
CHR$(26)	Graphics repeat code
CHR$(27)	Dot address code
CHR$(145)	Cursor up mode (upper case)
CHR$(146)	Revert from reverse to normal mode.

In addition to the CHR$(16) code, which indicates where printing is to begin on the sheet, you can use TAB and SPC. Sometimes there is a possibility of confusion over these two commands. TAB indicates an absolute position. TAB (10) will start the printing at point 10 (column 10) on the sheet. You could put a number of TAB(10) instructions in your program, and the material would be printed at column 10. If you wish to have one portion of the material begin at column 10 and another portion on the same line begin at column 60, you would need to use the CHR$(16) command in the form indicated in Fig. 9-6.

Try the following:

```
PRINT TAB(10);"#";TAB(10);"#"
PRINT SPC(10);"#";SPC(10);"#"
```

Each of these lines will put the # in different spots on the screen. The first will place them next to one another, and the second 10 columns apart.
Now try the following:

```
OPEN 1,4
PRINT#1,"";SPC(10);"#";SPC(10);"#"
```

Then replace the SPC with TAB and you will see that you achieve the same effect! The above is not designed to put you off using either TAB or SPC but to warn you that the results will not always be what you either expect or want! Use CHR$(16) instead!

As I remarked at the beginning of the chapter, a printer is an extraordinarily useful device. The machine is not limited to merely printing out program listings, as many of the examples in this chapter have attempted to point out.

Perhaps one of the most useful aspects of the combination of computer and printer is in the field of word processing. You are forgiven for thinking that the term word processing was invented after the introduction of the microcomputer. It didn't. The term is a horrid one, but serves to distinguish the activity from the process of mere typing or writing by hand.

Any manuscript requires that there be alterations from time to time, even if those alterations are confined to correcting spelling mistakes.

Sometimes vast changes are required: you might wish to add material in the middle of a chapter or even add a sentence in the middle of a paragraph. It is very irksome to have to retype a whole page for the sake of one sentence. Often this means the retyping of everything that follows (Ugh).

Word processing is the means whereby you can make all of these alterations quickly, with varying degrees of simplicity depending on the software you are using, without becoming too annoyed!

There are word processing programs available for the C-64 ranging from the fairly simple to the quite complex and complete. Material is stored in the computer, transferred to disk, and then printed out only when you are certain that the document comforms exactly to your requirements (or those of your academic advisor!). In the latter case, how-

ever, be warned that many institutions are fussy about the quality of print from a dot-matrix printer. They will not accept papers let alone theses in that form. Only fully formed characters produced by a daisy-wheel printer are considered acceptable.

All of which brings us back to the starting point of this chapter: whether or not to use the 1525 printer or buy an interface that will allow the use of almost any other printer you care for.

You could, of course, enjoy the luxury of a dot-matrix for drafts and a daisy-wheel for polished copy. There are now printers that will produce very near letter-quality characters from a dot-matrix format as well as produce draft copies in rapid order. The good news is that the price of printers is dropping daily. It is still possible to purchase reconditioned machines that have come from older word processing installations. Be sure that they will run when driven by your C-64. Get advice from someone who really knows what he or she is talking about in the field. The computer store salesman is not the person to ask for this kind of advice!

Chapter 10

I may be a little strange for it was a long time after I bought my first computer that I acquired any commercial software. I preferred to write my own. When I did buy some software, I was rather disappointed. Nonetheless, it occurs to me that it is very likely that with the purchase of a disk drive will go the purchase of some software; and I propose to deal with the simple process of loading programs from a disk. The reference manual that comes with the 1541 disk drive does explain everything, but it might just be a little confusing to the first-time user.

USING THE DISK DRIVE

Incidentally, you may have a friend who owns the VIC-1540 disk drive. This device is now no longer made and will only work (unmodified) on the VIC-20 computer. The VIC-1541 disk drive will work with both computers, but requires a small program to make it work with the VIC-20. Straight out of the box, the 1541 is ready to work with the C-64 without effort on your part.

The manual is quite clear on plugging the various bits and pieces in. If you have a printer, the cable from it will go in the socket to the left on the disk drive, assuming that you are facing the disk drive with cables at the back.

When all is set up, switch on the disk drive by means of the switch at the back, then switch on the printer if you have one, and finally switch on the C-64. The red light will come on in the front of the drive and then go out while the green light comes on. Only now may you insert a disk.

The following concerns only ready-to-use disks that have programs on them. If you are about to use a blank disk fresh from the box, wait a little while. I'll deal with that in a moment or two.

Hold the disk flat with the label on top. There will be a small square notch on the lefthand side. There will also be a long notch in the black cover through which the disk itself can be seen. This long notch will be facing away from you. Carefully insert the disk into the slot in the front of the drive and then close the slot by pressing down on the flat piece of metal sticking out in front. A click will tell

you that the disk is inserted properly. Now, before you go any further, just push on that same piece of flat metal. The disk should pop out at you. I suggest that you do this two or three times to make sure you have the feel of it.

When you are comfortable with this, leave the disk in place and then turn your attention to the keyboard.

Type LOAD (either in full or L (shift 0)"$",8.

The $ sign is the code for the directory or menu of programs on the disk. The 8 refers to the disk drive. If you omit the 8, the computer will think you are retrieving a program from tape.

In no time at all the computer will tell you that it is searching for $, has begun loading $, and has completed the process. If you have been accustomed to tape, the speed will both amaze and delight you!

Now type LIST (not RUN). The menu of programs will raster up the screen, the rate at which it does so being controlled by the CTRL key.

To select a program, either enter the word LOAD followed by the name of the program in quotes, followed by a comma, followed by 8; or bring the cursor up to the program line, move it across past the acronym PRG and then erase back as far as the second set of quotation marks. Then type a comma and 8. Now move the cursor back to the beginning of the line, where you will find a number which tells how many blocks the program takes up, and type LOAD. Then press the return key.

The screen will show what is happening. Again, in a very short space of time the computer is ready. Now type RUN and press the return key to use the program. If you have problems, go over the entire procedure once more, double checking every step.

You may have been tempted to buy some of the so-called educational software available from your dealer. This software has been placed in the public domain by Commodore. All you actually pay for is the disk upon which the software is copied. Computer assisted instruction software is my specialty, and I regret to have to say that much of what is available is of very poor quality. Programs that make use of graphics will not run on the C-64 without a lot of modification. After you have read this book, absorbed it, and can do everything in it blindfolded, you will have the skill to make these changes—if you consider the effort worthwhile, that is.

Meanwhile, concern yourself with the question of dealing with a new disk. With the drive there came a disk. This was perhaps the one which you just loaded.

The first program on this is called How to Use. Be warned that you should read what I have to say on the matter first and then go through the program. (Actually, that is a little pretentious of me, the 1541 manual is very well written!)

Type in the following: LOAD"$",8 as directed above. When the READY appears type LIST; then, as described above; select the program Performance Test.

When it is loaded enter RUN, and then follow the instructions. A scratch disk is an unused disk or one that you have no further use for (unlikely for a first time user, I know!).

The program will check the disk drive for you and give you a blow by blow account of its behavior. You run this program from time to time, but more particularly if you suspect damage of some kind.

There are a number of other programs on that test disk that you will find more useful when you have played with the disk unit a little more. Let us say that you have one or two favorite programs on tape that you would like to put on disk. First load the program from tape in the normal way; the fact that you have a disk drive attached to the C-64 does not alter what you do with the machine otherwise.

The disk drive is recognized by the computer as device 8. Thus, to save your program to disk tape SAVE"program name",8. Do not forget the comma! The disk will begin whirring, and the screen will tell you in no time flat that the program is saved!

Now place your entire collection of programs on disk, one at a time, in the same way. Always give each program a distinctive name. To retrieve your program you merely load the program as above.

One major advantage of disk over tape is that a list of programs is automatically placed on the disk. As mentioned before, LOAD"$",8 followed by LIST when the word READY appears will give you the entire listing by name and the number of blocks left free.

After you have played with the new disk, getting comfortable with loading and saving, take that disk out and reinsert the test disk. Type LOAD "$",8 and then LIST. In the directory you will find a program called C-64 Wedge. Load it and then run it. Take out the disk and reinsert your own. Now, instead of typing LOAD"program name",8, just type /"program name" with no comma and no 8. Up comes your program! Should you wish to see the directory, merely type @$ or the greater-than sign followed by $. When the little red light blinks gently on and off to indicate an error, pressing the @ or $ alone will give you the error message number, text, track, and block number.

NEW is mandatory when using a disk for the first time, or so the manual says. It makes sure the disk has nothing on it, and then places block and timing markers on it. The whole command is given as follows:

```
PRINT#15,"NEW8:name, ID".
```

The name of the disk is any you care to give it, and should appear on the sticky tab which goes on the outside of the disk. The ID is any two digits. They should also go on the tab.

When you update a program and alter it you have a choice of saving it under a new name or replacing the old program. The first method is obvious; the second method is accomplished by SAVE"@0:program name",8 in which case the entire program is saved in its new form in the old place.

If you have a printer, the directory can be listed with the usual printer commands (see Chapter 9). It is useful to keep printouts of lengthy directories. Paper has not yet been made obsolete!

From time to time you will find it necessary to erase a program or data file from the disk. Maybe you have no further use for the program or you have transferred the program to another disk for better organization of your programs and find the initial placement superfluous. Whatever the reason, you now would like to use the space for something else. There is a command which allows you to do this:SCRATCH. The format for this command is:

```
PRINT#15, "SCRATCH0: program name"
```

which can be abbreviated to

```
PRINT#15, "S0: program name"
```

Unlike a tape, in which all programs are accessed sequently, a disk drive is able to access the correct program by moving the head across the spinning disk to the required track. This is rather like the way a particular number on a phonograph record can be accessed by moving the stylus to the required cut. There is no need to play all the tracks in order to reach the one you want.

The disk spins around approximately 5 times a second. There are 35 tracks on the disk; these are composed of 683 blocks of which 664 are free for your use. The total number of names that can be stored in the directory is 144; thus you may place no more than 144 programs on one disk. Each program would have to be fairly short (less than one kilobyte), but you can see that there is plenty of space on a disk.

Having loaded the disk directory a few times you might begin to wonder if there is not a faster way to find those programs you know exist on the disk without rastering through the entire contents. Well, yes there is.

Let us say that you have the disk that contains the programs used in this book. You know that there are some sequences that deal with sprites and would like to look at just those. These programs all begin with the word *sprite* followed by a number. If you ask the computer to LOAD"SPRITE",8, you will probably get an error message. Instead, enter the following: LOAD"SPRITE?",8. The question mark is a *wild card* that stands for any number or letter in that position in the program name.

The program that loads into the C-64 is the

first program called Sprite that exists on the disk. In order to see the whole list of programs called Sprite, you must change the command slightly to the following: LOAD"$0:SPRITE?",8.

Now type LIST, and you will see that all those sprite programs that are modified by a single digit are listed in directory form. In order to see those programs that are modified by more than one digit, the requisite number of question marks must be entered in the load command; thus, SPRITE14 would be found by entering "SPRITE??".

I cannot labor the point that in writing your own programs, you should make use of frequent saves as you progress. You will have found that the process is a little tedious with the tape recorder. The disk allows for such fast saving and loading that it is now no longer a chore to save a partially finished program. There is another advantage to saving the disk. Whereas with tape you must either overwrite previous work or save the new program after previous versions, with the risk that the computer will look for and load a version that is more primitive than your last version; with the disk you have the choice of keeping different versions or retaining only your latest one.

There is a command that allows you to modify a version of a program without taking up too much space. It is: SAVE"@0: program name",8. This will update your program. You could, of course, call the program by a different name, perhaps X2 or X3, each modifier indicating a new version. There is a chance of losing track of what each version does of course, but practice and familiarity with your own operating procedures will help to keep matters straight.

VERIFY does exactly what it does on tape. The format is the same as for the other commands you have dealt with: VERIFY"program name",8.

My companion book on the VIC-20 contains a file program that allows for the easy storage and subsequent access of whatever information one wishes. However, it is a rather extravagant use of computer memory for all of the information that is stored on tape or disk winds up using RAM.

To explain things more clearly: the program itself takes up a given amount of memory. Running the program causes the memory to fill up with the names, addresses, recipes, record collection, or what have you that you wish to store. You save the resulting program including the data. When you reload that program into the computer it takes up all the memory it needs, often leaving you with little RAM to work with further.

Fortunately it is possible and sensible to make use of all that space on the tape or disk without consuming all that precious memory in your computer, even though you have 64K!

What happens is simply that you make use of the tape or disk as a storage device while the program is running. Each item of information you wish to store is transferred to the storage medium at once; the computer has no need to store it in its memory because the tape or disk holds it.

Accessing the information uses a reverse process. One simply calls the information from the storage medium as needed, dumps it temporarily into the computer RAM, and then discards it when it is no longer needed. One might use the analogy of the schoolbag and bookshelf: the shelf holds all of your books. The bag holds only those needed for that day's classes. The shelf is the tape or disk, and the bag is the computer.

There is an important difference between tape and disk. Tape storage is linear or sequential only. In order to find item x, you must traverse through the entire contents of the tape.

A disk file can be organized sequentially but this is a rather wasteful method, for the disk must still be traversed through each item in order to find the one required. A much better method is to use random-access files. In this type, the item needed is selected directly without having to go through the entire contents. You might well argue that it doesn't really matter, for the disk is so fast. Believe me, there will come a time when you will wish that the disk was a whole lot faster. It is very strange but true that the user soon begins to catch up on the speed of the computer and wishes for something ever faster!

The TEST/DEMO disk that comes with your 1541 disk drive contains both sequential and random-access file programs that are quite easy to

use. Make sure that you first load the programs into your C-64, then replace the DEMO disk with one that you have already formatted. Copy the program onto that disk (just so that you have a back-up copy) and proceed to use the program. Do not try to use the DEMO disk or else you will get an error message. The message tells you that there is a *write-protect* device in place on the disk. Take the disk out of the drive, and find the piece of silvery adhesive tape that sits on the left side of the disk. This tape covers the slot on the left. The tape causes the disk drive to assume that there is no notch and that nothing can be recorded on the disk. If you remove the tape, you will be able to add *utility* programs (programs which speed your general computing work as opposed to games and the like) of your own. The write-protect tape serves a very useful purpose, of course, and should not be removed without first thinking carefully about what you are going to do. Always replace it. It is a safeguard against program loss!

Properly speaking, anything you have on disk should be called a *file*, although some of the files are programs. Files can be *concatenated*. You have met that term before in reference to using strings. (The program in which the computer generates random sentences uses concatenated string arrays.)

A simple example of concatenation is the command to join two programs together to form a new program. The command for this is:

```
PRINT#1, "C:new name=old name 1, old name 2"
```

The letter C actually stands for copy. You are simply copying two old programs onto the same disk. If you wished to make a copy of one program somewhere else on the same disk (for back-up purposes, say) a similar form of the copy command is used:

```
PRINT#1, "C:SPRITE30=SPRITE1"
```

This would mean that the program Sprite1 is now stored as Sprite30 in addition to the original program.

Each device that the computer addresses or talks to has a number. The computer is essentially a dumb creature and therefore must know exactly which portion of itself it is dealing with at any time. Parts such as the Datassette, the keyboard, or the screen are dealt with automatically, it seems. When you want to deal with another device such as the disk drive, you must give the device a number so that the computer will not send the information to the wrong portion of itself. The device number of the disk drive is 8. That is why that number appears at the end of a load or save command when using the disk.

What if you are using two disks? Somehow or other you must indicate which of the two disks is being accessed or else the whole outfit will *hang-up*.

If you have bought a second disk-drive you should read the instructions carefully to find out how to change the device number from 8 to 9. There is a hardware modification that is not all that difficult to carry out. If you feel a trifle tremulous at doing this yourself, ask that it be done at the store when you are buying the drive. There is also a software modification. The disadvantage of this is that you must run the short program each time you wish to use both drives.

The Commodore Executive machine purchased with two drives will already have the devices labeled correctly.

When using two drives be sure that you have provided individual and distinctive identity numbers to each of the disks, so that the computer knows which one is in each drive. Quite obviously you must also keep track of what you want to send and where you want to send it, and indicate the proper drive by means of either 8 or 9 at the end of each command.

Chapter 11

It is time to look at some further aspects of the language called BASIC. It is all very well to be able to do clever things with graphics and sound, but if you do not have a full understanding of the tricks that are available, you are likely to produce pretty pictures and pleasing sounds that actually do nothing.

SPECIAL TRICKS IN BASIC

You have already glanced at the CHR$(x) keyword. You will recall that if we present a line, such as PRINT CHR$(88) the screen will show an upper case X. Remember that every single item on the keyboard, including such things as cursor controls and the return key, has its own code number. A list of them is included in the *User's Guide* that came with your computer. The code number for the return key is 13. Thus, to check whether or not the return key has been pressed, you would first assign the code to a variable such as X$ in the form X$=CHR$(13), and then allow the program to do the check using an ordinary if-then statement.

```
10 X$=CHR$(13)
20 INPUTA$
30 IF A$=X$ THEN PRINT "YOU PRESSED
   THE RETURN KEY, DIDN'T YOU?"
```

The above example is not very useful beyond trying to fool folk who are unfamiliar with computers into thinking they have psychic powers! A more useful ploy would be in a situation wherein a simple pressing of the return key causes the program to continue, without the user having to respond to a question. The space bar could be used in the same fashion and is used this way in much of the educational software I have seen from Commodore.

The function keys, about which I have so far said nothing, also have their codes. The codes run from 133 to 140, eight in all. The first four numbers, 133 through 136, refer to the odd numbered keys. Thus 133 refers to F1 and 135 to F5. The odd numbered keys are activated by simple pressing. The even numbered keys are accessed by first holding down the shift key. F1 thus becomes F2. The even numbered keys are coded 137 to 140.

```
10 X$(1)=CHR$(133)    F1 is assigned to variable X$(1)
11 X$(2)=CHR$(137)    F2 is assigned to variable X$ (2)
```

etc.

You can use each function key as a means of producing some action which is unique. It might be that you have a complex program, such as a word-processor, which allows you to move to a different operation, say editing, by the mere pressing of one function key. Let us say that you isolate the printing process in this manner. CHR$(137), which is F2, could be the trigger for printing. The fact that you have to press the shift key in order to print is a safeguard. You are not likely to print out your material accidentally!

Other keys can be used to trigger particular activities. It is usual to confine yourself to keys that are otherwise unused. For example, it would be quite frustrating to assign the letter B to some situation or other, only to find that you cannot use it in normal writing! (Actually that is unlikely for reasons you will already understand—but the point is still valid.) Keys that may be available for assignment to special functions include the special signs such as + and −, the vertical and left arrows, the asterisk, the ampersand (&), and the @ sign. These keys are used extensively in the word-processing program which you find at the end of this book, and which is included in the disk and tape produced by TAB BOOKS Inc, containing all the programs in this book.

A computer is marvelous at being able to repeat an action for as many times as you wish, forever if need be! A simple example is the already familiar for-next loop. Not quite so obvious is the DEF FN keyword. This allows you to define a formula as a variable and then merely call that variable whenever you need to use that formula. For example:

```
10 DEF FN X(1)= Y+19
20 INPUT Y
30 PRINT FN X(1)
```

Whatever value you assign to Y will then be used in the formula assigned to FN X(1). FN X(2) could be a quite different formula. This keyword proves remarkably useful in a program such as one designed to calculate tax. In order to claim all that you are entitled to, it is often necessary to juggle figures around. By using DEF FN you can store the formulae and enter all sorts of values, the result changing instantly with enabling you to make equally instant comparisons. Now isn't that useful?!!!

The get keyword can often cause some confusion. Why bother with GET when there is INPUT? The input command reads what the user types in its entirety. The get keyword reads the keyboard one character at a time. So what? GET is useful in a situation such as the one mentioned previously wherein the function keys are assigned special duties. The statement is placed in a loop as follows:

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

The line recycles as long as no key is pressed. By means of a series of lines that check for specific keys, such as function keys, the program is directed to perform some subsequent action. Incidentally, you will find the keyword INKEY$ in some programs written for machines other than Commodore. It performs the same function and is therefore readily translatable into GET.

I have dealt with GOSUB by using it in programs. However, you might well be left with the impression that you can only GOSUB from a low line number to a high number. This is not so, for you can GOSUB just as easily from a high line number to a low one. A situation might arise in which the program has progressed to, say, line number 5540. You now need material that is contained in line 2370 to 2530. GOSUB 2370 will place you there and the return statement will put you back at 5540 when you have finished that section. The same principle applies, of course, to GOTO, but that might be more obvious. SUB implies under, or in a programming sense, beyond, but under can also be over—if you see what I mean!

It is often necessary to siphon off just a portion of a piece of text for special treatment. The means to do this are three-fold.

Let us say that you wish to glean just the first name of an individual when the whole name is given. Obviously the normal manner of presenting names is to place the first name first; thus *John Herriott*. It would be a nuisance to see the full name appear each time, likewise if you wished to use only the last name. Somehow or other you need to be able to suppress the unwanted portion. The means of doing this fall into the category known as *string functions*. They are: LEFT$, MID$ and RIGHT$. Be sure that you do not omit the $ at the end, or else a syntax error will be reported.

Using the example above, you would deal with the first four characters, first assigning these characters to a variable; thus:

```
10 A$="JOHN HERRIOTT"
20 B$=LEFT$(A$,4): PRINT B$
```

The result of all this magic is that the computer will print only JOHN and ignore the rest. To print only the last name, you would command the computer to ignore the first five characters, or rather only use the last eight, as follows:

```
20 B$=RIGHT$(A$,8):PRINT B$
```

The result is HERRIOTT.

What if you wished to cull a series of characters from the middle? MID$ is the string function used here, but it is rather more complex. You must indicate not only the precise length of the slice needed but also the starting point from which the slice runs. Thus the MID$ function must contain two sets of numbers; the first, the starting-point character and the second, the number of characters to be included. The form is thus MID$(B$,5,3), which in the example above would produce the result HER! To complete the joke we could set the value of MID$(B$,8,4) which would produce RIOT!

There is, of course, a difficulty with the first example. You don't know how long a person's first name is going to be. Arbitrary assignment of a value for LEFT$ might produce a situation in which an individual with a name like HECTOR would be called HECT throughout the program. That would be improper. You could get around the problem, as opposed to solving it altogether, by using another function known as an *integer function*.

LEN(A$) is the function in question. It tells you how many characters there are in a given string. Spaces between characters are included of course, for a name such as WILLIAMAANDERSON is normally written WILLIAM A. ANDERSON, and thus contains a total of 19 characters.

The trick is assigning LEN to a string in order to cull just the first name would consist of checking to see how many characters have been entered by the user, and if the total is more than a certain value, a prompt would appear saying something like:

```
JUST YOUR FIRST NAME PLEASE.
```

```
10 INPUT"PLEASE TYPE YOUR NAME";A$
20 IF LEN(A$) 12 THEN PRINT" JUST YOUR
   FIRST NAME PLEASE":GOTO 10
```

Of course, you have to assume that there would be no name longer than 12 characters. Montmorency will fit, but a double name such as Mordecai St. John will not! If the bearer of that fine handle is accustomed to being known by it in its entirety, you must suffer the loss of friendship, or blame the computer for being dumb! Care must be exercised for a name such as John Smith will fit into our limits quite nicely.

The ultimate (or almost ultimate!) solution would be to check for the first space and use only that which comes before it. For the last name to be used, you would check for the second space (if you're using initials), except for the case of pianist Richard Herriott whose initials are R. M. J. Herriot. He would be known as J. Herriott, which is my wife!

You might well, by this time, be thinking that computers create more problems that they solve. I have been presenting the most awkward situations in order to encourage thought. It is very nice to be able to do tricks, but when it comes down to the fine print, a decision has to be made as to whether a feature is worth the effort. Only you can decide.

Although the commands list, run and new have been dealt with in various examples, I wish to introduce a note of caution. It can happen (as it did to me . . . with most frustrating results!) that one of these words creeps into your program without your noticing it. I was working on a program and entering commands very rapidly. Suddenly my program would not run, but only list! In vain I sought the reason and had begun to blame the computer, the supplier of electricity, my wife for running the dishwasher, the kids for being kids (they had been no where near the machine, mark you!), and the world in general and had begun to contemplate the life of a hermit as being the only worthwhile profession.

I sent the listing to the printer and found the problem immediately. Line 23 contained the command LIST. Obviously I had somehow pressed these two numbers, quite inadvertently before typing the command list and had not noticed. Thus, the program carried out the instruction as soon as it reached line 23. Can you imagine what would have happened if the command NEW had been entered?

It is likewise with the RUN command. This command has the habit of clearing out any variables, and should the command find its way into your program by accident, you will find yourself with a program that aborts due to the fact that the variables have nothing assigned to them!

POS is a curious command. It assesses the current cursor position on the screen. Should you have a situation wherein you wish to limit the number of characters input by the user, you could limit the line length using POS in the form:

```
200 IFPOS(0) 34 THEN PRINT CHR$(13)
```

This would mean that as soon as position 34, which is, of course, less than the 40 character width of the Commodore screen output, is reached by the user, a carriage return would be enacted, bringing the remainder of the input to the next line. The zero in parentheses is a dummy argument. The important numbers are those which follow the function. The range of numbers that can be used runs from 0 to 79 (80 characters in all, that being the extent of the logical line on the Commodore).

The remainder of the words in Commodore BASIC mainly concern mathematical matters and will be obvious to those who wish to use them. There are three, AND, OR and NOT, which are known as logical operators. They test for truth or its lack. They can be used in mathematical program or in textual ones, for the simple reason that the computer is actually only dealing with numbers when you get right down to it. An example might be as follows:

```
10 INPUT A$, B$
20 IF A$="FEVER"AND B$="COUGH" THEN
200
30 IF A$="COUGH" AND B$="SPOTS" THEN
300
```

Quite clearly there is (so far at least) no result if A$= FEVER and B$ = SPOTS. You can go on with a series of IF-THENs which test for all manner of combinations. If only one of two conditions needs to be true, OR can be used.

For mathematical computation, there is a hierarchy of operations. This was laid down many years ago. Simply, the computer does certain operations before it does others. The order is as follows:

Exponentiation (the UP arrow to the right of the keyboard)
Negation (minus sign)
Multiplication/Division
Addition/subtraction
Relational operations (greater than/less than, greater or equal to and the converse)
NOT
AND
OR

It is useful at times to give the same variable label with the appropriate modifier to two types of input that concern the same item. Let us say you wish to produce an income and expense account program. Quite clearly you wish to match category with amount. Categories are text entries such as *automobile*, and the amount is a numerical value

```
1 REM  EXPENSE ACCOUNT
5 PRINTCHR$(147)
10 INPUT"ENTER MONTHLY INCOME";S
15 PRINT
20 INPUT"EXPENSE CATEGORY 1";O1$
25 INPUT"AMOUNT";O1
30 PRINT:INPUT"EXPENSE CATEGORY 2";O2$
35 INPUT"AMOUNT";O2
40 PRINT:INPUT"EXPENSE CATEGORY 3";O3$
45 INPUT"AMOUNT";O3
50 PRINTCHR$(147)
55 T=O1+O2+O3
60 B=T/S
80 PRINT:PRINT
100 PRINT"MONTHLY INCOME"SPC(2);"TOTAL EXPENSES"SPC(3);
    "BALANCE"
110 PRINT"$";S;SPC(12);"$";T;SPC(6);"$";S-T
```

Fig. 11-1. The Expense Account program.

such as 100. The first item could be labeled C1$, and the second C1.

It may be that you wish to round down the amount you enter as having been placed in a bank account. This is good practice, for the cents mount up over quite a short time! Say you actually put in $503.92. You can cause the computer to deal only with the dollars by using a line such as: INPUT D% (% indicates an integer). You can enter $503.92 if you wish, but the computer will ignore the cents. The surprise is that you will end up with more money than your C-64 says you have. It is always nice to prove a computer wrong!

Just for fun I have appended a short program that begins to deal with household expenses. It is shown in Fig. 11-1. It is only the beginning of a good program. Perhaps you might like to expand it and personalize it. Try expressing the outcome of your calculations as a percentage of income. Provide yourself with a balance sheet which can be sent to the printer. Produce a bar chart. Color the result. Use sprites to compare sizes and percentages. Play music to yourself to cheer yourself at the results. Get the C-64 to play a Bronx cheer at the tax man!

Chapter 12

Close to the beginning of this book, you encountered the concept of the menu. A menu is a list of items from which you make a selection. Commonly, menus appear at the beginning of a program (at least it appears that way to the user) and allow the user to select a particular function. The first function is usually that of initialization. The remainder fall in some sort of heirarchical structure which is usually self-evident.

This chapter consists mainly of two such programs: a data base (note the lower case; I feel that uppercase would be pretentious for such a small program) and the second is a mini-word processor. Both are contained on the disk or tape that is available as an accompaniment and complement to this book.

A DATA BASE PROGRAM

First consider the data base, which is shown in Fig. 12-1. This program allows for the input of a variety of items. The program is self-contained but forms, when you have entered and sorted the items, a separate collection of data which is called a file. This file is then given a name and stored. In its current form, it can only be stored on tape. The information can be presented by the printer, however. Perhaps you would like to use the methods used in the word processing program to store the file on disk.

There is a limit to the amount of information you can store. This program was originally written for a ZX81 with only 16K. Hence the little calculations at the beginning of the program. They can be altered or removed. Depending upon how you store the items on disk or tape there need virtually be no limit. (There is a limit, of course, but it would exceed the usefulness of the program: better to store less and access it more quickly.

Items can be called up by name, altered, deleted or what-have-you, according to need. You can find out how many slots are left available for further items; you having decided initially how many items you wish to store in toto.

The file you create will be saved under that filename on disk or tape. Upon starting up, make

Fig. 12-1. The 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 OR DISK"
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
1002 PRINTTAB(2)"THIS PROCEDURE WILL DESTROY ALL PREVIOUS RECORDS"
1003 PRINT:PRINT:PRINTTAB(4);"PRESS 'C' TO";
1004 PRINT:PRINTTAB(4);"CONTINUE OR PRESS '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+2)
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 'D' IF USING DISK       'T' IF USING TAPE"
2555 INPUTC$
2560 IFC$="D"THENSAVE"N$",8
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=1TO500:NEXTI
6030 GOTO10
```

sure you do not attempt to reinitialize the program, or else it will seem that you will have lost all your data. There is a warning to this effect at the beginning of the program.

A WORD PROCESSOR PROGRAM

The next program is a mini word processor. People now no longer write, they process words. There is truth in this too, for you are now able to change whole blocks of text at the touch of a few keys and change words, alter spellings, insert odd sentences, and create new paragraphing without having to resort to a great deal of retyping. The word processor has been around for quite a long time. Now it is in the domain of anyone who has a microcomputer and a printer.

This program is written for the C-64. You may wish to modify the program: you must decide how you wish to do it. The variable labels are all fairly logical. The C-64 recognizes only the first two letters of a variable and so that is all I have included to conserve memory. LL is line length; RV$ is reverse on, and OV$ is reverse off (from observe, do you think?); W refers, wherever it appears either in conjunction with another letter or on its own, to words. Logical, don't you think? Some variables have been assigned rather arbitrarily.

The best thing to do with the program is run it, see how it works, and then change it to your needs. Save the new version on a utility disk. If you have the two-disk Executive machine from Commodore, one drive can contain the program disk and the other the file disk. In this case you must make sure that the file goes to the right disk. This will probably mean changing some address or device codes in the program.

You are allowed to change certain of the parameters of the text output, even though there are default values set at the beginning of the program. Thus you can change top and bottom margins (TM and BM), left margin, and line length. Set the top of the perforation just above the ribbon level on the printer, and the program will prevent you from printing on the perforation. Make sure you have ample paper before printing, for the VIC1525 has no out-of-paper signal; rather, it will continue merrily printing all your fine text on the platten—and scrolls have been out of fashion for quite a few years!

One small note which has to do rather with reserved words than with word processing. The matter arose during the writing of this program however, so now is as suitable a moment to comment on it as any. I had been using the variable LE to denote line length. At one point in the program I had a statement which read, in part, IFLE=LETHEN... or some such. The computer told me that there was an error! It was perfectly sound, however. Then I realized that the computer was not reading it in the same way that I was. It was trying to read it as LET HEN! this, of course, made no sense to it and so it reported an error.

A reserved word is any word that belongs to the vocabulary of the BASIC language. These words may not be used, except within quotes, in any ambiguous situation, nor may they even appear to have been used, as in the above example. Be warned!

Figure 12-2 contains the mini word processor. Alter it as you wish.

Fig. 12-2. The Word Processor program.

```
2 POKE53281,12:POKE53280,2:POKE646,15
4 LM=5:LL=70:TM=4:BM=4:RC=1:RE=500
5 RV$=CHR$(18):OV$=CHR$(146):CL$=CHR$(157)
6 CR$=CHR$(29):CD$=CHR$(17):CU$=CHR$(145):C$=CHR$(19)
10 DIMTB$(500):X=65536
11 PRINTCHR$(147)
12 PRINT
13 POKE646,2
15 PRINTTAB(4)"THE FOUNDATIONS OF A MINI C-64 WORD PROCESSOR "
```

```
16 PRINT:PRINTTAB(8)"BY JOHN HERRIOTT":POKE646,0
17 PRINT"■    KEEP THE FOLLOWING IN MIND AS YOU■■■ USE THE
   PROCESSOR."
18 GOSUB2000
19 PRINT"■   PRESS RETURN TO CONTINUE":INPUTZ9:GOTO400
20 REM:SUBROUTINE TO CHECK FOR AVAILABLE MEMORY
22 ME=FRE(0):IFME<0THENME=ME+X
24 PRINTTAB(5)"TOTAL BYTES LEFT:  ";ME:RETURN
100 PRINTCHR$(147):PRINTRV$"REFERENCE#";P:FORZ=1TO6:PRINT:NEXT
102 PRINT"  CLR/HOME:BRINGS CURSOR TO END OF LINE"
103 PRINT" SHIFT+CLR/HOME:BRINGS CURSOR TO BEGINNING OF LINE"
104 PRINT:PRINT" CURSOR CONTROLS BEHAVE AS NORMAL"
105 PRINT:PRINT" THE INST/DEL KEY BEHAVES AS USUAL IN BOTH
    SHIFTED & UNSHIFTED"
106 PRINT" MODE"
107 PRINT:PRINT" THE FUNCTION KEY F1 WILL RETURN YOU TO THE
    REVIEW EDIT MENU"
108 FORD=1TO4000:NEXT
110 IFTB$(P)=""THENTB$(P)=" "
120 PRINTC$CD$CD$;TB$(P)
121 L=1:PRINTC$CD$CD$;MID$(TB$(P),L,1);CL$;
122 GETWE$:IFWE$=""THEN122
123 E=ASC(WE$):IFE=133THEN166
124 IFE=29ANDL+1>240THEN122
125 IFE>31ANDE<96THEN154
126 IFE=29THENPRINT;MID$(TB$(P),L,1);:GOTO162
127 IFE=157ANDL-1<1THEN122
128 IFE=157THENPRINT;MID$(TB$(P),L,1);:L=L-1:PRINTCL$CL$;
    :GOTO164
129 IFE=145ORE=17THEN122
130 IFE=148ANDL+1>240THEN122
131 IFE=147THENPRINT;MID$(TB$(P),L,1);:GOTO121
132 IFE=19THENPRINTC$CD$CD$;TB$(P);CL$;:L=LEN(TB$(P)):GOTO164
133 IFE<>148THEN146
140 PRINTRV$;MID$(TB$(P),L,1);CL$;CHR$(148);" ";CL$OV$;" "CL$;
142 TW$=TB$(P):TW$=RIGHT$(TW$,LEN(TW$)-L+1):TB$(P)=LEFT$
    (TB$(P),L-1):WE$=" "
144 TB$(P)=TB$(P)+WE$+TW$:GOTO122
146 IFE<>20THEN154
148 L=L-1:IFL<1THEN121
150 TW$=TB$(P):TB$(P)=LEFT$(TB$(P),L-1):TW$=RIGHT$(TW$,
    LEN(TW$)-L)
152 TB$(P)=TB$(P)+TW$:PRINTCHR$(20);:GOTO122
154 IFL+1>240THEN122
156 TW$=TB$(P):TB$(P)=LEFT$(TB$(P),L-1):TW$=RIGHT$
    (TW$,LEN(TW$)-L)
158 TB$(P)=TB$(P)+WE$+TW$
160 PRINTMID$(TB$(P),L,1);
```

```
162 L=L+1:IFMID$(TB$(P),L,1)=""THENTB$(P)=TB$(P)+" "
164 PRINTRV$;MID$(TB$(P),L,1);CL$;:GOTO122
166 PRINTOV$;MID$(TB$(P),L,1):RETURN
200 TB$(P)=""
202 FORN=1TO240
204 GETW$:IFW$=""THEN204
206 B1=ASC(W$)
208 IFB1=20THENGOSUB1500:PRINTCHR$(157);CHR$(32);CHR$(157);
    :GOTO220
210 IFB1=133THENN1=240:GOTO400
212 IFB1=94THENPRINT"     ";
214 IFB1=43THENPRINT
216 PRINTW$;
218 TB$(P)=TB$(P)+W$
220 NEXT
222 RETURN
300 IFTB$(P)=""THENRETURN
302 IFASC(LEFT$(TB$(P),1))=94THENPRINT"     ";TB$(P);:RETURN
304 IFASC(LEFT$(TB$(P),1))=43THENPRINT:PRINTTB$(P);:RETURN
306 PRINTTB$(P);:RETURN
400 PRINTCHR$(147);CHR$(142):PRINT
420 PRINTTAB(15)"FACILITIES"
422 PRINT
430 PRINTTAB(5)"═══════════════════════════════════"
435 PRINT
440 PRINTTAB(10)"1. ENTER TEXT"
442 PRINT
445 PRINTTAB(10)"2. REVIEW/EDIT TEXT"
447 PRINT
455 PRINTTAB(10)"3. READ FROM DISK"
457 PRINT
460 PRINTTAB(10)"5. WRITE TO DISK"
462 PRINT
465 PRINTTAB(10)"5. WRITE TO PRINTER"
467 PRINT
470 PRINTTAB(10)"6. LEAVE SYSTEM"
475 PRINT:GOSUB20
477 FORD=1TO2000:NEXT
480 PRINT:PRINTTAB(8)"ENTER YOUR SELECTION BY NUMBER"
490 GETA0$:IFA0$=""THEN490
500 IFA0$="1"THEN600
505 IFA0$="2"THEN700
510 IFA0$="3"THEN1130
520 IFA0$="4"THENGOSUB1200
525 IFA0$="5"THENGOSUB1000
530 IFA0$="6"THENCLOSE200:END
540 GOTO400
600 FORG=1TO500
605 IFTB$(G)=""THENN2=G:G=500
```

```
610 NEXT
615 PRINTCHR$(147)
620 PRINT"     ENTER THE REFERENCE NUMBER OR PRESS 'RETURN'
    FOR NEXT AVAILABLE"
622 PRINT" REFERENCE NUMBER":INPUTN2
625 IFN2<00RN2>500THENPRINT"THE LIMITS ARE 1 TO 500":GOTO620
628 PRINTCHR$(147):PRINT"PRESS THE -1 FUNCTION KEY WHEN YOU ARE
    READY TO ";
629 PRINT"RETURN TO THE MAIN MENU."
630 PRINT;"━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━";CHR$(14);
635 FORP=N2TO500
640 PRINTCHR$(18);P;CHR$(146)
645 IFTB$(P)<>""THENPRINTCHR$(18);"REF BLOCK ACTIVE":
    FORZ=1TO100:NEXT:GOTO660
650 GOSUB200
655 IF(TB$(P))=""THENP=500
660 NEXT
665 GOTO400
700 PRINTCHR$(147):INPUT"ENTER REFERENCE NUMBER FOR REVIEW ";P
705 IFP<1ORP>500THENPRINTRV$"LIMITS ARE 1 500":FORD=1TO2000:
    NEXT:PRINTOV$:GOTO70
710 PRINTCHR$(147):PRINT:PRINT"━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━"
715 PRINTCHR$(14);RV$;"REFERENCE NUMBER ";P;OV$
720 IFTB$(P)<>""THEN800
730 PRINTCHR$(18);"  UNOCCUPIED "
740 GOTO805
800 GOSUB300
805 PRINTC$:FORY=1TO6:PRINTCD$;:NEXT
810 PRINT"━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━"
815 PRINTRV$"CODED KEYS:"OV$
818 PRINT
820 PRINTTAB(5)"B   BACKWARDS"
822 PRINT
825 PRINTTAB(5)"F   FORWARDS"
827 PRINT
830 PRINTTAB(5)"E   EDIT"
832 PRINT
835 PRINTTAB(5)"D   DELETE"
837 PRINT
840 PRINTTAB(5)"X   EXIT "
845 PRINTRV$"CONTROL CODES:";OV$:PRINT
850 PRINTTAB(10)"@  DOUBTLE WIDTH"
855 PRINTTAB(10)"↑  NEW PARAGRPAH"
860 PRINTTAB(10)"+  BLANK LINE"
865 PRINTTAB(10)"←  LEFT MARGIN"
870 PRINTRV$"INDICATE CHOICE ";OV$
875 GETA0$:IFA0$=""THEN875
880 PRINTCHR$(147)
885 IFA0$="X"THEN400
```

```
887 IFAO$="F"ANDP<500THENP=P+1:GOTO700
890 IFAO$="F"THEN700
892 IFAO$="B"ANDP>1THENP=P-1:GOTO700
895 IFAO$="B"THEN700
896 IFAO$="E"THENGOSUB100:PRINTCHR$(147):GOTO810
897 IFAO$="D"THENTB$(P)="":GOTO700
898 IF AO$="X"THEN400
1000 PRINTCHR$(147)
1010 PRINT"ORGANIZE PRINT ROUTINE"
1011 POKE646,15
1012 PRINT"■MATERIAL WILL REPEAT IF IT IS LESS THAN A FULL
     PAGE.■";
1013 PRINT"PRESS THE RUNSTOP KEY TO   STOP THE PRINTING."
     :POKE646,11
1014 PRINT"■DO YOU WANT TO ENTER YOUR OWN PARAMETERSFOR
     PRINTING■"
1015 INPUT QR$
1016 IF LEFT$(QR$,1)<>"Y"THEN1085
1020 PRINT"LEFT MARGIN?(NUMBER OF CHARACTERS INDENT)"
1030 LM=5:INPUTLM:IFLM>70THENPRINT"LIMITS ARE 1 TO 70":GOTO1020
1040 PRINT"LINE LENGTH? (TOTAL CHARACTERS PER LINE":LL=70:
     INPUTLL
1050 IFLL<20ORLE>80THENPRINT"LIMITS ARE 20 TO 80":GOTO1040
1055 TM=3:PRINT"ENTER TOP MARGIN (NUMBER OF LINES)"
1060 INPUTTM:IFTM<1ORTM>65THENPRINT"LIMITS ARE 1 TO 65":
     GOTO1055
1065 BM=3:PRINT"ENTER BOTTOM MARGIN (NUMBER OF LINES)"
1070 INPUTBM:IFBM<1ORBM>65THENPRINT"LIMITS ARE 1 TO 65":
     GOTO1065
1075 RC=1:PRINT"INDICATE STARTING REFERENCE NUMBER"
1080 INPUTRC:IFRC<1ORRC>500THENPRINT"LIMITS ARE 1TO 500":
     GOTO1075
1085 PRINT"■SET TOP OF FORM ON THE PRINTER, PRESS
     ANY KEY TO CONTINUE"
1088 GETQR$:IFQR$=""THEN1088
1090 OPEN4,4,7:CMD4: GOSUB1385
1092 LM$="":FORJ=1TOLM:LM$=LM$+CHR$(32):NEXT:WF=0
1094 FORRC=RCTORE
1096 IFTB$(P)=""ANDTB$(P+1)=""THENP=500:GOTO1100
1100 GOSUB1300
1110 NEXT
1115 IFWF=1THENWP$=LM$+W$:PRINTWP$
1120 PRINT##:CLOSE4
1125 RETURN
1130 PRINTCHR$(147):PRINT"TEXT FILE IS NOW LOADING"
1135 OPEN15,8,15
1140 OPEN5,8,5,"0:TEXTFILE,S,R"
1145 GOSUB1600:IFEF=1THENEF=0:FI=0:GOTO1180
1150 INPUT#5,FI
```

```
1155 FORP=1TOFI:TB$(P)=""
1157 GET#5,C$:IFASC(C$)<>13THENTB$(P)=TB$(P)+C$:GOTO1157
1160 PRINT".";:NEXT
1165 CLOSE5:CLOSE15
1170 PRINT:PRINTFI;"TEXT NOW STORED.PRESS ANY KEY TO
     CONTINUE"
1175 GETQR$:IFQR$=""THEN1175
1180 GOTO400
1200 PRINTCHR$(147):PRINT"TEXT FILE NOW BEING SAVED PLEASE
     WAIT":PRINT
1210 OPEN15,8,15
1220 OPEN5,8,5,"@0:TEXTFILE,S,W"
1225 GOSUB1600:IFEF=1THENEF=0:FI=0:GOTO1275
1230 FORP=1TO500
1235 IFTB$(P)<>""THENFI=P
1240 NEXTP
1245 PRINT#5,FI
1250 FORP=1TOFI
1255 PRINT#5,TB$(P)
1260 NEXTP
1265 CLOSE5:CLOSE15
1270 PRINTFI;"TEXT BLOCKS SAVED. PRESS ANY KEY TO     CONTINUE"
1275 GETQR$:IFQR$=""THEN1275
1280 RETURN
1300 IFTB$(P)=""THENRETURN
1302 TW=ASC(LEFT$(TB$(P),1))
1305 IFTW=94THENGOSUB1365:CC=LM+5:B$="     ":WF=1:GOTO1325
1308 IFTW=43THENGOSUB1365:PRINT:LC=LC+1:GOSUB1375:GOTO1325
1310 IFTW=64THENGOSUB1365:DW=1:PRINTCHR$(14):GOSUB1325:
     GOSUB1365:RETURN
1315 IFTW=95THENGOSUB1365:GOTO1325
1320 TW$=TB$(P):GOTO1330
1325 TW$=RIGHT$(TB$(P),LEN(TB$(P))-1):IFTW$=""THENWF=0:RETURN
1330 IFWF=0THENW$=""
1335 FORZ=1TOLEN(TW$):IFW$=""ANDMID$(TW$,Z,1)=CHR$(32)THENCC=
     CC+1:Z=Z+1
1340 CC=CC+1:W$=W$+MID$(TW$,Z,1)
1345 IFLL-CC<10ANDMID$(TW$,Z,1)=" "THENWF=1:GOSUB1365
1350 IFCC>LLTHENWF=1:GOSUB1365
1355 NEXTZ:IFW$<>""THENWF=1
1360 RETURN
1365 IFWF=1ANDW$<>""THENWP$=LM$+W$:PRINTWP$:W$="":CC=
     LM:LC=LC+1
1370 IFDW=1THENPRINTCHR$(15);:DW=0
1375 IFLC<(66-BM)THEN1399
1380 FORI=1TOBM:PRINT:NEXT
1385 FORI=1TOTM:PRINT:NEXT
1390 LC=TM
```

```
1399 WF=0:RETURN
1500 IFLEN(TB$(P))<1THENTB$(P)="":W$="":N1=240:GOTO1530
1510 TB$(P)=LEFT$(TB$(P),LEN(TB$(P))-1)
1520 IFN1>10RN1=1THENN1=N1-1
1530 RETURN
1600 INPUT#15,E1$,E2$,E3$,E4$
1610 IFVAL(E1$)>0THENPRINTE1$,E2$,E3$,E4$:CLOSE5:CLOSE15:EF=1
1620 RETURN
2000 POKE646,0
2001 PRINT"■1. ALWAYS EDIT AND REVIEW YOUR TEXT       BEFORE ";
2003 PRINT"YOU SAVE IT ON THE DISK."
2005 PRINT"■2. PUT YOUR TEXT FILES ON A FORMATTED
     DISK. DO ";
2010 PRINT"NOT USE THE PROGRAM DISK."
2015 PRINT"■3. DO NOT TRY TO PRINT TEXT LOADED FROM
     THE DISK DURING THE ",
2020 PRINT"SAME RUN AS YOU     PRINT TEXT ENTERED FROM
     THE KEYBOARD."
2030 RETURN

READY.
```

Chapter 13

This promises to be a short chapter. It will deal with one of the cheapest peripherals obtainable: the joystick. Mercifully the joystick control on the C-64 is simple.

USING THE JOYSTICK

As you might expect, there is an address to peek for each joystick port: 56321 for one and 56320 for the other. My machine had the two ports re-

```
1 REM:JOYSTICK CONTROL CORRECTED VERSION
3 REMCONTROL PORT 1
4 REM CONTROL PORT 2 56320
5 PRINTCHR$(147)
100 FORK=1TO10:READR$(K):NEXT
110 DATA"N","S","","W","NW","SW"
120 DATA"", "E","NE","SE"
130 PRINT"DIRECTION IS:::::"
140 GOSUB 190
150 IFR$(JD)=""THEN170
160 PRINTR$(JD);" ";
170 IFFR=16THEN140
180 PRINTCHR$(147)
185 GOTO130
190 JD=PEEK(56321):FR=JDAND16:JD=15-(JDAND15)
200 RETURN
```

Fig. 13-1. The Joystick program.

versed when compared with what the *Programmer's Reference Guide* had to say about it. In addition, the program that is included in the reference guide not only ran in reverse but also omitted some of the directions. Before I go any further, Fig. 13-1 shows the corrected version. If your joystick does not work with this program, unplug it and put it in the other port. Make a note for future reference.

Line 100 assigns the eight compass directions that correspond to the possible movements of the joystick to the R$ array. Line 160 prints the result of that action. Line 190 is the one that does the work of reading not only the stick movement but also the fire button. The variables used are JD for Joystick Direction and FR for FiRe.

In actual use, the program lines could contain information or formulas that would steer something, maybe a sprite, across the screen in response to the movement of the joystick. Manipulation of two joysticks would simply mean dealing with the two addresses instead of just one. Experiment and enjoy yourself!

Chapter 14

Take a look at the scenarios of a number of computer games and you will see that they are all roughly the same. The bulb accompanying the full-color advertisement will run somewhat as follows (pick and chose items exactly as you please):

You are a jet-fighter pilot/in charge of a space ship/lost in the desert/caves/on an alien planet/in downtown Chicago/in the sewers of London. Your mission is to overcome the enemy in the form of: air-to-air missiles, alien space fighers, desert creatures malformed from years of inbreeding/alien planet creatures/panhandlers. For each creature destroyed you score a certain number of points but deplete the energy you began with. The purpose of the game is to reach the end, either with a large number of points, or without being overcome yourself, or both.

GAME PROGRAMS

Such games fall into two categories: the verbal type and the full-graphic, sound-effect type. On occasion they might be somewhat mixed.

You can write your own games following the above notions but to be really original you must come up with something really quirky. In the TV show M.A.S.H. the inmates of the Swamp play a most unlikely game in which moves appear to be made on a chess board; one scores with a royal flush and wins with a touchdown! Perhaps one could produce the ultimate computer game in which each barrel collected, filled, and dumped in the right spot causes a flying saucer to be destroyed thus providing the player with a new clue to the location of the hidden treasure. This in turn gives you three free turns at a game of Hangman. For each time you win, you get extra energy to escape from the desert island you have somehow reached by overcoming the Zombies, Zorks and Cruddy-Critters. You win when you reach the control room of the alien cruiser and take control of the entire planet. The ultimate game's name would be Nightmare!

There are many magazines available that print programs of all sorts, including games. There are one or two magazines that devote their pages solely

to VIC-20 and C-64 computers. Each issue contains a game or two, often written by readers but sometimes by expert programmers, although the latter are likely to write their programs in machine code and try to sell them to software houses.

The process of typing in a new game can be arduous, particularly if it is a long program. There is plenty of room for typing errors, and, if you are just beginning programming, you will have a terrible time trying to trace the error. It is far better to try and understand what the program does and how it does it by careful examination of the code. Some programs will be very well documented with plenty of REM statements indicating which section of the program you are looking at and what it is supposed to do. If you have typed in a program from a book or magazine and can get a copy of your listing on a printer, then do so. You can then compare your version with the printed version and spot any errors in a short space of time. Often, errors will be as simple as forgetting the $ on a string variable or leaving out a parenthesis.

BASIC is often blamed for sloppy programming techniques, and there are proponents of other languages who argue that programming is a science and should therefore be highly structured. The truth is that programming is an art whichever language you use and that sloppy programming is to be blamed on the programmer, not the language. There are sloppy speakers and writers of natural languages too, yet we do not blame English or German for the deficiencies of the user!

There are certain ways in which you can write efficient programming code. The first rule is to keep it simple. You will have noticed in the SPRITE programs that the poked addresses are clustered; that is, the initial address is labeled by a variable and later addresses are indicated by the variable + X. This allows the writer to keep in mind which address is being referred to in every case. It is useful to add a REM indicating which variable refers to what address.

It is also a good idea to keep various sections of the program in discrete sections. This allows for checking of those sections on an individual basis prior to linking them together.

CONVERTING PROGRAMS

Converting programs from the VIC-20 for use on the C-64 is not all that difficult. You will need a copy of the *Programmer's Reference Guide* for each machine or at least a complete memory map for each computer.

A memory map is merely a list of the addresses used for each function and attribute of the computer. Screen, color, sound, and start of BASIC addresses are different for each machine and must be changed accordingly. A point of confusion for those unfamiliar with the VIC-20 is the fact that some of the addresses change when memory is added to the machine; you will thus find some programs for the expanded version with different addresses from those for the ordinary 3.5K version.

Converting programs from other machines can be a little difficult. In the first place, not all versions of BASIC are the same. Each machine has the fundamental vocabulary, to be sure, but then there are additional *words* that refer to specific functions. HTAB and VTAB will be seen in some programs. These refer to specific spots on the screen and must be translated into Commodore's screen and color memory locations.

Peeks and pokes are very tricky things to handle when working with a machine of a different color, and a memory map for the machine in question is absolutely vital to successful translation. Often, discretion is the better part of valour and the program, once understood in its original version, should be rewritten for the new machine.

Attempts to translate from an entirely different language into Commodore BASIC (or any other BASIC, for that matter) should be avoided until you really know what you are doing—and still avoided even then. Far better to write from scratch than scratch what you have written.

This chapter began by talking about games and then seems to have wandered off into the nether regions of translations from forms of BASIC programs written for computers other than the C-64. There is a reason for this. You will find that there are many more programs written for all the other machines than for the C-64 alone. Some magazines reproduce versions for a selection of the most

popular machines all of which have similar attributes in terms of sound, color, joystick input, and so forth. Most of the programs, usually games, will make some use of graphics.

I have chosen to present the listing of a program which was written for the most inexpensive computer around; the popular ZX81 or Timex/Sinclair 1000. The program is shown in Fig. 14-1. The game tries to show how artificial intelligence can be simulated on a computer.

The computer knows a few animals. By asking you questions it tries to guess the indentity of the animal you are thinking about. In addition, it asks you to tell it a question that is appropriate to any animal it does not already know. It stores both the question you have given it and the name of the animal, and uses the information for future games as long as you save the entire program.

In many ways, the program runs like a file program. You store information in a new section each time you run the program. If you begin by typing the word run, however, you lose the gathered information and the game starts afresh. In order to make use of what the computer has learned you must enter GOTO START. Then all the information on animals absorbed can be drawn upon by the computer.

This is a remarkable program. It will amaze your friends and even astonish you. In order to use it on the C-64 you must translate it from the ZX81 BASIC.

Note that the listing shows single commands on each line. Sinclair BASIC does not support multistatement lines. Some of the commands differ from those with which you should now be familiar.

The first unusual one you will meet is the pause statement. Substitute a simple FORD=1TO3000:NEXT for any PAUSE you see. The value for the second number is up to you.

CLS is the command to clear the screen. Use CHR$(147) instead.

PRINT AT does not exist on the C-64. Instead you must use TAB or SPC. The ZX81 command allows you to print material on the screen by indicating line and column. The only way to do this on the C-64 is to use screen and color pokes, which is awkward. Present the material on the screen as you see fit, as long as it is readable!

FAST and SLOW exist on the ZX81 because the machine is slow in the ordinary mode. The computer is trying to look after the image on the screen at the same time as it is doing its computing. FAST stops the screen output so that all the energy can be used for computing. The C-64 has no need of these commands, therefore they can be ignored and the lines omitted.

INKEY$ is the equivalent of GET, which has been dealt with earlier. GET is placed in a one-line loop but the following lines still check for input.

You will notice some statements that read GOSUB GET or GOSUB CLEAR SCREEN. This sort of thing will not work on the C-64. You can only branch to a specific line number. The lines which place the appropriate line numbers in the variables are at 930 to 980. Each time you see a branch that addresses a variable, substitute the line number. GOSUB GET would become GOSUB3010, although it is likely that some of the line numbers will change.

The form for DIM must be altered according to the C-64 format. The DIM statement should only contain the 101, without the 20 or the 45 seen in this listing.

The format for LEN will also change to the C-64 type. Experiment with using the MID$ function when you see items like T$ (I, To K).

Line 2000 is a memory check included to make sure that you do not run out of memory. It is a good idea to include such a line in the C-64 format in your program. Even with 38K to play with, you can easily fill it all after a couple of hours playing.

It is very likely that you will have to do a fair amount of juggling with the program before it runs as it should. Do not let that deter you, for that is the way with most programs. For obvious reasons this program is not available on the disk or tape which contains the other programs in this book, but it is the only one not so included!

Having converted this program, other non-graphic programs you might find for machines other than the C-64 will hold no terrors. ZX81 BASIC is one of the more difficult BASICs to convert.

Fig. 14-1. The Animal program.

```
  5 REM "ANIMAL"
 10 PRINT " IF YOU HAVE USED THE RUN COMMAND ALL STORED DATA
    HAS BEN LOST"
 15 PRINT
 20 PRINT " TO MAKE USE OF WHAT THE ZX81     HAS LEARNED,
    PRESS break,RELOAD  FROM TAPE "
 25 PRINT " AND THEN USE "" GOTO START"" TO BEGIN PROGRAMME"
 30 PRINT
 35 PRINT AT 10,0;" IF YOU MAKE A MISTAKE IN ENTER- ING
    MATERIAL,YOU CAN CORRECT IT"
 40 PRINT "BY TYPING ""S"""
 50 PRINT "PRESS enter TO BEGIN"
 55 PAUSE 3E4
 90 FAST
180 GOSUB 900
250 CLS
260 PRINT "THINK OF AN ANIMAL, AND I WILL  TRY TO GUESS WHAT
    IT IS"
270 PRINT AT 21,0;"PRESS enter WHEN YOU ARE READY"
295 CLS
310 LET C$=""
320 FOR Z=1 TO NS
322 GOSUB ASK
324 NEXT Z
330 LET K=LEN C$
333 FOR I=NS+1 TO N
337 IF T$(I, TO K)=C$ THEN GOTO 350
340 NEXT I
342 GOTO 450
350 LET Z=I
352 LET I=N
354 NEXT I
360 GOSUB ASK
362 GOTO 330
450 REM *GUESSED OR GIVE UP*
460 IF A$="Y" THEN PRINT G$
465 IF A$="Y" THEN GOTO 700
467 GOTO MEMCHECK
470 PRINT "I GIVE UP,WHAT IS IT?"
475 INPUT M$
480 IF M$="" THEN GOTO 475
482 IF LEN M$>=35 THEN PRINT "NAME TOO LONG, PLEASE MODIFY"
483 IF LEN M$>=35 THEN GOTO 475
485 PRINT "          ";M$
490 PRINT
495 LET H$=Q$(Z)(7 TO )
497 GOSUB CLEAR SCREEN
```

```
500 PRINT "TELL ME A GOOD QUESTION ABOUT A ";H$
520 INPUT N$
521 IF N$="S" THEN GOTO 5010
523 IF N$="" THEN GOTO 520
525 IF LEN N$>45 THEN PRINT "YOUR QUESTION IS A LITTLE TOO
    LONG, PLEASE MODIFY"
527 IF LEN N$>45 THEN GOTO 520
530 IF N$(LEN N$)<>"?" THEN LET N$=N$+"?"
532 GOSUB CLEAR SCREEN
533 PRINT N$
535 PRINT
540 PRINT "WHAT WOULD BE THE ANSWER  FOR ";M$;"?";" ";
550 GOSUB GET
560 LET R$=INKEY$
565 IF R$<>"S" AND R$<>"Y" AND R$<>"N" THEN GOTO 550
567 IF R$="S" THEN GOTO 5040
570 PRINT ("yes" AND R$="Y")+("no" AND R$="N")
575 PAUSE 60
580 PRINT
625 LET Q$(Z)=N$
655 LET X$=T$(Z)
660 GOSUB TRUNCATE
665 LET T$(N+1)=T$(Z, TO K)+"Y"
670 LET T$(N+2)=T$(Z, TO K)+"N"
675 LET Q$(N+1)="IS IT "+(M$ AND R$="Y")+(H$ AND R$="N")+"?"
680 LET Q$(N+2)="IS IT "+(M$ AND R$="N")+(H$ AND R$="Y")+"?"
690 LET N=N+2
695 GOSUB CLEAR SCREEN
700 PRINT "WOULD YOU LIKE TO DO THAT AGAIN?"
710 GOSUB GET
740 LET A$=INKEY$
750 IF A$="Y" THEN GOTO 250
760 IF A$<>"N" THEN GOTO 710
800 CLS
810 PRINT "PREPARE TAPE RECORDER FOR SAVE"
820 PRINT
830 PRINT "PRESS play AND record, AND THEN enter  TO SAVE
    COMPLETE FILE"
840 PAUSE 4E4
850 SAVE "ANIMAl"
860 CLS
870 GOTO 240
900 REM *INITIALISATION*
910 DIM T$(101,20)
920 DIM Q$(101,45)
930 LET START=240
940 LET ASK=1350
```

```
 950 LET TRUNCATE=1510
 960 LET CLEAR SCREEN=2000
 970 LET MEMCHECK=2510
 980 LET GET=3010
1010 LET N=11
1020 LET NS=3
1030 LET T$(1)="S"
1040 LET T$(2)="S"
1050 LET T$(3)="S"
1060 LET T$(4)="NNN"
1070 LET T$(5)="NNY"
1080 LET T$(6)="NYN"
1090 LET T$(7)="NYY"
1100 LET T$(8)="YNN"
1110 LET T$(9)="YNY"
1120 LET T$(10)="YYN"
1130 LET T$(11)="YYY"
1140 LET Q$(1)="DOES IT HAVE FOUR LEGS?"
1150 LET Q$(2)="IS IT DOMESTIC?"
1160 LET Q$(3)="DOES IT EAT MEAT?"
1170 LET Q$(4)="IS IT A WORM?"
1180 LET Q$(5)="IS IT AN EAGLE?"
1190 LET Q$(6)="IS IT A CHICKEN?"
1200 LET Q$(7)="IS IT A MAN?"
1210 LET Q$(8)="IS IT AN ELEPHANT?"
1220 LET Q$(9)="IS IT A WOLF?"
1230 LET Q$(10)="IS IT A COW?"
1240 LET Q$(11)="IS IT A DOG?"
1250 LET G$="GOOD, I GUESSED IT."
1280 RETURN
1350 GOSUB CLEAR SCREEN
1360 PRINT Q$(Z);" ";
1370 GOSUB GET
1410 LET A$=INKEY$
1420 IF A$="Y" OR A$="N" THEN GOTO 1440
1430 GOTO 1370
1440 LET C$=C$+A$
1450 PRINT ("YES" AND A$="Y")+("NO" AND A$="N")
1460 RETURN
1510 FOR K=1 TO LEN X$
1520 IF X$(K)=" " THEN GOTO 1540
1530 NEXT K
1540 LET K=K+1
1550 RETURN
2000 IF PEEK 16442<=5 THEN CLS
2010 RETURN
2510 GOSUB CLEAR SCREEN
2520 IF N<=99 THEN GOTO 470
```

```
2530 CLS
2540 PRINT "NO ROOM FOR NEW ANIMALS"
2550 PRINT AT 5,10;"menu"
2560 PRINT AT 10,0;"1.ERASE CURRENT ZOO AND BEGIN   AGAIN"
2570 PRINT "2.CONTINUE PLAYING WITH CURRENT ZOO"
2580 PRINT "3.SAVE ZOO"
2590 PRINT "4.END"
2600 PRINT AT 21,0;"ENTER NUMBER OF CHOICE"
2610 LET A$=INKEY$
2630 IF A$="1" THEN GOTO 180
2640 IF A$="2" THEN GOTO START
2650 IF A$="3" THEN GOTO 800
2660 IF A$="4" THEN STOP
2670 GOTO 2610
3010 SLOW
3020 IF INKEY$<>"" THEN GOTO 3020
3030 IF INKEY$="" THEN GOTO 3030
3040 FAST
3050 RETURN
5015 PRINT "ENTER NEW ANIMAL"
5020 INPUT M$
5030 GOTO 482
5040 CLS
5042 PRINT "ENTER NEW QUESTION"
5045 FAST
5050 INPUT N$
5060 GOTO 525
```

Magazines and software houses will often pay for good program translations. See what you can do!

A FINAL NOTE

The Commodore world grows apace. Since the introduction of the VIC-20 and C-64 and its companion, the Commodore Executive, the interest in microcomputers has grown considerably.

Not only are there books written which deal specifically with those machines, such as this book and its companion on the VIC-20, but there are also magazines that either include some material for these machines or deal with them alone. Most magazines fall into the former category. In the latter group are *Compute Gazette* and *Commander.*

Each magazine includes detailed material on everything from beginning computing to machine code. All magazines that deal with Commodore machines include fine examples of games and utility programs. Most of them have been written by people who were the veriest tyros but who persevered and became expert programmers in a few months.

Such is the demand for material that good articles, written for any level of programming, dealing with any application of computers in the real (or imaginary) world are likely to be considered for publication.

It is a good plan to take out a subscription for one or more magazines, for there is a good chance that bookstands will run out of your favorite, leaving you chomping at the bit with frustration because you cannot read the follow up to a series of articles.

Appendices

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
The Talk Program

The Talk program, which is listed in full in Fig. B-1, 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 C-64 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?"

```
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
```

Fig. B-1. The Talk program.

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 acknowledgement of the source of the notion.

Appendix C

Converting Programs to Commodore 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 C-64. 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 Commodore 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 C-64, 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 Commodore 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 Commodore 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 C-64 it cannot be used for this purpose. The best way to duplicate the action of print at statements is to use cursor controls of 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 VIC-20, then watch out. Unless you understand absolutely thoroughly the precise, exact address referred to in the utmost detail, you will send your C-64 to a place where there aren't any phones!

Some commands you will see will refer to graphics. Most do not apply to the C-64.

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	draws a circle.
PAINT		colors in an area.
DRAW		provides an easy way to draw lines.
LINE		draws 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 can be approximated using the pokes to screen and color memory discussed quite early in this book.

UNPLOT is a statement that erases areas where material had previously been plotted. CHR(32) serves as well as anything in most circumstances.

Appendix D
Programming Problems

There will be times when you will have difficulty entering material. Somehow or other, whatever you try to do the C-64 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 C-64 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 appropriate 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 them FORM1, FORM2, FORM3, etc. It is all perfectly logical to you, but the computer will interpret the labels in an entirely unexpected fashion. The C-64 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, 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 C-64 will pick out the OR and try to read it as a reserved word. Just be careful when assigning variable names!

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$(20) followed by a for-next loop that allows you to indicate the number of entries: FOR1 = 1TOX: INPUT A$(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 C-64 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 ERROR This appears 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 FOR1=0TO10, yet there are only 10 items in a 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 although it should not often happen on the C-64. 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 53248 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 because it obviates the necessity of a series of if-then statements.

OVERFLOW There is a limit to a size of the numbers that the C-64 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, 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 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 func-

tion, 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 C-64 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 theC-64 will not. This does not mean that the C-64 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.

Appendix F
The Programs

The appendix contains listings of many of the programs in the book. They are listed here in complete, accurate form for your convenience in typing them into your C-64. There are some additional programs that were not in the book which use programming techniques that have been presented. Experiment with them to better understand how to get more out of your computer.

Program 1: Demo

This program is a menu-driven program that shows what a computer can do, including teaching, calculating, and sorting.

```
2 POKE53281,1:DIMA$(30)
5 PRINT"■"
10 PRINT:PRINT"THIS IS A DEMONSTRATION OF THE KINDS OF"
12 PRINT"THINGS A COMPUTER CAN DO."
15 FORD=1TO4500:NEXT
17 PRINT:GOTO1650
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"
36 PRINT"■5. END"
40 PRINT:PRINT"PLEASE PRESS THE APPROPRIATE NUMBER"
45 INPUTA
46 IFA=5THEN END
```

```
50 ON A GOTO1000,2000,3145,3000
56 )T
154 +
1000 PRINTCHR$(147)
1010 PRINT"CALCULATION ROUTINE"
1015 PRINT:PRINT"A LADY OPENS AN ACCOUNT AND DEPOSITS $1 ON
     THE FIRST DAY"
1020 FORD=1TO4000:NEXT
1025 PRINT:PRINTTAB(2)"EACH DAY THEREAFTER SHE DEPOSITS"
1030 PRINTTAB(2)"DOUBLE THE AMOUNT OF THE PREVIOUS DAY"
1035 FORD=1TO4500:NEXT
1040 PRINT:PRINT" YOU STATE HOW MANY DAYS SHE DOES THIS"
1045 PRINT" AND I WILL TELL YOU THE TOTAL AMOUNT AT THE END OF
     THE PERIOD"
1050 INPUTB
1055 T=0
1060 FORN=1TOB
1065 S=2↑N
1070 T=T+S
1075 NEXTN
1077 PRINTCHR$(147)
1080 PRINTTAB(3)"AFTER ";B+1;"DAYS, THE TOTAL AMOUNT IN"
1083 PRINT"THE ACCOUNT IS $";T;"."
1085 FORD=1TO5000:NEXT
1087 PRINTCHR$(31)
1090 PRINT:PRINT"WOULD YOU LIKE TO TRY THAT AGAIN?"
1095 INPUT"YES OR NO";A$
1100 IF A$="YES" THEN 1000
1105 INPUT"WOULD YOU LIKE TO TRY A DIFFERENT
     CALCULATION PROBLEM";A$
1107 IFA$="YES" THEN 1150
1110 IFA$="NO" THEN 25
1150 PRINT"□"
1152 PRINT"THIS IS A NEW PROBLEM"
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"□"
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"■":PRINT:PRINT"THE LAST SUBJECT GOT ";B
1220 FORD=1TO4500:NEXT
1225 PRINT"■■"
1230 PRINTCHR$(147)
1235 PRINT"NOW YOU GIVE ME A PROBLEM"
1240 FORD=1TO1000:NEXT
1245 PRINT"■": PRINT"CHOOSE FROM THE LIST:-"
1250 PRINT"1.MULTIPLY 2.DIVIDE"
1255 PRINT"3.ADD 4.SUBTRACT"
1257 PRINT"5. RETURN TO MAIN MENU"
1260 INPUTB
1263 IFB=5GOTO20
1265 ON B GOTO1300,1400,1500,1600
1300 PRINTCHR$(147)
1305 PRINT"ENTER TWO NUMBERS:-"
1307 PRINTCHR$(14)"(ONE AT A TIME)"
1310 INPUTX,Y
1312 PRINTCHR$(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
```

```
1555 PRINT:PRINT
1560 PRINTCHR$(147)
1565 GOTO 1245
1600 PRINTCHR$(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)
1645 GOTO1245
1650 PRINT"THE COMPUTER IS A VERY POWERFUL TOOL FOR ALL KINDS
     OF ACCOUNTING."
1655 PRINT:PRINT" IT CAN ALSO ASSIST IN THE PREPARATION"
1658 PRINT"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 ALTERED IN ANY"
1675 PRINT "WAY THE WRITER CHOOSES ■BEFORE "
1678 PRINT"■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  SHORT TIME"
1700 PRINT" BY MEANS OF A ■PRINTER.■"
1705 FORD=1TO10000:NEXT
1710 PRINTCHR$(147)
1715 PRINTTAB(1)"A ■MODEM■ IS A DEVICE THAT ALLOWS YOU TO"
1720 PRINT"TRANSMIT INFORMATION FROM ONE COMPUTER■ TO ANOTHER
     VERY RAPIDLY";
1725 PRINT" OVER THE         TELEPHONE."
1735 PRINT"ALL THE WORLD IS AT YOUR FINGER TIPS."
1740 FORD=1TO11000:NEXT
1745 PRINTCHR$(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 EMPHASIZED":PRINT
2045 PRINT"■IN REVERSE■";" ■AND IN■":PRINT
2050 PRINT"■A";" ■VARIETY ";"■OF";" ■COLORS"
2052 FORD=1TO8000:NEXT
2055 PRINT"■"
2060 PRINT"■"
2065 PRINTTAB(5)"INSTRUCTIONAL PROGRAMS 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"
2082 PRINT"WITHOUT THE STUDENT BEING AWARE"
2085 PRINT"■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$="YES"THEN2200
2140 PRINT"■"
2145 PRINT"YOU ARE RIGHT"
2150 PRINT"A CAREFULLY PREPARED PROGRAM WILL ALLOW"
2155 PRINT"A STUDENT TO WORK AT  HIS OR HER OWN RATE"
2160 FORD=1TO7000:NEXT
2175 PRINT"ALTHOUGH IT IS POSSIBLE TO INCREASE THE LEARNING
     RATE"
2180 PRINT"■"
2185 PRINT"ONLY CERTAIN TYPES OF COMPUTERS CAN BE USED TO
     TEACH--YES OR NO?"
2190 INPUTY$
2192 IFY$="NO"THENPRINT"YOU ARE RIGHT.":GOTO2305
2195 GOTO2300
2200 PRINT"■"
2205 PRINT"WHILE IT IS POSSIBLE TO CONTROL OR EVEN INCREASE
     THE LEARNING"
2210 PRINT"RATE, MOST PROGRAMS 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"
2325 ESSENTIAL TO THE MATERIAL,
2325 PRINT"OBVIOUSLY A COMPUTER SUCH AS THE"
2327 PRINT"COMMODORE 64 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"
2357 PRINT"CAI MATERIAL IS A LENGTHY PROCESS."
2360 PRINT"YET THE MATERIAL 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  SUIT
     NEW AND CHANGING";
2385 PRINT" REQUIREMENTS."
2390 FORD=1TO6000:NEXT
2395 POKE646,4
2398 PRINT:PRINT:
2399 FORD=1TO5000:NEXT:PRINT:PRINT:PRINT
2405 POKE646,5
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(5)"■└──────────┘"
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 MANY NAMES YOU ARE
     GOING TO ENTER"
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$="YES"THEN3045
3143 GOTO20
3145 PRINTCHR$(147)"A COMPLETE FILING SYSTEM CAN BE EXAMINED"
3150 PRINT
3165 FORD=1TO5000: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 FOR THE RETRIEVAL OF MATERIAL";
3195 PRINT" IN ANY ORDER."
3200 FORD=1TO8000:NEXT
3205 PRINT"JUST AS WITH A FILING CABINET, YOU MUST KNOW HOW
     MUCH SPACE YOU HAVE
3210 PRINT"BUT FINDING ITEMS IS AS EASY AS  TYPING  IN THE
     TITLE"
3215 FORD=1TO10000:NEXT
3220 GOTO20
```

Program 2: 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 3: 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=0TON+1:PRINTA$(I):NEXTI
140 OPEN1,4
150 FORI=0TON+1:PRINT#1,A$(I):NEXTI
160 CLOSE1,4
170 RUN
```

Program 4: 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"
```

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$(125)
21 PRINT TAB(X+ 8)CHR$(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: Homed

This home medicine program demonstrates the use of computer aided diagnosis. It is an example of how useful the C-64 can be in providing fast information. The program is far from complete.

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"■■■■■■■";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"■"
2001 PRINTX$(1),X$(8)
2002 PRINT"■■■■■■■■■■■■■■■■";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$(5)
8012 PRINT"■■■■■■■■■■■■■■■■";A$(97)
8013 FORD=1TO7000:NEXT
8015 GOTO5
8050 PRINT"■"
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"■■■■■■■■■■■■■■■■";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 8: Train

```
1  INPUT"■CHOOSE A NUMBER FOR COLOR";H
2  INPUT"AND TYPE YOUR NAME□";Z$
5  S=54272:POKES+24,15:POKES,220:POKES+1,68:POKES+5,80:POKES+6,
   195
10 POKES+7,120:POKES+8,100:POKES+12,148:POKES+13,195
15 PRINT"□":V=53248:POKEV+21,1
20 FORS1=12288TO12350:READ01:POKES1,01:NEXT
25 FORS2=12352TO12414:READ02:POKES2,02:NEXT
30 FORS3=12416TO12478:READ03:POKES3,03:NEXT
35 POKEV+39,H-1:POKEV+1,68
37 POKEV+23,1:POKEV+29,1:POKEV+23,2:POKEV+29,2:POKEV+23,3:
   POKEV+29,3
40 PRINTTAB(160)"■THE ";Z$;" EXPRESS"
45 P=192
50 FORX=0TO347STEP2
55 RX=INT(X/256):LX=X-RX*256
60 POKEV,LX:POKEV+16,RX
70 IFP=192THENGOSUB200
75 IFP=193THENGOSUB300
80 POKE2040,P:FORT=1TO10:NEXT
85 P=P+1:IFP>194THENP=192
90 NEXT
95 END
100 DATA0,3,224,3,223,240,7,225,255,7,251,255,7,240,60,0,0,24,
    0,0,24
101 DATA0,0,24,255,0,24,159,28,24,25,28,24,25,255,255,25,255,
    255
102 DATA31,255,255,255,255,255,255,255,255,255,255,255,255,
    255,255,255,255
103 DATA60,1,224,24,0,192
104 DATA48,3,224,111,220,240,95,220,255,87,231,255,79,92,60,
    203,224,24
105 DATA235,224,24,120,0,24,255,0,24,191,28,24,121,28,24,25,
    255,255
106 DATA25,255,255,31,255,255,255,255,255,255,255,255,255,255,
    255
107 DATA255,255,255,255,255,255,60,1,224,24,0,192
108 DATA32,3,224,120,7,252,248,15,255,248,11,255,216,15,60,160,
```

```
    30,24,0,30,24
109 DATA0,8,24,255,0,24,159,28,24,25,28,24,25,255,255,25,255,
    255,31,255,255
110 DATA255,255,255,255,255,255,255,255,255,255,255,255,255,
    255,255
111 DATA60,1,224,24,0,192
199 END
200 POKES+4,129:POKES+4,128:RETURN
300 POKES+11,129:POKES+11,128:RETURN
```

Program 9: Umlaut Ö

```
5 REM:UMLAUT O
6 PRINTCHR$(147)
10 OPEN1,4
15 PRINT"PRESS FUNCTION KEY 1 TO PRODUCE CHARACTER ON PRINTER"
20 GETA$:IFA$=""THEN20
30 IFASC(A$)=133THEN70
40 PRINT#1,A$
50 PRINTA$;
60 GOTO20
70 DATA0,0,57,68,68,57,0,0
80 PRINT#1,CHR$(8);
90 FORR=1TO8
100 READA
110 PRINT#1,CHR$(A+128);
120 NEXT:PRINT"*";
130 PRINT#1,CHR$(15)
140 RESTORE:GOTO20
```

Program 10: Sound

```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT
20 POKES+5,88:POKES+6,195
30 POKES+24,15
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,33
80 FORT=1TODR:NEXT
90 POKES+4,32:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177.250
130 DATA25,177,125,28,214,250
140 DATA32,94,250,25,177,250
150 DATA28,214,250,19,63,250
```

```
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

Program 11: Average

```
5 PRINTCHR$(147)
10 PRINT:PRINT:PRINT"THIS PROGRAM COMPUTES AVERAGES"
15 FORD=1TO2000:NEXT
18 PRINTCHR$(147)
19 T=0
20 PRINT:PRINT:PRINT:PRINT"ENTER STUDENT NAME (AND PRESS
   RETURN)"
25 INPUTA$
30 PRINT"HOW MANY MARKS? (TOTAL OF TESTS, HOMEWORK ETC.)"
35 INPUTN
50 FORI=1TON
55 PRINT"ENTER MARK NO. ";I
60 INPUTX
65 T=T+X
70 NEXT
80 M=T/N
85 PRINT"STUDENT ";A$
90 PRINT"TOTAL= ";T
100 PRINT"AVERAGE= ";M
105 PRINT"PRESS 'P' FOR HARD COPY (AND RETURN)"
110 PRINT"PRESS 'C' FOR NEXT STUDENT (AND RETURN)"
115 INPUTC$
118 IFC$="P"THEN140
120 IFC$="C"THEN18
130 IFC$<>"C"THEN110
140 OPEN1,4:CMD1
150 PRINT"STUDENT ";A$,"TOTAL ";T,"AVERAGE ";M
160 PRINT#1:CLOSE1,4:GOTO110
```

Program 12: Scroll

```
1 PRINTCHR$(147)
2 A=53265
3 REM:SCROLL GRAPHIC
10 POKEA,PEEK(A)AND247
20 PRINTCHR$(147)
30 FORX=1TO24:PRINTCHR$(17);:NEXT
40 POKEA,(PEEK(A)AND248)+7:PRINT
50 PRINTTAB(I)CHR$(41)
60 FORP=6TO0STEP-1
70 POKEA,(PEEK(A)AND248)+P
```

```
80 FORX=1TO500:NEXT
85 PRINTCHR$(32)
90 NEXT
```

Program 13: Stave

```
20 PRINTCHR$(147)
200 PRINTCHR$(147):PRINT:PRINT:PRINT
201 FORI=1TO38:PRINTCHR$(100);:NEXT
202 PRINT
203 FORI=1TO38:PRINTCHR$(100);:NEXT
204 PRINT
205 FORI=1TO38:PRINTCHR$(100);:NEXT
206 PRINT
207 FORI=1TO38:PRINTCHR$(100);:NEXT
208 PRINT
209 FORI=1TO38:PRINTCHR$(100);:NEXT
900 POKE52,48:POKE56,48:CLR
910 POKE56334,PEEK(56334)AND254
920 POKE1,PEEK(1)AND251
930 FORL=0TO511:POKEL+12288,PEEK(L+53248):NEXT
940 POKE1,PEEK(1)OR4
945 POKE56334,PEEK(56334)OR1
950 FORI=12576TO12623:POKEI,PEEK(I+25600):NEXT
955 POKE53272,(PEEK(53272)AND240)+12
960 FORWH=12576TO12583:READA:POKEWH,A:NEXT
965 FORHA=12584TO12591:READB:POKEHA,B:NEXT
970 FORQU=12592TO12599:READC:POKEQU,C:NEXT
975 FOREI=12600TO12607:READD:POKEEI,D:NEXT
980 FORSI=12608TO12615:READE:POKESI,E:NEXT
985 FORFL=12616TO12623:READF:POKEFL,F:NEXT
1000 DATA0,0,0,0,120,68,60,0
1001 DATA4,4,4,4,60,68,60,0
1002 DATA4,4,4,4,60,124,60,0
1003 DATA4,6,5,4,60,124,60,0
1004 DATA4,6,5,6,61,124,60,0
1005 DATA32,32,32,44,52,36,56,0
```

Program 14: Color

```
1 REM:COLOR CHANGE OF BACKGROUND
5 PRINTCHR$(147)
10 P=53281
20 INPUTC
30 POKEP,C
35 PRINT"HELLO"
40 FORD=1TO1000:NEXT
50 POKEP,6
60 GOTO5
```

Program 15: Alien Visions

```
199 PRINTCHR$(147)
200 POKE53270,PEEK(53270)OR16
201 INPUTX
210 POKE53283,X
211 FORD=1TO1500:NEXT
212 POKE53281,X
213 FORD=1TO1500:NEXT
214 PRINT:PRINT:PRINT"HELLO"
215 POKE53282,X
216 FORD=1TO1500:NEXT
220 GOTO201
```

Program 16: Letters

```
10 PRINTCHR$(147)
15 POKE646,2
20 PRINT"GUESS MY LETTER"
22 POKE646,1
24 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
25 Y=0
30 X=INT(RND(1)*26)+1
35 X$=MID$(A$,X,1):PRINT
40 PRINT"I'M THINKING OF A LETTER"
50 PRINT"CAN YOU GUESS IT?"
55 FORD=1TO2000:NEXT
56 PRINTCHR$(147)
60 INPUT"ENTER YOUR GUESS";Y$
70 IFY$<X$THENPRINT"TOO LOW":GOTO55
80 IFY$>X$THENPRINT"TOO HIGH":GOTO55
90 IFY$=X$THENPRINT"RIGHT ON!"
92 IFY$=X$THENY=Y+1
98 IFY=10THEN120
100 GOTO30
120 PRINT"YOU SCORED TEN POINTS!":GOTO30
```

Program 17: Who

```
1 PRINTCHR$(147)
2 INPUT"HOW MANY PEOPLE";X
4 DIMA$(X)
5 DIMB$(X)
6 FORI=1TOX
7 REM:THIS PROGRAM IS THE BEGINNING OF A GOOD IDEA. FINISH
  IT(& FIND THE ERROR!)
10 PRINT"I WOULD LIKE YOU TO ENTER YOUR FIRST    NAME, PLEASE"
```

```
20 INPUTA$(I)
30 PRINT"THANKYOU ";A$(I)
40 PRINT"NOW I WOULD LIKE YOU TO ENTER SOME      DETAIL ABOUT
   YOURSELF BY FILLING"
50 PRINT"IN THE GAP IN THE STATEMENT 'I AM A ----"
60 INPUTB$(I)
70 PRINTCHR$(147)
80 PRINT"THANKYOU ";A$(I)
85 PRINT"YOU ARE A ";B$(I)
90 NEXTI
100 PRINT"PLEASE STATE WHAT YOU ARE BY COMPLETING THE SENTENCE:
    'I AM A -----"
120 INPUTY$
130 IFY$=B$(I)THENPRINT"IF YOU ARE ";B$(I);" THEN YOU MUST BE"
140 FORD=1TO1000:NEXT
150 PRINT"JUST A MOMENT WHILE I THINK--"
160 PRINT"AH ";A$(I)
170 FORD=1TO4000:NEXT
180 GOTO100
```

Program 18: Addresses

This program can be useful when you develop your own character sets.

```
1 REM:THIS PROGRAM CALCULATES SCREEN ADDRESS- ES FOR EACH
  CHARACTER
5 REM SCREEN ADDRESSES
6 OPEN1,4
7 PRINTCHR$(147)
10 INPUT"ENTER CHR$ VALUE";X
20 A=12288
25 Y=A+(X*8)
30 PRINT#1,Y,Y+7,CHR$(X)
35 PRINTY,Y+7,CHR$(X)
60 GOTO10
```

Program 19: Endless Addresses

Sample output is included below.

```
1 REM:SOME ODD THINGS HAPPEN WITH PROGRAMM BOTH ON THE SCREEN
  AND THE PRINTER.
2 REM:STICK WITH IT!
5 REM SCREEN ADDRESSES
6 OPEN1,4
7 PRINTCHR$(147)
10 A=12288
12 FORX=0TO127
25 Y=A+(X*8)
```

```
27 PRINT#1,X
30 PRINT#1,Y,Y+7,CHR$(X)
35 PRINTY,Y+7,CHR$(X)
60 NEXTX
70 PRINT#1:CLOSE1,4
```

65		
12800	12815	A
66		
12816	12823	B
67		
12824	12831	C
68		
12832	12839	D
69		
12840	12847	E
113		
13192	13199	●
114		
13200	13207	–
115		
13208	13215	●
116		
13216	13223	|
117		
13224	13231	/

Program 20: Programmed Characters

```
5 REM:PROGRAMMABLE CHARACTERS.
  PRESS FUNCTION 1
6 REM: TO SEE PROGRAMM IN ACTION
7 PRINTCHR$(147):PRINT"PRESS F1"
10 POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND251
20 FORI=0TO63:FORJ=0TO7:POKE12288+I*8+J,PEEK(53248+I*8+J)
30 NEXTJ:NEXTI
40 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1
50 POKE53272,PEEK(53272)AND240)+12
60 FORCH=60TO63:FORBY=0TO7
70 READNU
80 POKE12288+(8*CH)+BY,NU
90 NEXTBY:NEXTCH
100 PRINTCHR$(147)TAB(255)CHR$(60);
110 PRINTCHR$(61)TAB(55)CHR$(62)CHR$(63)
120 GETA$:IFA$=""THEN120
130 POKE53272,21
```

```
140 DATA4,6,7,5,7,7,3,3
150 DATA32,96,224,160,224,224,192,192
160 DATA7,7,31,95,143,127,255,255
170 DATA224,224,224,248,248,248,240,224
180 END
```

Program 21: Instant Color

```
199 PRINTCHR$(147)
200 POKE53270,PEEK(53270)OR16
201 INPUTX
210 POKE53283,X
211 FORD=1TO1500:NEXT
212 POKE53281,X
213 FORD=1TO1500:NEXT
214 PRINT:PRINT:PRINT"HELLO"
215 POKE53282,X
216 FORD=1TO1500:NEXT
220 GOTO201
```

Program 22: Time

```
10 X=TI/60
20 Y=X/60
30 PRINTY"MINUTES SINCE POWER UP"
```

Program 23: Scoop

```
10 S=54272
15 FORX=15TO1STEP-1
20 FORI=100TO250
30 POKES,I:POKES+1,I:POKES+24,X
35 POKES+7,I/3.5:POKES+8,I/3.5
36 POKES+14,I/2.5:POKES+15,I/2.5
40 NEXT:NEXT
```

Program 24: Noise

Enter an attack/decay value for A, a sustain/release value for B, and a waveform value for C.

```
5 PRINTCHR$(147)
10 S=54272
15 INPUTA,B,C
18 POKES+4,C
20 POKES+5,A:POKES+6,B
30 POKES,75:POKES+1,34:POKES+24,15
40 FORD=1TO2000:NEXT
100 POKES+24,0
```

```
105 POKES+4,0:POKES+5,0:POKES+6,0
200 GOTO10
```

Program 25: Colors

```
1 REM:BACKGROUND AND BORDER COLORS
5 PRINTCHR$(147)
10 FORBA=0TO15
20 FORBO=0TO15
30 POKE53280,BA
35 PRINT:PRINT:PRINTBA;"BA",BO;"BO"
40 POKE53281,BO
50 FORX=1TO1000:NEXT
60 NEXTBO:NEXTBA
```

Program 26: Binary to Decimal Converter

Remember not to enter anything but zeros and ones. The results are interesting, but meaningless.

```
5 REMBINARY TO DECIMAL CONVERTER
10 INPUT"ENTER 8-BIT BINARY NUMBER";A$
12 IFLEN(A$)<>8THENPRINT"8 BITS PLEASE...":GOTO10
15 TL=0:C=0
20 FORX=8TO1STEP-1:C=C+1
30 TL=TL+VAL(MID$(A$,C,1))*2↑(X-1)
40 NEXTX
50 PRINTA$;"BINARY ";"=";TL;" DECIMAL"
60 GOTO10
```

Program 27: Binary to Decimal Converter for the Printer

```
5 REMBINARY TO DECIMAL CONVERTER
6 OPEN1,4
10 INPUT"ENTER 8-BIT BINARY NUMBER";A$
12 IFLEN(A$)<>8THENPRINT"8 BITS PLEASE...":GOTO10
15 TL=0:C=0
20 FORX=8TO1STEP-1:C=C+1
30 TL=TL+VAL(MID$(A$,C,1))*2↑(X-1)
40 NEXTX
50 PRINTA$;"BINARY ";"=";TL;" DECIMAL"
55 PRINT#1,TL;" DECIMAL"
60 GOTO10
```

Program 28:

This is the first of ten programs that deal with sprites. Experiment to create effects of your own. Use editing instead of typing in each program.

```
5 REMSPRITE MOVING
8 X=0
```

```
10 PRINTCHR$(147)
20 POKE2040,192
30 FORS=12288TO12350+62:POKES,255:NEXT
40 V=53248
50 POKEV+21,1:POKEV+39,X:POKEV+1,100
70 FORI=1TO255:POKEV,I
80 NEXT
85 POKE53281,X+2
90 X=X+1
100 GOTO20
```

Program 29:

```
5 REM SPRITES MOVING DIAGONALLY
10 PRINTCHR$(147)
20 POKE2040,192
30 FORS=12288TO12350+62:POKES,255:NEXT
40 V=53248:Z=255
50 POKEV+21,1:POKEV+39,1
60 FORX=1TO15:POKEV+39,X:FORD=1TO20:NEXT
65 POKE53281,X+2
70 FORI=1TO255:POKEV,I:POKEV+1,Z-I:NEXT
80 NEXT
```

Program 30:

```
5 REM SPRITES MOVING DIAGONALLY
10 PRINTCHR$(147)
20 POKE2040,192
30 FORS=12288TO12350+62:POKES,255:NEXT
40 V=53248:Z=255
50 POKEV+21,1:POKEV+39,1
60 FORX=1TO15:POKEV+39,X:FORD=1TO20:NEXT
65 POKE53281,X+2
70 FORI=255TO1STEP-1:POKEV,I:POKEV+1,I:NEXT
80 NEXT
```

Program 31:

```
5 REM SPRITES MOVING DIAGONALLY
10 PRINTCHR$(147)
20 POKE2040,192
30 FORS=12288TO12350+62:POKES,255:NEXT
40 V=53248:Z=255
50 POKEV+21,1:POKEV+39,1
60 FORX=1TO15:POKEV+39,X:FORD=1TO20:NEXT
65 POKE53281,X+2
70 FORI=255TO1STEP-1:POKEV,I:POKEV+1,Z-I:NEXT
80 NEXT
```

Program 32:

```
5 REM SPRITES MOVING DIAGONALLY
10 PRINTCHR$(147)
20 POKE2040,192:POKE2041,193
30 FORS1=12288TO12350+62:POKES1,255:NEXT
32 FORS2=12352TO12414+62:POKES2,255:NEXT
40 V=53248:Z=255
50 POKEV+21,3:POKEV+39,1
60 FORX=1TO15:POKEV+39,X:POKEV+40,X+1:FORD=1TO20:NEXT
65 POKE53281,X+2
70 FORI=255TO1STEP-1:POKEV,I:POKEV+1,I
72 POKEV+2,Z-I:POKEV+3,Z-I:NEXT
80 NEXT
90 RUN
```

Program 33:

```
5 REM SPRITES MOVING DIAGONALLY
10 PRINTCHR$(147)
20 POKE2040,192:POKE2041,193:POKE2042,194
30 FORS1=12288TO12350+62:POKES1,255:NEXT
32 FORS2=12352TO12414+62:POKES2,255:NEXT
34 FORS3=12416TO12478+62:POKES3,255:NEXT
40 V=53248:Z=255
50 POKEV+21,7:POKEV+29,4:POKEV+23,2
60 FORX=1TO15:POKEV+39,X-1:POKEV+40,X+1:POKEV+41,X+2:
   FORD=1TO20:NEXTD
65 POKE53281,X+3
66 FORA=1TO255:POKEV+4,A:POKEV+5,100
67 NEXTA
70 FORI=255TO1STEP-1:POKEV,I:POKEV+1,I
74 POKEV+2,Z-I:POKEV+3,Z-I
76 NEXTI
79 NEXTX
```

Program 34:

```
5 REM SPRITES MOVING DIAGONALLY
10 PRINTCHR$(147)
20 POKE2040,192:POKE2041,193:POKE2042,194
30 FORS1=12288TO12350+62:POKES1,255:NEXT
32 FORS2=12352TO12414+62:POKES2,255:NEXT
34 FORS3=12416TO12478+62:POKES3,255:NEXT
40 V=53248:Z=255
50 POKEV+21,7:POKEV+29,4:POKEV+23,2
60 FORX=1TO15:POKEV+39,X-1:POKEV+40,X+1:POKEV+41,X+2
65 POKE53281,X+3
```

```
66 FORA=1TO255STEP10:POKEV+4,A:POKEV+5,100
70 FORI=255TO1STEP-1:POKEV,I:POKEV+1,I
74 POKEV+2,Z-I:POKEV+3,Z-I
76 NEXTI
77 NEXTA
79 NEXTX
```

Program 35:

```
5 REMSPRITE MOVING
9 X=0
10 PRINTCHR$(147)
20 POKE2040,192
30 FORS=12288TO12350+62:POKES,255:NEXT
40 V=53248
50 POKEV+21,1:POKEV+39,X:POKEV+1,100
70 FORI=1TO347
72 R=INT(I/256):L=I-R*256
75 POKEV,L:POKEV+16,R
80 NEXT
85 POKE53281,X+2
90 X=X+1
100 GOTO20
```

Program 36:

```
5 REM SPRITES COLLISION
10 PRINTCHR$(147)
20 POKE2040,192:POKE2041,193
30 FORS1=12288TO12350+62:POKES1,255:NEXT
32 FORS2=12352TO12414+62:POKES2,255:NEXT
40 V=53248:Z=255
50 POKEV+21,3:POKEV+39,1
60 FORX=1TO15:POKEV+39,X:POKEV+40,X+1
65 POKE53281,X+2
70 FORI=255TO1STEP-1:POKEV,I:POKEV+1,I
72 IFPEEK(V+30)AND1=2THENPOKEV+21,0
73 POKEV+2,Z-I:POKEV+3,Z-I
75 NEXT
80 NEXT
```

Program 37:

```
5 REM SPRITES MOVING DIAGONALLY
10 PRINTCHR$(147)
20 POKE2040,192
30 FORS=12288TO12350+62:POKES,255:NEXT
```

```
40 V=53248:Z=255
50 POKEV+21,1:POKEV+39,1
60 FORX=1TO15:POKEV+39,X:FORD=1TO20:NEXT
65 POKE53281,X+2
70 FORI=255TO1STEP-.05:POKEV,I:POKEV+1,I:NEXT
80 NEXT
```

Program 38: Special Character Printing

```
5 REM:PRINTER DOT GRPAHICS
10 DATA144,168,197,196,197,168,144,0
20 OPEN1,4
30 PRINT#1,CHR$(8)
40 FORI=1TO8
50 READA
60 PRINT#1,CHR$(A);
70 NEXT
80 PRINT#1:PRINT#1
```

Program 39: Bar Chart Printer

```
5 REM:BAR CHART PRINTER
10 A=128
20 OPEN1,4
30 INPUT"LENGTH";L
40 PRINT#1,CHR$(8);
60 PRINT#1,CHR$(26)CHR$(L)CHR$(255)
70 PRINT#1
```

Program 40: The Hebrew Character Set

```
1 POKE52,48:POKE56,48:CLR
2 FORI=12288TO12936:POKEI,PEEK(I+25600):NEXT
3 POKE53272,(PEEK(53272)AND240)+12
5 PRINTCHR$(147)
8 REM"HEBREW"
10 FORA1=12296TO12303:READA:POKEA1,A:NEXT
20 DATA65,99,52,24,44,70,99,0
30 FORB1=12304TO12311:READB:POKEB1,B:NEXT
40 DATA127,63,2,10,2,127,127,0
50 FORV1=12464TO12471:READV:POKEV1,V:NEXT
60 DATA127,63,1,1,1,63,127,0
70 FORG1=12344TO12351:READG:POKEG1,G:NEXT
80 DATA7,3,1,1,3,7,13,0
90 FORD1=12320TO12327:READD:POKED1,D:NEXT
100 DATA127,127,2,2,2,2,2,0
110 FORH1=12352TO12359:READH:POKEH1,H:NEXT
```

```
120 DATA127,127,1,33,33,33,33,0
130 FORZ1=12496TO12503:READZ:POKEZ1,Z:NEXT
140 DATA6,3,2,2,2,2,2,0
150 FORW1=12472TO12479:READW:POKEW1,W:NEXT
160 DATA6,3,1,1,1,1,1,0
170 FORX1=12480TO12487:READX:POKEX1,X:NEXT
180 DATA127,127,33,33,33,33,33,0
190 FORT1=12448TO12455:READT:POKET1,T:NEXT
200 DATA102,43,83,65,65,67,126,0
210 FORY1=12488TO12495:READY:POKEY1,Y:NEXT
220 DATA12,12,4,8,0,0,0,0
230 FORK1=12376TO12383:READK:POKEK1,K:NEXT
240 DATA127,127,3,19,3,126,126,0
250 FORL1=12384TO12391:READL:POKEL1,L:NEXT
260 DATA96,32,32,62,2,12,8,0
270 FORM1=12392TO12399:READM:POKEM1,M:NEXT
280 DATA108,46,81,65,65,95,94,0
290 FORN1=12400TO12407:READN:POKEN1,N:NEXT
300 DATA14,6,2,2,2,14,15,0
310 FORS1=12440TO12447:READS:POKES1,S:NEXT
320 DATA126,63,33,97,97,98,124,0
330 FORE1=12328TO12335:READE:POKEE1,E:NEXT
340 DATA50,59,17,17,14,28,56,0
350 FORP1=12416TO12423:READP:POKEP1,P:NEXT
360 DATA126,63,17,61,33,63,126,0
370 FORF1=12336TO12343:READF:POKEF1,F:NEXT
380 DATA126,62,49,49,33,63,126,0
390 FORC1=12312TO12319:READC:POKEC1,C:NEXT
400 DATA99,35,20,8,4,63,127,0
410 FOR01=12424TO12431:READ0:POKE01,0:NEXT
420 DATA96,62,1,33,34,36,40,32
430 FORR1=12432TO12439:READR:POKER1,R:NEXT
440 DATA127,127,1,1,1,1,1,0
450 FORU1=12456TO12463:READU:POKEU1,U:NEXT
460 DATA1,82,123,41,73,82,124,0
470 FOR01=12408TO12415:READO:POKEO1,O:NEXT
480 DATA64,82,123,41,73,82,124,0
490 FORI1=12360TO12367:READI:POKEI1,I:NEXT
500 DATA64,126,33,73,65,33,97,0
510 FORJ1=12368TO12375:READJ:POKEJ1,J:NEXT
520 DATA64,126,33,65,65,33,97,0
```

Program 41: The Russian Character Set

```
1 POKE52,48:POKE56,48:CLR
3 FORI=12288TO12936:POKEI,PEEK(I+25600):NEXT
5 POKE53272,(PEEK(53272)AND240)+12
8 REM"RUSSIAN"
10 FORA1=12296TO12303:READA:POKEA1,A:NEXT
```

```
20 DATA0,8,20,20,34,68,65,65
30 FORB1=12304TO12311:READB:POKEB1,B:NEXT
40 DATA0,124,66,64,124,66,66,126
50 FORV1=12464TO12471:READV:POKEV1,V:NEXT
60 DATA0,65,42,28,28,28,42,120
70 FORG1=12344TO12351:READG:POKEG1,G:NEXT
80 DATA0,60,34,32,32,32,32,124
90 FORD1=12320TO12327:READD:POKED1,D:NEXT
100 DATA0,62,34,34,34,34,95,67
110 FORH1=12352TO12359:READH:POKEH1,H:NEXT
120 DATA0,97,50,28,8,28,38,67
130 FORZ1=12496TO12503:READZ:POKEZ1,Z:NEXT
140 DATA0,56,68,4,56,4,68,56
150 FORW1=12472TO12479:READW:POKEW1,W:NEXT
160 DATA0,124,66,66,124,66,66,124
170 FORX1=12480TO12487:READX:POKEX1,X:NEXT
180 DATA0,64,78,81,113,81,81,78
190 FORT1=12448TO12455:READT:POKET1,T:NEXT
200 DATA0,127,73,8,8,8,8,8
210 FORY1=12488TO12495:READY:POKEY1,Y:NEXT
220 DATA0,0,0,68,68,100,84,100
230 FORK1=12376TO12383:READK:POKEK1,K:NEXT
240 DATA0,68,72,112,72,68,66,67
250 FORL1=12384TO12391:READL:POKEL1,L:NEXT
260 DATA0,127,83,19,19,19,83,39
270 FORM1=12392TO12399:READM:POKEM1,M:NEXT
280 DATA0,65,99,85,73,65,65,65
290 FORN1=12400TO12407:READN:POKEN1,N:NEXT
300 DATA0,102,102,102,126,102,102,102
310 FORS1=12440TO12447:READS:POKES1,S:NEXT
320 DATA0,28,34,96,96,96,34,28
330 FORE1=12328TO12335:READE:POKEE1,E:NEXT
340 DATA0,124,64,68,124,68,64,124
350 FORP1=12416TO12423:READP:POKEP1,P:NEXT
360 DATA0,126,102,102,102,102,102,102
370 FORF1=12336TO12343:READF:POKEF1,F:NEXT
380 DATA0,8,62,73,73,73,62,8
390 FORC1=12312TO12319:READC:POKEC1,C:NEXT
400 DATA0,68,68,68,68,68,68,59
410 FORQ1=12424TO12431:READQ:POKEQ1,Q:NEXT
420 DATA0,107,107,107,107,107,127,1
430 FORR1=12432TO12439:READR:POKER1,R:NEXT
440 DATA0,120,68,68,120,64,64,64
450 FORU1=12456TO12463:READU:POKEU1,U:NEXT
460 DATA0,34,34,18,10,6,34,72
470 FORO1=12408TO12415:READO:POKEO1,O:NEXT
480 DATA0,24,36,66,66,66,36,24
490 FORI1=12360TO12367:READI:POKEI1,I:NEXT
500 DATA0,65,67,69,73,81,97,65
```

```
510 FORJ1=12368TO12373:READJ:POKEJ1,J:NEXT
520 DATA0,85,73,67,69,73,81,96
600 PRINT"□"
610 PRINT"ETO KARANDAQ? NET,ETO NE KARANDAQ"
```

Index

Using and Programming the Commodore 64, including Ready-To-Run Programs

If you are intrigued with the possibilities of the programs included in *Using and Programming the Commodore 64, including Ready-To-Run Programs* (TAB BOOK No. 1712), you should definitely consider having the ready-to-run tape or disk containing the software applications. This software is guaranteed free of manufacturer's defects. (If you have any problems, return the tape or disk within 30 days, and we'll send you a new one.) Not only will you save the time and effort of typing the programs, the tape or disk eliminates the possibility of errors that can prevent the programs from functioning. Interested?

Available on tape for Commodore 64, 64K

Available on disk for Commodore 64, 64K

at $19.95
for each tape or disk plus $1.00 each shipping and handling.

I'm interested. Send me:

_____ tape for Commodore 64 (6409S)
_____ disk for Commodore 64 (6410S)
_____ TAB BOOKS catalog
_____ Check/Money Order enclosed for $19.95
plus $1.00 shipping and handling for each tape or disk ordered.
_____ VISA _____ MasterCard

Account No. _____ Expires _____
Name _____
Address _____
City _____ State _____ Zip _____
Signature _____

Mail To: **TAB BOOKS INC.**
P.O. Box 40
Blue Ridge Summit, PA 17214

(Pa. add 6% sales tax. Orders outside U.S. must be prepaid with international money orders in U.S. dollars.)

TAB 1712

ISBN 0-8306-0712-9