GETTING STARTED ON YOUR BBC MICRO

Tim Hartnell and Alex Gollner

GETTING STARTED ON YOUR BBC MICRO

Tim Hartnell & Alex Gollner

Getting Started on Your BBC Micro

Futura
Macdonald & Co
London & Sydney

Also in this series

GETTING STARTED ON YOUR VIC 20
GETTING STARTED ON YOUR ORIC
GETTING STARTED ON YOUR ATARI
GETTING STARTED ON YOUR DRAGON
GETTING STARTED ON YOUR ZX81
GETTING STARTED ON YOUR SPECTRUM

This book is dedicated to the Lower Sixth at Hampstead School.

Thanks to Iris Gollner and Jeremy Ruston.

A Futura Book

First published in Great Britain in 1983
by Futura Publications
A Division of Macdonald & Co. (Publishers) Ltd
London & Sydney

ISBN 0 7088 2443 9

Reproduced, printed and bound in Great Britain by
Hazell Watson & Viney Limited,
Member of the BPCC Group,
Aylesbury, Bucks

Futura Publications
A Division of
Macdonald & Co (Publishers) Ltd
Maxwell House
74 Worship Street
London EC2A 2EN

A BPCC plc Company

CONTENTS

INTRODUCTION

Here in your hands you hold a map to one of the most fascinating journeys you will ever take. Even if you have never touched a computer before you bought your BBC micro, we hope to share with you — step by simple step — the secrets of computing programming.

It's not going to be difficult. We've deliberately included a number of major games in the book, so after you've entered and run the games, you'll discover you've learned to program almost while you were not looking. It's going to be painless and it's going to be fun.

The whole process is greatly simplified by the fact that you're learning on a BBC micro. Representing the leading edge of computer technology, the BBC micro is a computer that was designed to be *friendly*, and easy and rewarding to use.

Come on and join us now, as together we discover how you can make the most of your BBC micro.

Tim Hartnell and Alex Gollner
London, August, 1983

A NOTE ON THE AUTHORS

Tim Hartnell is the most widely published author of books related to the Sinclair computers in the world. Founder of the British National ZX Users' Club and its magazine Interface, he has written many books for the ZX80, T/S 1000 and T/S 2000. He is founding editor of the British bimonthly magazine ZX Computing, and is involved in user-group activities in the US and UK.

Alex Gollner is a 16-year old student living in London. He is still at school and has just passed his "O" level examinations. He has already had one computer book published successfully, GAMES FOR YOUR BBC MICRO — this was also edited by Tim Hartnell.

CHAPTER ONE DISCOVERING THE KEYBOARD

You have just acquired a great machine and you will find it the ideal introduction to the world of computers. As some people are wary of their first computer, we are going to take things slowly.

READ THIS BOOK WITH YOUR COMPUTER TURNED ON

It is useful to try things out on your computer as you read this book, so try to have your computer near you.

This is a manual, you do not read it in one session and come away a fully-blown computer programmer. You type things in when they are explained to you in the book, to experiment. In this way you can learn to program in easy steps and enjoy games as they come up.

THE KEYBOARD

At first glance the keyboard of the BBC micro looks a little intimidating, but as you learn more about programming, you'll be surprised how familiar it will become. In learning to program we must 'talk' to the computer. This is done through the keyboard. As you type the computer tells you what you have done by displaying it on the screen.

There are two kinds of keys on the keyboard: keys that do things and keys which simply put numbers and letters into the computer. There are also three red lights at the bottom left of the keyboard. The keys which simply put numbers and letters into the computer we shall call 'character keys'.

CHARACTER KEYS

To get you used to the arrangement of the keys on the computer just type randomly. It is quite similar to the typewriter arrangement, but has a few extra keys and the punctuation is set out differently.

If you can touch-type you must watch out for two things: do not use a capital 'O' for the number zero or use a lower case 'l' to input the number one.

If you have just turned the computer on, the lettering on the screen is likely to be in upper case. Again if you are used to typing you would expect to press the carriage- return key to go to the next line, but on computers the text automatically goes on to the next line. If you press RETURN (the computer carriage return) you get a display something like this:

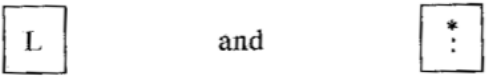
```
>JRFGBCUYT;RTTV;YUTYVBHGC

Mistake
>
```

Do not worry about anything that appears on the screen at the moment. Get used to the character keys so you can find them quickly and easily.

ACTION KEYS

THE SHIFT KEYS. So far you have been typing capital letters and numbers, but you have probably noticed the SHIFT, SHIFT LOCK and CAPS LOCK keys. These keys are related to two of the lights at the bottom left, the 'Shift Lock' and 'Caps Lock' lights. When you turn the computer on it always

assumes the CAPS LOCK key has been pressed, and to show that this has happened the 'Caps Lock' light will glow. To help you understand the way the shift keys work we will show you how they affect the character keys which we shall use as our examples:



In 'caps lock' mode when the 'caps lock' light is lit these keys give you 'L' and ':' respectively. In this mode the letter keys produce upper case letters though you still get the symbols on the lower parts of the punctuation keys. This is the mode the computer starts in because it is the most useful for typing in programs.

If you wish to get the symbols on the top part of the punctuation keys (whether you're in 'caps lock' mode or not), as for instance '*' in this case, you press SHIFT and while holding down SHIFT, press the key with the symbol you want.

If you press the CAPS LOCK key the 'caps lock' light goes out and the computer is in 'lower case' mode giving 'l' and ':' from our example keys. This mode simply gives you the symbols on the lower parts of the keys, as well as the lower-case letters.

If you now press the SHIFT LOCK key (the 'shift lock' light lights up) as on typewriters, all the character keys produce upper case letters and the upper symbols on the top half of the keys will appear on the screen. Using our example keys we would get 'L' and '*' respectively.

OTHER KEYS

CTRL (short for 'control') is a special kind of shift key which, when used in conjunction with the letter keys, changes them into action keys which 'control' the computer (as is the case with the other action keys). For example if you hold down CTRL and the letter 'G' that tells the computer to produce a

'beep' from its loudspeaker. Try it. It may not seem too useful at this point, but it could help wake an exhausted programmer! In programming this is called CONTROL G.

ESCAPE lets you 'escape' from a program, in other words, the program stops.

TAB is useful for word-processing programs — it immediately makes a space appear on the screen.

DELETE erases the character you have just typed (unless you have produced a space by pressing the TAB key). You can tell what you are about to erase by looking for the 'cursor': this is a flashing horizontal line as wide as a character which tells you where the next character you are about to type will appear. When you delete a character you delete the character to the left of the cursor.

BREAK acts as if you were turning the computer off and then back on very fast — it is there so you can stop the computer no matter what it is doing, and make it forget anything in its memory.

COPY, → , ← , ↑ and ↓ are called the 'editing keys'. These keys are used to change things that are on the screen. (See chapter three).

THE FUNCTION KEYS, i.e., the red keys f0 to f9 can be programmed to input a series of characters which you use a great deal — if you wanted to say 'BBC computer' many times in a program, instead of typing it out every time, one of the red keys could be programmed to input those letters at one depression. You will find an example of this in chapter five.

When communicating with the computer, we use commands in the form of keywords taken from ordinary English, which give you an idea of what they do. For instance, PRINT prints things on the screen or on a printer. SOUND makes the computer

output sound. There are many keywords to use in your BBC BASIC programs. When you are programming, the computer saves your typing by allowing you to use abbreviations of the keywords, the most frequently used words having the shortest abbreviations. PRINT for example can be input as P. and the less often used SOUND is shortened to SO. (full stops after these shortened keywords). (See your BBC manual for an alphabetical list of keyword abbreviations.)

As you have been typing you have seen that every time you press RETURN a 'greater than' symbol appears to the left of the screen with the cursor flashing next to it — this is the 'prompt', the computer is waiting for input

```
>_
```

That takes care of our introduction to the keyboard. You'll find that it is a lot simpler when you're programming the computer to know which mode to choose, and which one you're in, than might be apparent from this explanation. Stick with it, and you'll see.

CHAPTER TWO
PUTTING THINGS ON THE SCREEN

We pointed out in chapter one, when talking about the keyboard, that the action keys were the most important ones on the keyboard. We're now going to introduce another important action key, RETURN. Whenever you're typing, the computer will wait until you press RETURN before acting on the material you've entered. If you have written a program line, the computer will ignore that line, and it will sit at the bottom of the screen, with the cursor waiting patiently beside it, until you press RETURN.

Type this in: >PRINT 2

The same goes for the PRINT 2 you now have on the screen. Until you press RETURN the computer will do nothing about it. Press RETURN now and you should see the number 2 appear. This is how PRINT works. It takes the information which follows the command PRINT, with a few exceptions which we'll learn about in a moment, and PRINTs this on the screen, which after all is exactly what you'd expect it to do.

But your computer is more clever than that. If the word PRINT is followed by a sum, it will work it out before printing, and give

you the result of that sum. Try it now. Enter the following line, then press RETURN:

```
>PRINT 5+3
```

You should see the figure 8 appear. The computer added 5 and 3 together, as instructed by the plus (+) sign, then printed the result on the screen. It can do subtraction, as well (clever inventions, these computers). Type this in, and press RETURN to see subtraction (and PRINT) at work:

```
PRINT 7-2
```

Now, the computer can — of course — do a wide range of mathematical tasks, many of them far more sophisticated than simple addition and subtraction. But, there is a slight hitch. When it comes to multiplication, the computer does not use the × sign which you probably used at school. Instead, it uses an asterisk (*), and for division, instead of ÷ the computer uses a slash sign (/).

DOING MORE THAN ONE THING AT ONCE

The computer is not limited to a single operation in a PRINT statement. You can combine as many as you like. Try the next one, which combines a multiplication and division. Type it in, then press RETURN to see the computer evaluate it:

```
>PRINT 5*3/2
```

This seems pretty simple. Just type PRINT, then type in the material you want the computer to PRINT, and that's all there is to it. But, it is not as simple as that! Try the next one and see what happens:

```
>PRINT Testing
```

REPORT MESSAGES

That doesn't look too good. Instead of the word *testing* we've got a message from the computer which reads as follows:

```
No such variable
>
```

This is a computer report. It tells you when you've made a mistake. A complete list of the report codes is included in your manual. We won't try to explain the meaning of this one here, because variables are not on the curriculum for this chapter, but it means simply that the computer thought you wanted it to print a number, which had the name *testing*. Foolish machine. Computers may be very, very clever machines, but they need to be led by the hand, like a child and told exactly what you want them to do. Give them the right instructions, and they will carry them out tirelessly, and perfectly, without an error. But give them incorrect instructions, or — even worse — confuse them, and they give up in despair, or do something quite alien to your intentions.

STRINGS

If you want the computer to print the word testing you must put quote, or speech marks around the words, like this:

```
>
>PRINT "TESTING"
```

This time when you press RETURN, the word TESTING will appear. This is worth remembering. When you want the computer to print out some words, or a combination of words, symbols, spaces and numbers, you need to put quote marks around the material you want it to print. Information held in this way between quote marks is called a most peculiar name in computer circles. The jargon for information enclosed in quote marks is a *string*. So, in our example above, the word testing, when enclosed by quote marks, is a string.

OUR FIRST PROGRAM

Type the following into your computer. Notice that each line starts with a number. Type this number into the computer, then PRINT, get the opening quote marks from the 2 key (using SHIFT) then type in the words which follow on the same line, then get the closing quote marks from the 2 key again. Then, press RETURN. Instead of the first line appearing on the screen, the line is entered into the computer. To see that the computer has remembered the line type LIST followed by RETURN. This is the first line of your first program.

```
10 PRINT "Jack and Jill"
20 PRINT "went up the hill"
30 PRINT "To fetch a pail"
40 PRINT "of water."
```

Type in the next line (the one starting with 2Ø) and press RETURN once you have it all in place.

Look carefully at the material you've typed in. The most common mistake in this sort of program line is to leave the final quote mark off the end of the line. Check and recheck the line until it is exactly the same as the line in the listing above.

Lines do not have to be entered in strict numerical order. If you do miss a line out just type the line with its number press RETURN and the computer will put the line in its correct place in the program.

To see this in action, type the following line, press RETURN.

```
25 REM A line in the middle...
```

Now, the program should look like this when you LIST it.

```
10 PRINT "Jack and Jill"
20 PRINT "went up the hill"
```

```
25 REM A line in the middle...
30 PRINT "To fetch a pail"
40 PRINT "of water."
```

As you can see, the line numbered 25 has moved itself into the right position in the program, between lines 20 and 30. Now, RUN the program.

MAKING REMARKS

You should find that the new line, line 25, has not made any difference at all to the running of the program. Why not? Why did the computer decide to ignore line 25? The word REM stands for *remark* and is used within programs when we want to include information for a human being reading the program listing. You'll find REM statements scattered throughout the programs in this book. In each and every case the computer ignores the REM statements. They are there only for your convenience, for the convenience of the programmer, or of someone else reading the program.

Often you'll use REM statements at the beginning of the program, like this one:

```
5 REM Jack and Jill poem
```

You may wonder why this would be necessary. After all, it is pretty obvious that the computer is holding the 'Jack and Jill poem', even without the line 5 REM statement. You are right. In this case there is little point in adding a title REM statement to this program. But have a look at some of the more complicated programs a little further on in the book. Without REM statements you'd have a pretty difficult time trying to work out what the program was supposed to do.

REM statements are often scattered throughout programs. There they serve to remind the programmer what each section is supposed to do. Once you've been programming a while, you'll

be amazed at how many programs you'll collect in listing form which — when you go back to them in a month or so — will seem totally obscure. You won't have a clue how the program works, or even more important, what on earth it is, or what it is supposed to do. This is where you will find REM statements invaluable.

It is worth getting into good habits early as a programmer. So, we suggest you start right now adding REM statements to programs. If you come across programs, or program fragments in this book, which you want to keep, and which do not have REM statements, get into the habit of using REM statements by adding them to these programs. And make sure you use them in your original programs.

BACK TO PRINT

Let's return to the subject of the PRINT command. Empty your computer's memory by typing NEW. You can check that the program is no longer in the computer by trying to LIST it — there should be nothing there.

MORE PRINT STATEMENTS

Type the following program into your computer and then run it:

```
10 PRINT 1,2
15 PRINT
20 PRINT 1;2;3
25 PRINT
30 PRINT "Auntie Beeb"
35 PRINT
40 PRINT "23+34=";23+34
45 PRINT
50 PRINT 2*3
55 PRINT
60 PRINT 3^5
65 PRINT
70 PRINT "The Answer is ";23+5-7/6
```

you should get something like this on the screen:

```
1             2

123

Auntie Beeb

23+34=57

6
243

The Answer is 26.8333333
```

Now there is a lot we can learn from this program.

Firstly, as in the 'Jack an Jill' program, the computer executes a program line by line, starting at the lowest numbered one and proceeding through the line numbers in order until it runs out of numbers, then it stops. (You'll discover that this orderly progression of line numbers does not always apply, as there are ways of making the computer execute parts of a program out of strict numerical order, but for the time being, it is best to assume that the program will be executed in order).

Look first to line 10 of your program. You can see that there is a comma between the 1 and the 2. This has the effect of getting the computer to print the first number (1) nine characters in from the left hand side of the screen, and the second number (2) ten characters on from that, at about the middle. When you use a comma like this to divide the things which follow a PRINT statement (but *not* when the comma is in a string, that is, between quote marks), it divided the screen into columns of ten, printing that which follows the comma starting at the next available column of the screen. If the line had read PRINT 1,2,3,4,5 it would have printed the 1 at the start of the first line, the 2 ten characters on and onto the 5 *underneath* the one, because this was the 'next available ten characters'. If you're not too clear what we mean by this, try it yourself. Numbers are printed to the right of the column, while strings are printed to the left of each of the ten character columns.

The second line of the program (line 15) is just the word PRINT with nothing following it. This has the effect, as you can see in the printout (and on your television screen), of putting a blank line between those lines which do include material after the word PRINT. (The same comment applies to lines 25, 35, 45, 55 and 65).

Line 2Ø has three numbers (1, 2 and 3) separated not by commas (as in line 1Ø) but by semicolons (found on the O key). Instead of separating the output of the numbers as the comma did, you'll see that it causes them to be printed hard up against each other, as in the second line of the program. You use the semicolon (;) when you want some printed material to follow other printed material without a break. But as the numbers did not begin with a semicolon they started at the end of the first 1Ø characters.

Line 3Ø has the words Auntie Beeb and this is a ..... If you mentally said *string* when you came to those dots, then you're learning well. The word is a string, in computer terms, because it is enclosed within quote marks.

Line 4Ø is rather interesting. For the first time we have included numbers and a symbol (=) within a string. As you can see, the computer prints *exactly* what is within the quote marks, but works out the reslt of the calculation for the material outside the quote marks, giving — in this case — the result of adding 23 to 34. Try to remember that the computer considers everything within quote marks as words, even if it is made up from numbers, symbols, or even just spaces, or any combination of them, while it counts everything that is *not* within quote marks in a PRINT statement as a number. This is why it got so upset earlier when we told it to print *testing* without putting the word in quote marks. It looked for a number which was called *testing*, and because it could not find one (as we had not told the computer to let testing equal some numerical value), it refused to co-operate.

So line 4Ø treats the first part, within quote marks, as a string, and the second part, outside quote marks, as numerical information which it processed.

In line 5Ø we see the asterisk (*) used to represent multiplication and the computer quite reasonably works out what 2 times 3 is and prints the answer 6. In line 6Ø we come across a new, and strange sign, a little upward arrow. This means 'raise to the power' so line 6Ø means PRINT 3^5. Now, it is pretty difficult for a computer to print a number half way up the mast of another number, so we use the upward arrow to remind you (by pointing upward) that it really means 'print the second number up in the air'. You probably know that 7^2 is read as 'seven squared'. It means to raise seven to the second power. In a similar way, 3^5 means raise three to the fifth power which is exactly what the computer does in line 6Ø.

The final line of this program combines a string ('the answer is') with numerical information (23 + 5 – 7/6). You can see that, as expected, the computer works out the sum before printing the answer, and prints the string exactly as it is. Look closely at the end of the string. You'll see there is a space there. After the closing quote is a semicolon (;) which, as we learned in line 2Ø, joins various elements of a PRINT statement together. This semicolon means that the result of the calculation is printed up next to the end of the string. However, if there were no space within the quote marks, the result of printing line 7Ø would be:

```
The answer is26.833333
```

As you can see, this is by no means as clear as the original version.

That brings us to the end of the second chapter of the book. We're sure you'll be pleased at how much you've learned so far, and are looking forward to continuing your learning. But now you've earned a break. So take that break, and then come back to the book to tackle the third chapter.

CHAPTER THREE
RINGING THE CHANGES

It's all very well getting things onto the computer's screen as we learnt to do in the last chapter, but from time to time you'll discover we need to be able to get printed material *off* the screen during a program, to make way for more PRINT statements. We do this with a command called CLS, for CLear the Screen.

CLEAR THE SCREEN

Enter the following program into your computer and run it. The dollar sign after the A in line 20 comes from the 4 key. You get the dollar sign by holding down the shift while pressing the four key.

```
10 PRINT "TESTING"
20 INPUT A$
30 CLS
```

When you run the program, you'll see the word TESTING appear at the top of the screen, more or less as you'd expect:

```
TESTING
```

However, below that you'll see a question mark within a flashing cursor next to it:

This is an *input prompt*. An input prompt, which appears in a program when the computer hits the word INPUT, means the computer is waiting for you to enter something else into the computer, or just to press RETURN. You'll recall that we spoke earlier about *strings* and about how they were anything which was enclosed in quote marks. The computer signals that it is talking about a string by using the dollar sign. So line 2Ø is waiting for a string *called* A$ to be input by you.

Anyway, when you respond to the input prompt by pressing ENTER, you'll see the screen clears, and *testing* disappears. Where did it go? We pointed out that the computer works through a program in line order. Firstly, with this program, it printed *testing* on the screen, then progressed to line 2Ø, where it waited for an input (or for you to press RETURN). Once you'd done this in line 2Ø, it moved along to line 3Ø where it found CLS and obeyed that instruction. The instruction was to *clear the screen*, so the computer did just that, and the screen went clear.

Run the program a few more times, until you've got a pretty good idea of what is happening, and you've followed through — in your mind — the sequence of steps the computer is executing.

DOING IT AUTOMATICALLY

Instead of waiting for you to press ENTER, you can write a program which clears the screen automatically, as our next example demonstrates. Enter this next program into your computer, type RUN, then RETURN, and sit back for the Amazing Flashing Word demonstration.

```
10 PRINT "AUTOTESTING"
20 FOR A=1 TO 1000
30 NEXT A
40 CLS
```

```
50 FOR A=1 TO 1000
60 NEXT A
70 RUN
```

Run this program, and you'll see the word AUTOTESTING alternately flashing off and on at the top of the screen. What is happening here? Let's look at the program, and go through it line by line. Firstly, as you know, line 10 prints AUTOTESTING at the top of the screen. Next, the computer comes to line 20, where it meets the word FOR. We'll be learning about FOR/NEXT loops (as they are called) in detail in a later chapter, but all you need to know here is that the computer uses FOR/NEXT loops for *counting*. In this program lines 20 and 30 (for FOR is in line 20, the NEXT in line 30) tell the computer to count from one to 1000, before moving on. As you can see, it does this counting pretty quickly.

So, it waits for a moment while counting from one to 1000. Then it comes to line 40, which is the command CLS, which tells the computer to clear the screen. It does this, and AUTOTESTING disappears from the screen. The computer then encounters, in lines 50 and 60, another FOR/NEXT loop, so waits a while as it counts form one to 1000 again. Continuing on in sequence, it comes to line 70, where it finds the word RUN. Now, as you know, you get the computer to execute a program by typing RUN, followed by RETURN. But RUN is a key word, so there is no reason why you cannot use it *within* a program, as we have in line 70 of this program. When the computer comes to line 70, it obeys the command, and RUNs the program again, so that it goes through the AUTOTESTING printing, counting to 1000, clearing the screen, counting to 1000 again, and then hitting RUN so that the whole dance starts over. This is an endless loop so to stop the program press the ESCAPE key.

CHANGING PROGRAM LINES EASILY

Your computer is provided with an EDIT function which makes it very simple to change the content of lines within a program. NEW the current program, then enter the

following program into your computer. *Do not press RUN*. We want to explain how to edit the program:

```
10 REM An Edit Test
20 PRINT "Test Again"
30 PRINT "And Again"
```

If you want to put a line in the middle, you should remember that if you type

```
>25 PRINT "An Errorr
```

and the program is listed, it will appear in between lines 20 and 30. But if you look at the line carefully, you will see that if you were to RUN it the line would generate the error:

```
Missing " at line 25
```

So you should correct it, but, instead of rewriting the line you can use the arrow keys to create the editing cursor. Lightly tap the up-arrow key and the underlining cursor moves up one character and where it was a white block appears. The block is the 'printing cursor' and the flashing line is the copying cursor. When you press RETURN the cursors come together. LIST the program and move the 'copying cursor' to the beginning of line 25. Then press and hold down COPY. The copying cursor copies the line and the printing cursor will print the line as the cursor copies it.

In this example the copying cursor is about to copy the space between "An" and "Error" so the printing cursor is about to print the copied space:

```
>25 PRINT "An Errorr
>25 PRINT "An■
```

Here is our line completely copied:

```
>25 PRINT "An Errorr■
```

Then after the deletion of the "r" and the addition of a " (quote)

```
125 PRINT "an Error"
```

To finish editing, press RETURN entering the new line into the computer.

CHAPTER FOUR
DESCENT INTO CHAOS

It's time now to start developing some real programs. You'll notice that from this point on in the book there are some rather lengthy programs. Many of them will contain words from the BASIC programming language which have *not* been explained. This is because, as programs become more complex (and far more satisfying to run) it becomes more and more difficult to keep programming words which have not been explained out of the program. However, it is not a major problem.

We are working methodically through the commands available on the computer. When you come across a word in a program which seems unfamiliar, just enter it into the program. You'll find that you'll soon start picking up the meaning of words which have not been explained, as you see how they are used within a program. So, if you find a new word, don't worry. The program will work perfectly without you knowing what the word is, and investigating the listing after you've seen the program running is likely to allow you to work out what it means anyway.

RANDOM EVENTS

In the world of nature, as opposed to the manufactured world of man, randomness appears to be at the heart of many events. The number of birds visible in the sky at any one time, the fact

that it rained yesterday and may rain again today, the number of trees growing on one side of a particular mountain, all appear to be somewhat random. Of course, we can predict with some degree of certainty whether or not it will rain, but the success of our predictions appears to be somewhat random as well.

When you toss a coin in the air, whether it lands heads or tails depends on chance. The same holds true when you throw a six-sided die down onto the table. Whether it lands with one, the three or the six showing depends on random factors.

Your computer has the ability to generate random numbers which are very useful for getting the computer to imitate the random events of the real world. The computer uses the keyword RND which stands for RaNDom.

GENERATING RANDOM NUMBERS

We'll start by using RND just as it is to create some random numbers. Enter the following program, and run it for a while:

```
10 PRINT RND(1)
20 DUMMY=GET
30 GOTO 10
```

When you do, you'll see a list of numbers like these appear on the screen:

```
 0.94140622
 3.85285891E-3
 1.93786994E-3
 0.49706936
 0.99974149
 0.18751481
 0.99614711
```

```
6.05316719E-2
0.502936419
1.34153934E-2
0.308607607
0.753714576
0.876709256
0.500855211
2.45020725E-4
0.187500448
2.0065421 7E-3
0.559584621
```

As you can see, RND(1) generates numbers randomly between zero and one. If you leave it running, it will go on and on apparently forever, writing up new random numbers on the screen.

Now random numbers between zero and one are of limited interest if we want to generate the numbers and get them to stand for something else. For example, if we could generate 1's and 2's randomly, we could call the 1's *heads* and the 2's *tails* and use the computer as a kind of electronic 'coin'. If we could get it to produce whole numbers between one and six, we could use the computer as an imitation six-sided die.

Fortunately, there is a way to do this.

```
10 PRINT ;RND(6);" ";
20 DUMMY=GET
30 GOTO 10
```

When you run this program, you'll get a series of numbers (chosen at random between 1 and 6) like these:

```
4 3 1 6 6 4 6 2 2 1 3 1 2 3 3 5 1 2 5 4
5 5 6 3 6 5 6 5 5 6 2 2 5 6 1 5 6 5 2 3
4 4 6 1 6 4 4 5 1 6 3 4 2 2 5 2 4 2 4 4
2 2 1 1 5 2 3 5 5 1 5 5 2 5 2 5 4 5 1 1
6 2 6 3 2 1 2 1 6 4 4 2 5 6 1 5 5 3 1 3
2 4 6 4 6 2 6 2 2 1 2 2 5 1 2 2 6 2 2 2
2 3 3 1 5 4 1 2 6 4 2 1 5 2 5 5 3 5 3 5
3 2 3 1 6 2 1 1 5 2 3 1 4 1 6 5 3 1 6 6
2 5 5 3 6 6 2 5 1 2 5 3 2 3 6 4 2 6 4 5
2 3 4 1 5 3 2 3 5 2 5 4 4 2 1 3 2 1 2 4
6 3 5 4 4 2
```

Even though we can create vast series of numbers between 1 and 6 with a program like this, it is not particularly interesting.

FAST FOOD CRAZINESS

Enter and run the next program, which makes an interesting use of the computer's ability to generate random numbers. As you can see, it creates a scene where you have turned up at a fast food outlet, desperate for something to eat, and you've decided to let random number generator pick your food for you:

```
    10 REM FAST FOOD
    30 A=RND(4)
    40 PRINT "YOU'VE ORDERED"
    50 IF A=1 THEN PRINT "A HAMBURGER WIT
H ALL THE TRIMMINGS"
    60 IF A=2 THEN PRINT "A LARGE PORTION
 OF STEAMING FRENCH FRIES"
    70 IF A=3 THEN PRINT "A HUGE PLATEFUL
 OF FRESHLY COOKED RIBS"
    80 IF A=4 THEN PRINT "A PAIR OF 100%
MEAT HOT DOGS WITH SAUCE"
    90 PRINT
    95 DUMMY=GET
   100 GOTO 30
```

When you run this, you'll get something like this list of food on the screen:

```
YOU'VE ORDERED
A PAIR OF 100% MEAT HOT DOGS WITH SAUCE

YOU'VE ORDERED
A PAIR OF 100% MEAT HOT DOGS WITH SAUCE

YOU'VE ORDERED
A HAMBURGER WITH ALL THE TRIMMINGS

YOU'VE ORDERED
A HAMBURGER WITH ALL THE TRIMMINGS

YOU'VE ORDERED
A LARGE PORTION OF STEAMING FRENCH FRIES

YOU'VE ORDERED
A PAIR OF 100% MEAT HOT DOGS WITH SAUCE

YOU'VE ORDERED
A HAMBURGER WITH ALL THE TRIMMINGS

YOU'VE ORDERED
A PAIR OF 100% MEAT HOT DOGS WITH SAUCE

YOU'VE ORDERED
A HAMBURGER WITH ALL THE TRIMMINGS

YOU'VE ORDERED
A HUGE PLATEFUL OF FRESHLY COOKED RIBS
```

(Press any key to see what you have 'ordered').

Notice how the program sets the letter A to the value of the random number in line 3Ø. In this case, the letter A is *standing for* a number. It is called a *variable*, or, because it stands for a number (as opposed to standing for a word, or a string) it is called a *numeric variable*. In computer jargon, we say that, (in line 3Ø) the computer has *assigned* the value of the random number to the variable A. And, as you can see in lines 5Ø, 6Ø, 7Ø and 8Ø, the value assigned to A determines which food order you place. Line 95 makes the computer wait until you press a key (any of the alphanumeric keys) before the computer continues. Otherwise the computer would go on so fast that you would not be able to read it! Delete the line to see... Read this over if it seems incomprehensible the first time.

CHAPTER FIVE
ROUND AND ROUND WE GO

In this chapter we'll be introducing a very useful part of your programming vocabularly — FOR/NEXT loops. You'll recall that we mentioned FOR/NEXT loops when demonstrating the use of CLS to clear the screen. There, a loop was used to add a delay after the word was printed on the screen before the screen was cleared.

A FOR/NEXT loop is pretty simple. It has the form of two lines in the program, the first like this:

```
FOR A=1 TO 20
```

And the second as follows:

```
NEXT A
```

The *control variable*, the letter after FOR and after NEXT must be the same.

As a FOR/NEXT loop runs, the computer counts from the first number up to the second, as these two examples will show:

```
10 FOR A=1 TO 20
20 PRINT A;
30 NEXT A
```

When you run it, this appears on the screen:

```
 1   2   3   4
 5   6   7   8
 9  10  11  12
13  14  15  16
17  18  19  20
```

Here's another version:

```
10 FOR A=765 TO 781
20 PRINT A;
30 NEXT A
```

And this is the result of running it:

```
765  766  767  768
769  770  771  772
773  774  775  776
777  778  779  780
781>
```

STEPPING OUT

In the two previous examples, the computer has counted up in ones, but there is no reason why it should do so. The word STEP can be used *after* the FOR part of the first line as follows:

```
10 FOR A=10 TO 100 STEP 10
20 PRINT A;
30 NEXT A
```

When you run this program, you'll discover it counts (probably as you expected) in steps of ten, producing this result:

```
 10  20  30  40
 50  60  70  80
 90 100>
```

STEPPING DOWN

The STEP does not have to be positive. Your computer is just as happy counting backwards, using a negative step size:

```
10 FOR A=100 TO 10 STEP -10
20 PRINT A;
30 NEXT A
```

This is what the program output looks like:

```
 100          90          80          70
  60          50          40          30
  20          10>
```

MAKING A NEST

It is possible to place one or more FOR/NEXT loops *within* each other. This is called *nesting loops*. In the next example, the B loop is *nested* inside the A loop:

```
10 FOR A=1 TO 3
20 FOR B=1 TO 2
30 PRINT A;" TIMES ";B;" IS ";A*B
40 NEXT B
50 NEXT A
```

The nested programs produce this result:

```
 1 TIMES 1 IS 1
 1 TIMES 2 IS 2
 2 TIMES 1 IS 2
 2 TIMES 2 IS 4
 3 TIMES 1 IS 3
 3 TIMES 2 IS 6
```

You must be very careful to ensure that the *first* loop started is the *last* loop finished. That is, if FOR A . . . was the first loop you mentioned in the program, the last NEXT must be NEXT A.

Here's what can happen if you get them out of order (the B loop ends *after*, rather than before, the A loop):

```
10 FOR A=1 TO 3
20 FOR B=1 TO 2
30 PRINT A;" TIMES "$B$" IS "$A*B
40 NEXT A
50 NEXT B

                1 TIMES 1 IS 1
                2 TIMES 1 IS 2
                3 TIMES 1 IS 3

   No FOR at line 50
   >
```

MULTIPLICATION TABLES

You can use nested loops to get the computer to print out the multiplication tables, from one times one right up to twelve times twelve, like this:

```
10 FOR A=1 TO 12
20 FOR B=1 TO 12
30 PRINT A;" TIMES "$B$" IS "$A*B
40 NEXT B
50 NEXT A
```

Here's part of the output:

```
                5 TIMES 1 IS 5
                5 TIMES 2 IS 10
                5 TIMES 3 IS 15
```

```
5 TIMES 4 IS 20
5 TIMES 5 IS 25
5 TIMES 6 IS 30
5 TIMES 7 IS 35
5 TIMES 8 IS 40
5 TIMES 9 IS 45
5 TIMES 10 IS 50
5 TIMES 11 IS 55
5 TIMES 12 IS 60
6 TIMES 1 IS 6
6 TIMES 2 IS 12
6 TIMES 3 IS 18
6 TIMES 4 IS 24
6 TIMES 5 IS 30
6 TIMES 6 IS 36
6 TIMES 7 IS 42
6 TIMES 8 IS 48
```

There is no reason why both loops should be travelling in the same direction (that is, why both should be counting upwards), as this variation on the Times Table program demonstrates:

```
10 FOR A=1 TO 12
20 FOR B=12 TO 1 STEP -1
30 PRINT A;" TIMES ";B;" IS ";A*B
40 NEXT B
50 NEXT A
```

Here's part of the output of that program:

```
1 TIMES 12 IS 12
1 TIMES 11 IS 11
1 TIMES 10 IS 10
1 TIMES 9 IS 9
1 TIMES 8 IS 8
```

```
1 TIMES 7 IS 7
1 TIMES 6 IS 6
1 TIMES 5 IS 5
1 TIMES 4 IS 4
1 TIMES 3 IS 3
1 TIMES 2 IS 2
1 TIMES 1 IS 1
2 TIMES 12 IS 24
2 TIMES 11 IS 22
2 TIMES 10 IS 20
2 TIMES 9 IS 18
```

CRACKING THE CODE

It's time for our first *real* program. In this program (which uses several FOR/NEXT loops) CODEBREAKER, the computer thinks of a four-digit number (like 5462) and you have eight guesses in which to work out what the code is. In this program, written by Adam Bennett and Tim Summers, you not only have to work out the four numbers the computer has chosen, but also determine the order they are in.

After each guess, the computer will tell you how near you are to the final solution. A 'white' is the right digit in the wrong place in the code, a 'black' is the right number in the right place. You'll see that you are aiming to get four blacks. When you have, you have cracked the code. Digits may be repeated within the four-number code. Enter the program, and play a few rounds against the computer. Then, return to the book for a discussion on it, which will highlight the role played by the FOR/NEXT loops.

```
  5 CLS
 10 PRINT "**********************************
**********"
 20 PRINT
 30 PRINT
```

```
   40 PRINT "     CODEBREAKER... BY AJB,T
S & AB..."
   45 PRINT "     ========================
=========="
   50 PRINT
   60 PRINT "          When you are told t
o do"
   70 PRINT "          so,enter a 4-digit n
umber"
   80 PRINT "            and then press ENT
ER."
   90 PRINT "          Digits may be repea
ted"
  100 PRINT "         You have 8 goes to b
reak"
  110 PRINT "            the difficult cod
e."
  120 PRINT "**************************
************"
  130 DUMMY=GET
  140 CLS
  150 DIM B(4)
  160 DIM D(4)
  170 H=0
  180 FOR A=1 TO 4
  190 B(A)=RND(9)
  200 NEXT A
  210 FOR C=1 TO 8
  220 PRINT TAB(3,0)"Enter Guess Number
";C;" ";
  230 INPUT X
  240 IF X>9999 THEN GOTO 220
  250 PRINT
  270 PRINT TAB(3,C+1)X;" ";
  280 P=INT(X/1000)
```

```
290 Q=INT((X-1000*P)/100)
300 R=INT((X-1000*P-100*Q)/10)
310 S=(X-1000*P-100*Q-10*R)
320 D(1)=P
330 D(2)=Q
340 D(3)=R
350 D(4)=S
360 FOR E=1 TO 4
370 IF D(E)<>B(E) THEN GOTO 420
380 PRINT " BLACK ";
390 B(E)=B(E)+10
400 D(E)=D(E)+20
410 H=H+1
420 NEXT E
430 IF H=4 THEN GOTO 630
440 FOR F=1 TO 4
450 D=D(F)
460 FOR G=1 TO 4
470 IF D<>B(G) THEN GOTO 510
480 PRINT " WHITE ";
490 B(G)=B(G)+10
500 GOTO 520
510 NEXT G
520 NEXT F
530 FOR G=1 TO 4
540 IF B(G)<10 THEN GOTO 560
550 B(G)=B(G)-10
560 NEXT G
570 H=0
580 PRINT
590 NEXT C
600 PRINT
602 PRINT
604 PRINT "    You didn't get it.....
"
```

```
  610 PRINT "   The answer was ";B(1);B(2
);B(3);B(4)
  620 END
  630 PRINT
  632 PRINT
  634 PRINT "    Well done, Codebreaker..
."
  640 PRINT
  642 PRINT "   You got the answer"
  650 PRINT "   In Just ";C;" goes"
>
```

Here's what the screen looks like after one round:

```
Enter Guess Number 7 ?6691

6549   BLACK   WHITE
8888
9876   WHITE   WHITE
6654   BLACK   BLACK
3091   BLACK   BLACK
3054
6691   BLACK   BLACK   BLACK   B

 Well done Codebreaker...

You got the answer
In Just 7 goes
>_
```

We'll now go through the program line by line, a practice we'll be following for several of the programs in this book. If you don't want to read the detailed explanation now (and there may be parts of it which are a bit difficult to understand at your present stage), by all means skip over the explanation and then come back to it later when you know a little more.

Lines 10 and 120 print a number of asterisks to rule off the title

and instructions, with blank lines printed by lines 2Ø, 3Ø, 5Ø and 25Ø. After the title is printed by line 4Ø, it is 'underlined'.

Line 14Ø waits for you to press any key when you have read the instructions. When you've read them press any key. Two *arrays* are dimensioned in lines 15Ø and 16Ø. We get to arrays in a later chapter. For now, all you need to know is that by saying DIM B(4) you tell the computer you want to create a *list* of objects, called B, in which the first item can be referred to as B(1), the second as B(2), and so on. These two arrays are used for storing the numbers picked by the computer, and for your guess.

H is a *numeric variable* (we've mentioned numeric variables before) which is set equal to zero in the next line. In line 41Ø, one is added to the value of H each time a black is found, so that if H equals four, the computer knows all the digits have been guessed and goes to the routine from line 63Ø to print up the congratulations.

The lines fromd 18Ø to 2ØØ work out the number which you will have to try and guess. Line 19Ø uses the RND function we've talked about to get four random numbers between zero and nine, and stores one each in the elements of the B array. Note that the first FOR/NEXT loop of our program appears here. The A in line 18Ø equals one the first time the loop is passed through, two the second time, and so on, so that the A in line 19Ø changes as well.

Our next FOR/NEXT loop, which uses C, starts in the next line. It counts from one to eight, to give you eight guesses. Line 23Ø accepts your guess, using the words 'Enter guess number 1' as an input prompt. The numeric variable X is set equal to your guess, and line 24Ø checks to make sure you have not entered a five-digit number. If you have (if X is greater than 9999) then the program goes back to line 23Ø to accept another guess.

The next section of the program, right through to line 58Ø, works out how well you've done, using a number of

FOR/NEXT loops (360 to 420, 440 to 520, 460 to 510 and 530 to 560). Line 590 sends the program back to the line after the original FOR C= . . . to go through the loop again. If the C loop has been run through eight times, then the program does not go back to line 220, but 'falls through' line 590 to tell you that you have not guessed the code in time, and to tell you what it is. Line 610 prints out the code.

If you do manage to guess it, so that H equals four in line 430, then the program jumps to line 630 to print out the congratulatory message. Every time you run the program, the program ends and if you want to have another game, you need to type RUN and press RETURN. To save you this typing you can make one of the function keys print RUN and input RETURN in this way: *KEY 0 RUN|M. The "|" comes from the ¦ key. You can program the other keys to input useful strings like LIST|M, MODE 7|M. The |M inputs RETURN.

PACKING 'EM IN

You can save quite a bit of space in the program (although it doesn't always make it easier to work out what is going on within the program, or to pick up errors) by using *multi-statement lines*. With a multi-statement line, you put more than one statement to each line number. Each statement on the same line must be separated by a colon (:).

Here is a condensed version of CODEBREAKER produced by putting more than one statement on a line in places, and by using apostrophes (') in lines 10, 20, 50 and 90. Compare the two listings, and see how a little bit of space can be saved with multi-statement lines.

```
10 CLS:PRINT STRING$(39,"*")"" CO
DEBREAKER... BY AJB,TS & AG..."" "STR
ING$(31,"=")"" When you are told
to do"" so,enter a 4-digit number
"" and then press ENTER.""
Digits may be repeated""
20 PRINT " You have 8 goes to b
reak"" the difficult code.""ST
RING$(39,"*"):DUMMY=GET:CLS:DIM B(4),D(4
):H=0:FOR A=1 TO 4
30 B(A)=RND(9):NEXT:FOR C=1 TO 8:PRIN
T TAB(3,0)"Enter Guess Number ";C;" ";:I
NPUTX:PRINT'TAB(3,C+1)X;" ";:D(1)=INT(X/
1000):D(2)=INT((X-1000*D(1))/100):D(3)=I
NT((X-1000*D(1)-100*D(2))/10):D(4)=((X-1
000*D(1))-100*D(2)-10*D(3))
40 FOR E=1 TO 4:IF D(E)<>B(E) THEN GO
TO 50:ELSE PRINT " BLACK ";:B(E)=B(E)+10
:D(E)=D(E)+20:H=H+1
50 NEXT E:IF H=4 THEN PRINT"" Wel1
done, Codebreaker..."""" You got the a
nswer"" In Just ";C;" goes":END
60 FOR F=1 TO 4:D=D(F):FOR G=1 TO 4:I
F D<>B(G) THEN GOTO 70:ELSEPRINT " WHITE
";:B(G)=B(G)+10:GOTO 80
70 NEXT
80 NEXT:FOR G=1 TO 4:IF B(G)<10 THEN
GOTO 90 ELSE B(G)=D(G)-10
90 NEXT:H=0:PRINT:NEXT:PRINT"" You
didn't get it.....""" The answer was
";B(1);B(2);B(3);B(4):END
```

CHAPTER SIX
CHANGING IN MID-STREAM

We pointed out at the beginning of the book that, in most situations, your computer follows through a program in line order, starting at the lowest line number and following through in order until the program reached the final line.

This is not always true. The GOTO command sends action through a program in whichever order you decide.

Enter the following program, and *before* you run it, see if you can work out what the result of running it will be:

```
10 GOTO 50
20 PRINT "This is 20"
30 GOTO 80
50 PRINT " This is 50"
60 GOTO 20
80 PRINT "  This is 80"
85 DUMMY=GET
90 GOTO 50
```

This rather pointless program sends the poor computer jumping all over the place, changing its position in the program

every time it comes to a GOTO command. Here's what you should see on your screen:

```
This is 20
  This is 80
 This is 50
This is 20
  This is 80
 This is 50
This is 20
  This is 80
 This is 50
This is 20
  This is 80
 This is 50
This is 20
  This is 80
```

The program starts at line 1Ø, and finding GOTO 5Ø there, moves on to line 5Ø to print out "THIS IS 5Ø". It then continues on to line 6Ø, where it finds the command "GOTO 2Ø". Without question, it zips back to line 2Ø to print out "THIS IS 2Ø" then goes to line 3Ø which directions it to line 8Ø. At line 8Ø, it finds the instruction to print out "THIS IS 8Ø" which it does. From there it gets to line 9Ø, and finds "GOTO 5Ø" which is just about where we began. . . . and starts all over again.

RESTRICTIVE PRACTICES

Using GOTO in this way is called *unconditional*. The command is not qualified in any way, so the computer always obeys it. Another group of words generally found together on the computer are IF, THEN and ELSE. IF something is true, THEN do something, ELSE do otherwise. IF you are hungry, THEN order a hamburger, ELSE go home. IF you want some candy, THEN behave, ELSE be bad. IF condition, THEN

action, ELSE alternate action. (The ELSE part need not be included every time.) The next program, which 'rolls a die' (using the random number generator) and then prints up the result of that die roll as a word, uses a number of IF/ THEN lines:

```
 10 REM DICE ROLLS
 20 GOTO 100
 30 PRINT "ONE"
 35 GOTO 100
 40 PRINT "TWO"
 45 GOTO 100
 50 PRINT "THREE"
 55 GOTO 100
 60 PRINT "FOUR"
 65 GOTO 100
 70 PRINT "FIVE"
 75 GOTO 100
 80 PRINT "SIX"
100 DUMMY=GET
110 A=RND(6)
120 IF A=1 THEN GOTO 30
130 IF A=2 THEN GOTO 40
140 IF A=3 THEN GOTO 50
150 IF A=4 THEN GOTO 60
160 IF A=5 THEN GOTO 70
170 IF A=6 THEN GOTO 80
```

This is what you'll see when you run the program:

```
ONE
TWO
TWO
TWO
SIX
FOUR
```

```
ONE
THREE
FOUR
TWO
ONE
SIX
ONE
FIVE
SIX
THREE
```

So we've looked at non-conditional, and conditional GOTO's to send action all about the place within a program.

ANOTHER WAY TO FLY

There is another way to redirect the computer during the course of a program by the use of *subroutines*. A subroutine is part of a program which is run twice or more during a program, and is more efficiently kept outside the main program than within it.

This example should make it clear. In this, the computer throws a die over and over again. The first time it is thrown the computer is throwing for itself. The second time it is thrown for you. After each pair of dice has been thrown, it will announce who is the winner (highest number wins). The program uses a subroutine to roll a die. Enter and run the program, then return to the book and I'll explain where the subroutine is within the program, and how it works:

```
10 FOR T=1 TO 1000:NEXT
20 FOR C=1 TO 2
30 GOSUB 100
40 IF C=1 THEN A=D
50 IF C=2 THEN B=D
60 NEXT C
70 IF A>B THEN PRINT "I WIN"
80 IF A<B THEN PRINT "YOU WIN"
```

```
  90 GOTO 10
 100 REM This is Sub-Routine
 110 D=RND(6)
 120 IF C=1 THEN PRINT "I Rolled a ";D
 130 IF C=2 THEN PRINT "You Rolled a
";D
 140 FOR T=1 TO 1000:NEXT
 150 RETURN
```

This is what you'll see when you run it:

```
I Rolled a 6
You Rolled a 3
I WIN
I Rolled a 6
You Rolled a 6
I Rolled a 1
You Rolled a 4
YOU WIN
I Rolled a 3
You Rolled a 1
I WIN
I Rolled a 1
You Rolled a 4
YOU WIN
```

The program pauses for a short while on line 10, and then enters the C FOR/NEXT loop. When it gets to line 30, which it does (of course) once each time through the C loop, the program is sent to the *subroutine* starting at line 100. The 'die is rolled' in line 110, and the numeric variable D is set equal to the result of the roll. The next two lines print out the result of the roll, using an IF/THEN to determine whether the computer should print "I ROLLED A ..." or "YOU ROLLED A ...". There is a slight pause in line 140, and then the computer comes to the

word RETURN. The word RETURN signals to the computer that it must *return* to the line *after* the one which sent it to the subroutine. In this program, the relevant line is 40, because line 30 sent the program to the subroutine. There, the IF/THENs in lines 40 and 50 determine whether the value of the roll (D) should be assigned to the variable A or to the B.

Line 60 ends the FOR/NEXT loop, and then lines 70 and 80 determine whether the computer has won (which it will have done if A is greater than B, a condition which is tested using the > sign in line 70) or whether the human has won (which will happen if A is less than B, a condition tested in line 80 by use of the 'less than' symbol, <). From here, the program goes back to line 10 where it starts again.

Study this example until you're pretty sure you know how subroutines work.

LET'S ROLL AGAIN

You may wonder if it is possible to change the earlier program, which changed the number rolled by the die into a word, using subroutines. The answer is 'yes', although the program with subroutines is not much shorter than the version using GOTO. Here's one way it could be done:

```
10 REM DICE ROLLS
20 GOTO 100
30 PRINT "ONE"
35 RETURN
40 PRINT "TWO"
45 RETURN
50 PRINT "THREE"
55 RETURN
60 PRINT "FOUR"
65 RETURN
70 PRINT "FIVE"
75 RETURN
```

```
 80 PRINT "SIX"
 85 RETURN
100 DUMMY=GET
110 A=RND(6)
120 IF A=1 THEN GOSUB 30
130 IF A=2 THEN GOSUB 40
140 IF A=3 THEN GOSUB 50
150 IF A=4 THEN GOSUB 60
160 IF A=5 THEN GOSUB 70
170 IF A=6 THEN GOSUB 80
180 GOTO 100
```

COMPUTED DESTINATIONS

Your computer is capable of accepting a number, an assigned variable (such as B when it knows what number B represents) or a combination of these as instructions on the line number it should go to. In the previous program, you'll see that the GO SUB destinations get larger in a regular way, and the number rolled by the die also increases in a regular way. We can use this information to tidy up the section from lines 11Ø to 17Ø in the previous program, so that it looks like this:

```
10 REM DICE ROLLS
20 GOTO 100
30 PRINT "ONE"
35 RETURN
40 PRINT "TWO"
45 RETURN
50 PRINT "THREE"
55 RETURN
60 PRINT "FOUR"
65 RETURN
70 PRINT "FIVE"
75 RETURN
80 PRINT "SIX"
```

```
 35 RETURN
100 DUMMY=GET
110 A=RND(6)
120 GOSUB 20+10*A
180 GOTO 100
```

Another way of condensing GOSUB statements is the use of ON. When you see the statement

ON A GOSUB 100,200,720,1000,1

the program will go to the subroutine at line 100 if A is 1. If A is 2 the program will go to the subroutine at 200, if it is 3, the program GOSUBs to 720. This is useful if the places to be GOSUBed to are not in a regular number sequence.

Here is the same program using ON. . . GOSUB

```
 10 REM DICE ROLLS
 20 RETURN
 30 PRINT "ONE"
 35 RETURN
 40 PRINT "TWO"
 45 RETURN
 50 PRINT "THREE"
 55 RETURN
 60 PRINT "FOUR"
 65 RETURN
 70 PRINT "FIVE"
 75 RETURN
 80 PRINT "SIX"
 85 RETURN
100 DUMMY=GET
110 A=RND(6)
120 ON A GOSUB 30,40,50,60,70,80
180 GOTO 100
```

If you run this, you'll see it performs in exactly the same way as the other versions.

CHAPTER SEVEN
GETTING INTO THE MUSIC BUSINESS

The sound command is a great way to add life to your programs. It is amazing what a little sound can do to enhance a program.

SOUND is always followed by four numbers. The first number is the channel which the sound is to be output from. Imagine there are three loudspeakers in the computers. Each of these speakers has a number (1, 2 and 3). In the SOUND statement the first number tells the computer which speaker to output from. The second number defines what volume is to be used, ranging from a Ø (silent) to −15 (loudest). The third number is the pitch and is a number ranging from Ø to 255: each of these numbers has a separate note, Ø being the lowest in pitch. They are similar to the musical scale in that the pitch rises in quarter semi-tones with middle C having a value of 53. The fourth and final number in the statement tells the computer the length of time the note is to be sounded in units of a twentieth of a second. Here is a program which demonstrates the pitch range of the SOUND statement:

```
10 REM SOUND 1
20 REM
40 channel=1
50 volume=-15
60 duration=1
```

```
70 FOR note=0 TO 255
80 SOUND channel,volume,note,duration
90 NEXT note
```

This program used only one 'loudspeaker' (or channel). If you use three channels at the same time you get a different effect as this program demonstrates:

```
10 REM SOUND 2
20 REM
50 volume=-15
60 duration=1
70 FOR note=0 TO 255
75 FOR channel=1 TO 3
80 SOUND channel,volume,note,duration
85 NEXT channel
90 NEXT note
```

If as in this next program the pitch is slightly different for each channel the sound has more depth and chords can be introduced.

```
10 REM SOUND 3
20 REM
50 volume=-15
60 duration=1
70 FOR note=0 TO 249
75 FOR channel=1 TO 3
80 SOUND channel,volume,note+channel*
3,duration
85 NEXT channel
90 NEXT note
```

The pitch values need not be similar of course. Try this siren-effect program:

```
 10 REM SOUND 4
 20 REM
 40 volume=-15
 50 duration=1
 60 FOR note=0 TO 249
 70 SOUND 1,volume,note,duration
 80 SOUND 2,volume,note+6,duration
 90 SOUND 3,volume,256-note,duration
100 NEXT note
```

There is yet another channel available for your use numbered Ø, but that does not produce notes — it produces noise. This next program demonstrates the range of sounds available from the 'noise channel'. Press any key. You will hear eight different kinds of sound. Pitch Ø, 1 and 2 are different kinds of periodic frequency noise: high, medium and low. Pitch 4, 5 and 6 are different kinds of 'white noise'. Periodic frequencies sound like buzzes whilst 'white noise' sounds like the hiss heard between TV stations/channels when you tune a TV.

```
10 REM SOUND0 1
20 REM
40 volume=-15
50 duration=10
60 FOR pitch=0 TO 7
70 SOUND 0,volume,pitch,duration
80 DUMMY=GET
90 NEXT pitch
```

Pitch values 3 and 7 on the other hand give you more than three pitch options for periodic and white noise. The pitch of the noise is defined by the pitch of loudspeaker 1. If we turn the

volume off loudspeaker channel 1, we can hear the different pitch of the noise:

This next program demonstrates the sound of varying pitch periodic noise.

```
 10 REM SOUND0 2
 20 REM
 40 volume=-15
 50 duration=1
 60 FOR pitch=0 TO 255
 70 SOUND 1,0,pitch,duration
 80 SOUND 0,volume,3,duration
100 NEXT pitch
```

Input this line to hear the same in white noise:

```
80 SOUND 0,volume,7,duration
```

See chapter 12 to find out how you can create actual tunes with your programs.

There is another complicated sound command called ENVELOPE, which can vary the note being outputted by the SOUND command. Sound usually outputs a note with the same volume and pitch, that is when not controlled by ENVELOPE. The sound envelope is divided into two parts: the pitch envelope and the amplitude envelope. The pitch envelope is then sub-divided into three sections for use by the ENVELOPE parameters. The syntax for ENVELOPE looks like this:

ENVELOPE N, U, CS1, CS2, CS3, NS1, NS2, NS3, CASA, CASD, CASS, CASR, TA, TD

N = Envelope number (1 to 4)

U = Units to be used in ENVELOPE in hundredths of a second

CS1, CS2, CS3 = Changes in pitch per step in sections 1, 2 and 3

NS1, NS2, NS3 = Number of steps in section 1, 2 and 3

CASA, CASD, CASS, CASR = Changes in Amplitude per step during Attack, Decay, Sustain and Release phases

TA = Target amplitude at end of Amplitude stage

TD = Target amplitude at end of Decay stage.

As you can see there are many parameters needed to produce a sound.

The pitch envelope as we mentioned earlier is divided into three sections, 1, 2 and 3, each of these parts have two parameters: the change in pitch during one 'step' and the number of steps in that part. This means if the CS1 (the change in pitch per step for section 1) is 2 and the NS1 (the number of steps in section 1) is 10 the pitch will rise 20 notes during the ten steps. The length of the 'steps' is defined by the second number, in hundredths of a second.

The method by which the amplitude ENVELOPE is worked out is slightly different — in each of its four parts Attack, Decay, Sustain, Release each has a rate of change (CASA, CASS, CASD and CASR) but has no variable to dictate the length of time these four parts should continue changing. In our pitch example, the pitch was raised for Ø to 2Ø using the number of steps set by the variable NS1. However with amplitude a target level is given and the computer works out the number of steps needed to get to that level.

But, as you can see there are only two target amplitudes — for the first two sections (Attack and Decay) the target amplitude for sustain is not specified because sustain keeps on 'sustaining' until the end of the envelope (defined by the addition of the number of the steps during the pitch envelope multiplied by the number of centi-seconds per step). And the Release phase goes on until it reaches Ø volume.

Here is a short program for you to explore the versatility of the ENVELOPE statement. Use the cursor keys to copy and alter the previous envelope.

```
   10 REM Envelope
   20 REM
   60 REM
   70
   80 PROCinit
   90
  100 REPEAT
  110
  120 PROCalter
  130
  140 UNTIL FALSE
  150 DEF PROCalter
  160 ENVELOPE 1,E%(1),E%(2),E%(3),E%(4)
,E%(5),E%(6),E%(7),E%(8),E%(9),E%(10),E%
(11),E%(12),E%(13)
  170 SOUND 1,1,100,255
  180 PRINT"""Input next ENVELOPE "
  190 INPUT "ENVELOPE 1,"E%(1),E%(2),E%(
3),E%(4),E%(5),E%(6),E%(7),E%(8),E%(9),E
%(10),E%(11),E%(12),E%(13)
  200 ENDPROC
  210
  220 DEF PROCinit
  230 DIM E%(13)
```

```
  240 FOR T=1 TO 13
  250 READ E%(T)
  260 NEXT
  270 PRINT "ENVELOPE 1,";
  280 FOR T%=1 TO 13
  290 PRINT ;E%(T%);",";
  300 NEXT
  310 VDU 127
  320 ENDPROC
  330
  340 DATA 1,127,-127,127,4,255,6,0,0,0,
0,100,80
>
>
>
```

CHAPTER EIGHT
MAKING COMPARISONS

We all know the equals sign (=) and we've seen it in use in several program so far. We've also seen the greater than (>) and less than (<) signs. At this point of your learning, we thought it would be useful to briefly recap on what each of these signs are, and what they mean:

=	equals
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
<>	not equal to

You'll see these in use in many programs in this book, such as this next one, which allows you to challenge the computer to a game of BRICKBAT, written by Graham Charlton.

This is the listing for BRICKBAT.

```
10 REM Brickbat
20 REM
60 REM
```

```
   70
   80 MODE 1
   90 PROCinitialise
  100
  110 REPEAT
  120 PROCscreen
  130
  140 REPEAT
  150 PROCmove
  160 UNTIL L=0
  170
  180 UNTIL 0
  190
  200 PROCend
  210
  220 DEF PROCscreen
  230 COLOUR 3
  240 PRINT "'"   /=========================
==========\    |
                |    |
                |    |
                |"
  250 FOR L=1 TO 5
  260 FOR M=1 TO 20
  270 COLOUR 128+RND(3)
  280 PRINT "   ";
  290 NEXT
  300 NEXT
  310 COLOUR 128
  320 FOR T=6 TO 26
  330 PRINT TAB(0,T)"   |"TAB(38,T)"|  "
  340 NEXT
  350 SC=0
  360 VDU 30:PRINT SC;"0"TAB(15)LI;TAB(2
1)HI;"0"
```

```
370 C=-1:D=-1:X=19:Y=16+RND(6)
380 M=Y-2
390 ENDPROC
400
410
420 DEF PROCmove
430 Q=FNpoint(X,Y):IF Q<>0 THEN PROCdi
r
440 IF Y+C>29 PROCloose:PRINT TAB(M,27)
"      ":GOTO 360
450 PRINT TAB(X,Y)"*"
460 PRINT TAB(M,27)" ====== "
470 M=M+(INKEY(-98) AND M>0)-(INKEY(-6
7) AND M<34)
480 *FX 15
490 *FX 19
500 IF X+C<3 THEN C=-C : SOUND 1,-15,2
00,2
510 IF X+C>37 THEN C=-C: SOUND 1,-15,2
00,2
520 IF Y=27 AND X>M AND X<M+6 THEN PRO
Cdirect
530 IF Y+D<3 THEN D=-D:  SOUND 1,-15,1
00,2
540 IF X+C>4 AND X+C<36 PRINT TAB(X-(R
ND(2)-1),Y)"  " : ELSE PRINT TAB(X,Y)" "
550 X=X+C:Y=Y+D
560 ENDPROC
570
580
590 DEF FNpoint(X,Y)
600 LOCAL X6,Y6
610 X6=X*32+4
620 Y6=1024-(Y*32+4)
630=POINT(X6,Y6)
```

```
640
650
660 DEF PROClose
670 LI=LI-1
680 PRINT TAB(0,27)"
                    "
690 ENDPROC
700
710
720 DEF PROCdirect
730 D=-D
740 IF X=M+3 C=0
750 IF X>M+3 C=1
760 IF X<M+3 C=-1
770 SOUND 1,-15,150,3
780 ENDPROC
790
800
810 DEF PROCdir
820 SOUND 1,-15,250,3
830 SC=SC+1
840 IF D=-1 AND RND(6)<6 THEN D=1:GOTO
870
850 IF D=1 AND RND(6)>5 THEN D=-1:GOTO
870
860 D=D
870 PRINT TAB(0,0) SC;"0"
880 ENDPROC
890
900
910 DEF PROCinitialise
920 F=0
930 HI=50
940 LI=5
950 VDU 19,0,4,0,0,0,0,0
960 VDU 23;8202;0;0;0;
970 ENDPROC
```

In BRICKBAT, you use the Z and X keys to move the little black slide at the bottom of the screen back and forth, bouncing the ball off it (if you can). As the ball bounces up to the coloured X's above, it will strike them, making a sound, adding your score and causing the ball to disappear. You have ten balls per round. It is fascinating to see what happens if the ball gets momentarily 'trapped' *inside* the bricks, bouncing off them wildly, before it finds a way out and down. You'll see how effective this can be when you run the program.

Notice that our 'comparison symbols' (less than, and the rest) are used frequently in this program. They perform a number of tests in conjunction with IF/THEN statements, bouncing the ball off the slide at the bottom of the screen, the walls or the bricks, deciding when the ball has missed the slide, and continuing the game if (in line 16Ø) you still have a 'life' (that is, ball) left. The program restarts itself and can only be stopped by using ESCAPE or BREAK. If you do happen to press BREAK it is similar to typing NEW, in that it appears to delete your program from memory. Fortunately all is not lost. If you now type OLD before doing anything else the program will be restored to its original self.

The words AND and OR are also used in comparison lines, *chaining* two or more tests together, as in line 47Ø. They work as follows:

AND The computer does what follows the THEN if *both* of the conditions chained by the AND are true

OR The computer carries out the instruction following the THEN if *either* of the conditions are true.

Let's see how this works in practice. In line 47Ø, the computer checks to see IF you are passing the M key AND if M is less than 34, and if *both* these conditions are true, moves the slide one square to the right. Lines 5ØØ and 51Ø bounce the ball off the side walls. They check to see if the ball is about to hit the left

wall (IF Y + C< 3) or, in line 51Ø, if the ball is about to hit the right wall (IF Y + C> 37) and if *either* of these conditions is found to be true, the computer 'bounces' the ball, by changing the value of C, the variable which determines the change in the position of the ball across the screen from move to move.

CHAPTER NINE
TWO GAMES AND A SPEED TEST

It's time now to take a break from the serious business of learning to program the computer. As you can see in this chapter, we have a couple of major programs, which use many commands that have not yet been explained. We suggest you enter the programs just as they are, play them for your own enjoyment, then come back to the explanations which follow the listings *after* you have mastered the rest of the book. We do not think it is fair to keep you waiting for major programs until you've covered everything on the computer, so have decided to supply you with these programs at this point, and hope you'll enter then 'on trust', returning to this chapter for the explanations when you feel you are ready. Of course, you do not have to enter the programs right now. If you'd prefer to read the explanations first please do so.

PLAYING ALONE

Our first listing allows you to use your computer as a solitaire board. It is believed that Solitaire was invented by an imprisoned nobleman in France in the late 16th century, and was brought into England in the closing decade of the 18th century.

The aim of the game is simple to explain, but not so easy to achieve. You start on a board which has 33 positions marked.

There are 'pegs' in 32 of the positions on a real board, and the middle position is empty. To play, you simply jump over any of your pieces vertically or horizontally, so that you end up in an empty position. The piece which you have jumped over is removed from the board. The aim of the game is to end up with just one little man in the centre position.

As you can see, you are told the number of the move, and how many pieces are left on the board. You move by entering the co-ordinates of the piece you want to move, using the number down the side first, followed by the number across the top. These are entered as a single, double-digit number. If you wanted to move the piece which was two positions below the central hole at the start of the game, jumping into the central position, you'd enter 64, then press RETURN, followed by 44, and RETURN again. The board is reprinted, and you are then offered another move.

Here's how the program starts:

```
Enter side co-ord's first
Enter 99 to concede
        1 2 3 4 5 6 7

      1       # # #       1
      2       # # #       2
      3 # # # # # # # 3
      4 # # # * # # # 4
      5 # # # # # # # 5
      6       # # #       6
      7       # # #       7

        1 2 3 4 5 6 7
Moves so far:0

Which peg do you want to move?
```

```
 10 REM Solitaire
 20 REM
 60 REM
 70
 80 MODE 7
 90
100 PROCinit
110
120 REPEAT
130 PROCprint_board
140 PROCloop
150 UNTIL C=1
160
170 PROCend
180
190 DEF PROCloop
200 REM Accept Move
210 SOUND 1,-15,200,5
220 PRINT TAB(1,21)gl$;"Which peg do y
ou want to move";
230 INPUT A
240 PRINT TAB(1,21)"
                    "
250 IF A=99 THEN 400
260 IF A<11 OR A>77 THEN 210
270 IF A(A)<>P THEN 210
280 SOUND 1,-15,150,5
290 PRINT TAB(1,21)A;yl$;"to....";
300 INPUT B
310 PRINT TAB(1,21)"
                    "
320 IF B<11 OR B>77 THEN 280
```

```
  330 IF A(B)<>E THEN 280
  340 SOUND 1,-15,102,5
  350 A((A+B)/2)=E:A(A)=E:A(B)=P
  360 M=M+1
  370 C=0
  380 FOR F=11 TO 75
  390 IF A(F)=P THEN C=C+1
  400 NEXT
  410 PRINT TAB(3,1)cl$;"There are ";C;"
 pegs on the board "
  420 ENDPROC
  430
  440
  450 DEF PROCend
  460 IF A(44)=P THEN PRINT TAB(1,22)rl$
;"You did it!! in just ";M;" moves!!":EN
D
  470 PRINT TAB(1,22)bl$;"You failed!!Ga
me over...":END
  480
  490
  500 DEF PROCprint_board
  510 PRINT TAB(6,6)ml$;"Enter 99 to con
cede"
  520 PRINT "      "+rl$+nb$+yl$+"  1 2
3 4 5 6 7      "+bb$
  530 PRINT "      "+rl$+nb$+yl$+"
              "+bb$
  540 PRINT "      "+rl$+nb$+yl$+"1"+bl$+
" "+
  550 FOR D=11 TO 75
  560 T=10*(INT(D/10))
  570 IF D-T=8 THEN D=D+3:PRINT+yl$+T/10
;"  "+bb$:PRINT"      "+rl$+nb$+yl$+T/10+
1+bl$+" "+
```

```
  580 VDU A(D),32
  590 NEXT
  600 PRINT;"     ";yl$;"7  ";bb$
  610 PRINT "      ";rl$;nb$;yl$;"
            ";bb$
  620 PRINT "      ";rl$;nb$;yl$;"   1 2
3 4 5 6 7     ";bb$
  630 PRINT TAB(0,19)"Moves so far:";M
  640 PRINT''
  650 ENDPROC
  660
  670
  680 DEF PROCinit
  690 VDU 23;8202;0;0;0;
  700 CLS
  710 PROCcolours
  720 DIM A(87)
  730 E=42
  740 P=35
  750 FOR D=11 TO 75
  760 T=10*(INT(D/10))
  770 IF D-T=8 THEN D=D+3
  780 READ A(D)
  790 NEXT
  800 PRINT TAB(3,1);fl$;"Enter side co-o
rd's first"
  810 M=0
  820 ENDPROC
  830
  840
  850 DEF PROCcolours
  860 dl$=CHR$ 141
  870 rl$=CHR$ 129
  880 gl$=CHR$ 130
  890 yl$=CHR$ 131
```

```
 900 bl$=CHR$ 132
 910 ml$=CHR$ 133
 920 cl$=CHR$ 134
 930 nb$=CHR$ 157
 940 bb$=CHR$ 156
 950 fl$=CHR$ 136
 960 ENDPROC
 970
 980 DATA 32,32,35,35,35,32,32
 990 DATA 32,32,35,35,35,32,32
1000 DATA 35,35,35,35,35,35,35
1010 DATA 35,35,35,42,35,35,35
1020 DATA 35,35,35,35,35,35,35
1030 DATA 32,32,35,35,35,32,32
1040 DATA 32,32,35,35,35,32,32
```

```
  10 REM Solitaire
  20 REM
  30 REM By Alex Gollner
  40 REM
  50 REM (c) A. Gollner 1983
  60 REM
  70
  80 MODE 1
  90
 100 PROCinit
 110
 120 REPEAT
 130 PROCprint_board
 140 PROCloop
 150 UNTIL C=1
 160
 170 PROCend
 180
 190 DEF PROCloop
```

```
  200 REM Accept Move
  210 SOUND 1,-15,200,5
  220 PRINT TAB(1,22)"Which peg do you w
ant to move";
  230 INPUT A
  240 PRINT TAB(1,22)"
                      "
  250 IF A=99 THEN PROCend
  260 IF A<11 OR A>77 THEN 220
  270 IF A(A)<>P THEN 220
  280 SOUND 1,-15,150,5
  290 PRINT TAB(1,22)A;" to..";
  300 INPUT B
  310 PRINT TAB(1,22)"
                      "
  320 IF B<11 OR B>77 THEN 290
  330 IF A(B)<>E THEN 290
  340 SOUND 1,-15,102,5
  350 A((A+B)/2)=E:A(A)=E:A(B)=P
  360 M=M+1
  370 C=0
  380 FOR F=11 TO 75
  390 IF A(F)=P THEN C=C+1
  400 NEXT
  410 ENDPROC
  420
  430
  440 DEF PROCend
  450 IF A(44)=P THEN PRINT TAB(1,22)" Y
ou did it!! in just ";M;" moves!!":END
  460 PRINT TAB(1,22)"You failed!!Game o
ver...":END
  470
  480
  490 DEF PROCprint_board
```

```
  500 COLOUR 2
  510 PRINT TAB(3,1)"Enter side co-ord'
 first"
  520 PRINT''''TAB(6)'Enter 99 to concede
"
  530 COLOUR 3
  540 PRINT'"            1 2 3 4 5 6 7
 "
  550 PRINT'"          1 "
  560 COLOUR 1
  570 FOR D=11 TO 75
  580 T=10*(INT(D/10))
  590 IF D-T=8 THEN D=D+3:COLOUR 3:PRINT
;CHR$(127);" ";T/10;"    ":PRINT"
";T/10+1;" ";:COLOUR 1
  600 VDU A(D),32
  610 NEXT
  620 COLOUR 3
  630 PRINT;"    7  "
  640 PRINT'"            1 2 3 4 5 6 7
 "
  650 COLOUR 2
  660 PRINT''"Moves so far:";M
  670 PRINT''
  680 PRINT TAB(0,26)"There are ";C;" pe
gs on the board  "
  690 ENDPROC
  700
  710
  720 DEF PROCinit
  730 VDU 23,235,60,126,126,60,24,24,24,
0
  740 VDU 23,236,0,0,0,0,60,195,60,0
  750 VDU 19,0,4,0,0,0,0,0
  760 VDU 23;8202;0;0;0;
```

```
770 CLS
780 DIM A(87)
790 E=236
800 F=235
810 C=32
820 FOR D=11 TO 75
830 T=10*(INT(D/10))
840 IF D-T=8 THEN D=D+3
850 READ A(D)
860 NEXT
870 M=0
880 ENDPROC
890
900 DATA 232,232,235,235,235,232,232
910 DATA 232,232,235,235,235,232,232
920 DATA 235,235,235,235,235,235,235
930 DATA 235,235,235,236,235,235,235
940 DATA 235,235,235,235,235,235,235
950 DATA 232,232,235,235,235,232,232
960 DATA 232,232,235,235,235,232,232
```

You may have discovered by now that your computer has various display MODEs. All the modes are different and can be used for different applications. The previous program 'SOLITAIRE' had two versions, one in MODE 7 and the other in MODE 1.

Modes Ø to 6 are similar in that they all use the same character set, but some modes take up more memory and have different character widths. The first three modes (Ø to 2) use the same amount of memory but have different features.

MODE	0	1	2
Number of colours	2	4	16
Characters per line	80	40	20

The MODES with more characters per line have thinner characters so that they will fit on the screen.

In this book most of the programs run in either MODE 7 or MODE 1, this is because MODE 1 has clear graphics, four colours and clear lettering (this is good for action games). MODE 7 has the clearest character set (alphabet) and is the odd MODE out. To convert programs from one MODE to another a few changes are necessary.

MODE	0	1	2	3	4	5	6	7
Lines of text	32	32	32	25	32	32	25	25
Characters per line	80	40	20	80	40	20	40	40
Graphics	Yes	Yes	Yes	No	Yes	Yes	No	No

The two previous programs showed the different advantages of MODE 1 with its user-defined graphics to MODE 7 with clear text and colour handling.

In the MODE 7 program the colour is controlled through the use of strings of control characters (like CONTROL G) they are used in the form of string variables related to what they do — here are some examples:

rl$ = CHR$ (129) (Red letters)
nb$ = CHR$ (157) (New background)
fl$ = CHR$ (136) (Flashing characters)

See chapter 12 for more information on MODE 7.

TESTING YOUR SPEED

Our next program, REACTION TESTER, is much shorter than Solitaire, but just as much fun to play. You enter the program, press RUN, and the message STAND BY appears. After an agonizing wait, STAND BY will vanish, to be replaced with a flashing notice: "OK, PRESS THE 'Z' KEY". As fast as you can, you leap for the Z key, and press it, knowing that the computer is counting all the time.

The Computer then tells you how quickly you reacted, and compares this with your previous best time. The best score so far appears on the screen, and the computer then waits for you to press a key. The game continues until you manage to get your reaction time down to below a tenth of a second which is not an easy task!

Here is the listing of REACTION TEST:

```
10 REM Reaction Test
20 REM
```

```
   60 REM
   70
   80 HISCORE=500
   90 CLS
  100 PRINT TAB(10,5)"STAND
BY.........."
  110 FOR A=1 TO 3000+RND(1500)
  120 NEXT
  130 PRINT TAB(10,5)"O.K. PRESS THE 'Z'
KEY"
  140 TIME=0
  150 REPEAT
  160 A=INKEY(0)
  170 UNTIL A=90
  180 G=TIME
  190 PRINT ''"Your Score Was..
";G/100;" Seconds.."
  200 IF G<HISCORE THEN HISCORE=G
  210 PRINT ''"So the best score so far
is ";HISCORE/100'"   seconds..."
  220 DUMMY=GET
  230 IF HISCORE>10 GOTO 90
  240 PRINT ''"Well Done...You're the
Champ!!"
```

Line 80 sets the variable HISCORE to 500. The variable TIME is set to zero in line 140 and is incremented by 1 every 1/100th of a second by the computer (this is a feature of BBC BASIC for use in timing). The computer REPEATS UNTIL the Z key is pressed and the variable G is set to the value of TIME when the computer comes out of its keyboard checking loop. There are two ways of checking the keyboard using the keywords INKEY and INKEY$. These are both followed by a number of brackets. This tells the computer how long to wait before it gives up searching the keyboard. The only difference between

the two INKEY commands is the fact that INKEY reads the value of the key pressed while INKEY$ reads the string produced by the key pressed. If either of these is followed by a value of Ø in the brackets the keyboard is read when the program executes the line. They thus check if a key is being pressed when the INKEY command is executed. If no key is pressed during the time limit set then a null string (" " — a string of no length) or −1 is returned.

The keywords GET and GET$ work in the same way except that they make the computer wait until a key is pressed before the program continues.

G is compared to the best score (variable name HISCORE) in the following line, and HISCORE is adjusted to G if G is the lower of the two.

Line 23Ø checks to see if you have managed to get a score lower than 1/1Ø second, and if you have, does not loop back to line 9Ø but prints the "Well done... you're the champ!" message in line 24Ø.

A CORNER ON GO

Our next program is a fascinating one. It allows you to play against the computer in the Japanese board game *Hasami Shogi*. Whereas the board games with which you might be familiar — such as Chess and Checkers — are played on an eight by eight board, Hasami Shogi is played on a nine by nine board. The corner of a *Go* board (19 by 19) is often used.

Each player starts the game with 18 stones. Your stones look like letter "H", and the computer's stones look like letter "C".

You start at the bottom of the board playing up the screen, and the computer starts at the top. The aim of the game is to capture seven of your opponent's pieces. Actually, in the real-life Hasami Shogi, the aim of the game is to capture all of your opponent's pieces, but — as you'll see when you play this game — that would make each game last a long, long time.

You take it in turn to move, and you can move only one piece per turn. You move vertically or horizontally, but not diagonally. With each move, you have three choices:

- you can move into a vacant square which is above, below or beside your piece
- you can jump over one of your own pieces into a vacant square
- you can jump over an opponent piece into an empty square

Unlike Checkers, a piece which has been jumped over is not captured, and is not removed from the board. The only way you can capture a piece is by moving so that your piece *traps* a computer piece between two of yours. It captures your piece in a similar way. You do *not* lose your piece simply by moving in between two computer pieces. Therefore, you aim to get your piece next to a computer piece, with an empty square beyond that, so you can move into the empty square on a subsequent move.

```
10 REM Hasami Shogi
20 REM
60 REM
70
80 MODE 7
90
```

```
100 PROCinitialise
110
120 REPEAT
130 PROCcomputer_move
140 PROCprint_board
150 PROCplayer_move
160 PROCprint_board
170 UNTIL FALSE
180
190
200 DEF PROCcomputer_move
210 A=99
220 IF A(A)<>C THEN 300
230 IF A(A-10)=E THEN IF A(A-9)=H THEN
IF A(A-8)=C THEN B=A-10:GOTO 470
240 IF A(A-10)=E THEN IF A(A-11)=H THE
N IF A(A-12)=C THEN B=A-10:GOTO 470
250 IF A(A-10)=E THEN IF A(A+11)=H THE
N IF A(A+12)=C THEN B=A-10:GOTO 470
260 B=1
270 IF A+2*C(B)<11 OR A+2*C(B)>99 THEN
290
280 IF A(A+C(B))=E AND A(A+2*C(B))=H A
ND A(A+3*C(B))=C THEN A(A+2*C(B))=E:CS=C
S+1:GOTO 450
290 IF B<4 THEN B=B+1:GOTO 270
300 IF A>11 THEN A=A-1:GOTO 220
310 REM NON-CAPTURE
320 count=0
330 count=count+1
340 A=RND(1)*89+10
350 IF A(A)=C THEN 380
360 IF count<200 THEN 330
370 PRINT"You are now the Hasami Shogi
Master.."'"I give you the victory!":END
```

```
  380 B=i
  390 IF A+2*C(B)<11 THEN 410
  400 IF (A(A+C(B))=C OR A(A+C(B))=H)AND
 A(A+2*C(B))=E THEN B=A+2*C(B):GOTO 470
  410 IF (A(A+C(B))=E THEN 440
  420 IF B<4 THEN B=B+1:GOTO 390
  430 GOTO 360
  440 REM COMPUTER MOVES
  450 B=A+C(B)
  460 FOR T=1 TO 100:NEXT
  470 B1=B-10*(INT(B/10))
  480 A(B)=C:A(A)=E
  490 IF B1>7 THEN 510
  500 IF A(B+1)=H AND A(B+2)=C THEN A(B+
1)=E:CS=CS+1
  510 IF B1<3 THEN 530
  520 IF A(B-1)=H AND A(B-2)=C THEN A(B-
1)=E:CS=CS+1
  530 IF A>89 THEN 550
  540 IF A(B+10)=H AND A(B+20)=C THEN A(
B+10)=E:CS=CS+1
  550 IF A<29 THEN ENDPROC
  560 IF A(B-10)=H AND A(B-20)=C THEN A(
B-10)=E:CS=CS+1
  570 ENDPROC
  580
  590
  600 DEF PROCprint_board
  610 VDU 30
  620 PRINT TAB(12,22);CHR$(141);bb$;mi$
;"HASAMI SHOGI"Bb$
  630 PRINT TAB(12,23);CHR$(141);bb$;mi$
;"HASAMI SHOGI"Bb$
  640 PRINT TAB(10,2)bb$;ri$;"1 2 3 4 5
6 7 8 9";Bb$
```

```
  650 FOR M=90 TO 10 STEP -10
  660 PRINT TAB(8,((100-M)/10)+2)bb$;gl$
;CHR$(M/10+64);
  670 FOR N=1 TO 9
  680 PRINT;cl$;CHR$(A(M+N));
  690 NEXT
  700 PRINT;gl$;CHR$(M/10+64);Bb$;
  710 NEXT
  720 PRINT TAB(10,12)bb$;rl$;"1 2 3 4 5
 6 7 8 9"Bb$
  730 PRINT TAB(4,15)"Computer:";CS
  740 PRINT TAB(4,16)"Human:";HS
  750 IF CS>6 OR HS>6 THEN 770
  760 ENDPROC
  770 IF CS>HS THEN PRINT''"I win !!!!":
END
  780 PRINT''"You win !!!!":END
  790
  800
  810 DEF PROCplayer_move
  820 VDU 31,4,18
  830 SOUND 1,-15,200,5
  840 INPUT "From (letter,No.)",A$
  850 IF A$="S" THEN END
  860 IF LEN(A$)<>2 THEN 840
  870 VDU 11,9,9,9:PRINT "
              "
  880 SOUND 1,-15,150,5
  890 VDU 31,4,20:PRINT "From ";A$;" to
";:INPUT B$
  900 IF LEN(B$)<>2 THEN 890
  910 VDU 11,9,9,9:PRINT "
              "
  920 SOUND 1,-15,102,5
  930 A=10*(ASC(A$)-64)+VAL(RIGHT$(A$,1)
```

```
 940 B=10*(ASC(B$)-64)+VAL(RIGHT$(B$,1)

 950 Y=VAL(RIGHT$(B$,1))
 960 A(B)=H:A(A)=E
 970 IF A(B+1)=C AND A(B+2)=H AND Y<=7
THEN A(B+1)=E:HS=HS+1
 980 IF A(B-1)=C AND A(B-2)=H AND Y>=3
THEN A(B-1)=E:HS=HS+1
 990 IF B>79 THEN 1010
1000 IF A(B+10)=C AND A(B+20)=H THEN A(
B+10)=E:HS=HS+1
1010 IF B>=31 THEN IF A(B-10)=C AND A(B
-20)=H THEN A(B-10)=E:HS=HS+1
1020 ENDPROC
1030
1040
1050 DEF PROCinitialise
1060 CLS
1070 VDU 23;8202;0;0;0;
1080 DIM A(129),C(4)
1090 r1$=CHR$ 129
1100 g1$=CHR$ 130
1110 c1$=CHR$ 134
1120 bb$=CHR$ 156
1130 8b$=CHR$  32 + CHR$ 132 + CHR$ 157
1140 m1$=CHR$ 133
1150 H=72
1160 C=67
1170 E=255
1180 FOR Z=11 TO 29
1190 IF Z=20 THEN Z=21
1200 A(Z)=H
1210 NEXT
1220 FOR Z=31 TO 79
1230 IF 10*INT(Z/10)=Z THEN Z=Z+1
```

```
1240 A(Z)=E
1250 NEXT
1260 FOR Z=81 TO 99
1270 IF Z=90 THEN Z=91
1280 A(Z)=C
1290 NEXT
1300 HS=0
1310 CS=0
1320 FOR Z=1 TO 4
1330 READ C(Z)
1340 NEXT
1350 DATA -10,-1,1,10
1360 PROCprint_board
1370 FOR I=0 TO 24
1380 VDU 31,0,I,148,157,131
1390 NEXT
1400 ENDPROC
```

The program may seem very long, but it is somewhat simpler to understand than may at first appear. This is because it has been written using 'procedures'. These are lines separate from the main program and are called from the program by PROCname_of_procedure. To name a procedure, the first line of it starts DEF PROCname_of _procedure. The end of the group of lines is marked by the keyword ENDPROC. Once the procedure has been 'defined' it can be called from any part of the program.

Programming in this way is a good idea if you are developing a program which you think may be fairly long and involved: it enables you to keep track of what each section is doing. It also makes it easy to track down *bugs* (the common computer term for a program error, named after a malfunction in an early IBM computer which was said to have been the result of a moth causing a shortcircuit when it had got caught inside the cabinet). If the program is in procedures, the procedure which contains the bug is relatively easy to isolate.

The idea of 'structured programming' (i.e. using procedures) may become a little clearer if we understand how it works in this program:

MODE in procedures

Line 100 — Executes PROCinitialise which initialises all the variables in the program (starts at line 1050). It also calls PROCprint_board which displays the screen while the computer thinks where it will move.

Line 120 — Starts what is called a REPEAT/UNTIL loop — this is a loop which continually executes the lines between the REPEAT and the UNTIL, until the condition specified in the UNTIL is satisfied.

For example with REPEAT....UNTIL X>34 the program will loop from the REPEAT line to the UNTIL statement UNTIL X is greater than 34. Then it goes onto the next line. If after the UNTIL it says FALSE or 0 the loop loops forever, or until one of the lines in between the REPEAT and the UNTIL goes somewhere else.

Line 130 — This makes the computer move its stone with PROCcomputer_move.

Line 140 — Prints out the screen with PROCprint_board.

Line 150 — Calls PROCplayer_move in which the player's move is accepted.

Line 160 — Prints the board again with PROCprint_board.

Line 170 — Loops back to make the computer take its next move with PROCcomputer_move.

There are four ways the player can get out of the "infinite"

REPEAT/UNTIL ∅. Firstly the computer or you can win — this is detected during the procedure that prints out the board. You can concede by entering "S" instead of your move. The computer concedes if it cannot find a move after 2∅∅ attempts.

The first lines of the HASAMI SHOGI program can be applied to nearly any "computer versus human" boardgame program.

```
Initialise
REPEAT
Computer_moves
Print_board
Player_moves
Print_board
UNTIL FALSE
```

See if you can incorporate procedures in your programs so they can be altered with ease. Structured programming also aids the readability of programs, for example, IF A$ = "YES" THEN PROCinstructions ELSE PROCstart_game. Note that there cannot be spaces in procedure names.

CHAPTER TEN
STRINGING ALONG

You'll recall that several times in this book so far we have referred to *numeric variables* (letters like A or B, words like COUNT and GUESS, and combinations such as R2D2 or C3PO) and to *string variables* (a single letter followed by a dollar sign, such as a$ or A$ is a string variable). In this chapter we'll be looking at strings, and at things you can do with them.

THE CHARACTER SET

Every letter, number or symbol the computer prints has a ASCII code which stands for American Standard Code for Information Interchange (pronounced As-key), an industry standard. Telling the computer to print the *character* of that ASCII code produces the character. It is easy to understand this. The ASCII character code of the letter "A" has nothing to do with the value assigned to A when it is a numeric variable, but refers to "A" when we actually want the computer to print the letter "A" out. We'll put the "A" in quote marks when referring to it as a letter. Try it now. Enter this into your computer and see what you get:

```
>PRINT ASC("A")
```

You should get an answer of 65. Now 65, is the code of the letter "A". We can turn it back into an "A" by asking the computer to print the character that corresponds to code 65. We do this with the symbol CHR$.

```
>PRINT CHR$(65)
```

You can get the computer to print out every character with the next short program, which has — as you can see — a very long print out. Before running the program, to stop the display racing off the screen, press CONTROL N. When the program is run it stops when the bottom line on the screen is printed — then just touching the SHIFT key will print the next screenful, and so on. This is called putting the computer into "page mode". To get out of page mode press CONTROL O (CTRL and O simultaneously).

```
10 REM Character Generation
25 FOR A=32 TO 255
30 PRINT "ASCII Character ";A"===>";C
HR$(A)
40 NEXT
```

You will get something like this, but not identical (our printer happens to have a different character set from that of the computer. Using this program in different MODES you can see how the widths of the characters differ.

Some of the characters output by this program affect the printer so that the output changes a little in the middle of the program.

```
ASCII Character 32===>
ASCII Character 33===>!
ASCII Character 34===>"
ASCII Character 35===>£
ASCII Character 36===>$
ASCII Character 37===>%
ASCII Character 38===>&
ASCII Character 39===>'
ASCII Character 40===>(
ASCII Character 41===>)
ASCII Character 42===>*
```

```
ASCII Character 43===>+
ASCII Character 44===>,
ASCII Character 45===>-
ASCII Character 46===>.
ASCII Character 47===>/
ASCII Character 48===>0
ASCII Character 49===>1
ASCII Character 50===>2
ASCII Character 51===>3
ASCII Character 52===>4
ASCII Character 53===>5
ASCII Character 54===>6
ASCII Character 55===>7
ASCII Character 56===>8
ASCII Character 57===>9
ASCII Character 58===>:
ASCII Character 59===>;
ASCII Character 60===><
ASCII Character 61===>=
ASCII Character 62===>>
ASCII Character 63===>?
ASCII Character 64===>@
ASCII Character 65===>A
ASCII Character 66===>B
ASCII Character 67===>C
ASCII Character 68===>D
ASCII Character 69===>E
ASCII Character 70===>F
ASCII Character 71===>G
ASCII Character 72===>H
ASCII Character 73===>I
ASCII Character 74===>J
ASCII Character 75===>K
ASCII Character 76===>L
ASCII Character 77===>M
```

```
ASCII Character 78===>N
ASCII Character 79===>O
ASCII Character 80===>P
ASCII Character 81===>Q
ASCII Character 82===>R
ASCII Character 83===>S
ASCII Character 84===>T
ASCII Character 85===>U
ASCII Character 86===>V
ASCII Character 87===>W
ASCII Character 88===>X
ASCII Character 89===>Y
ASCII Character 90===>Z
ASCII Character 91===>[
ASCII Character 92===>\
ASCII Character 93===>]
ASCII Character 94===>^
ASCII Character 95===>_
ASCII Character 96===>`
ASCII Character 97===>a
ASCII Character 98===>b
ASCII Character 99===>c
ASCII Character 100===>d
ASCII Character 101===>e
ASCII Character 102===>f
ASCII Character 103===>g
ASCII Character 104===>h
ASCII Character 105===>i
ASCII Character 106===>j
ASCII Character 107===>k
ASCII Character 108===>l
ASCII Character 109===>m
ASCII Character 110===>n
ASCII Character 111===>o
ASCII Character 112===>p
```

```
ASCII Character 113===>q
ASCII Character 114===>r
ASCII Character 115===>s
ASCII Character 116===>t
ASCII Character 117===>u
ASCII Character 118===>v
ASCII Character 119===>w
ASCII Character 120===>x
ASCII Character 121===>y
ASCII Character 122===>z
ASCII Character 123===>{
ASCII Character 124===>|
ASCII Character 125===>}
ASCII Character 126===>~
ASCII Character 127===>
ASCII Character 128===>
ASCII Character 129===>
ASCII Character 130===>
ASCII Character 131===>
ASCII Character 132===>
ASCII Character 133===>
ASCII Character 134===>
ASCII Character 135===>
ASCII Character 136===>
ASCII Character 137===>
ASCII Character 138===>
ASCII Character 139===>
ASCII Character 140===>
ASCII Character 141===>
ASCII Character 142===>
ASCII Character 143===>
ASCII Character 144===>
ASCII Character 145===>
ASCII Character 146===>
ASCII Character 147===>
```

```
ASCII Character 148===>
ASCII Character 149===>
ASCII Character 150===>
ASCII Character 151===>
ASCII Character 152===>
ASCII Character 153===>
ASCII Character 154===>
ASCII Character 155===>
ASCII Character 156===>
ASCII Character 157===>
ASCII Character 158===>
ASCII Character 159===>
ASCII Character 160===>
ASCII Character 161===>¡
ASCII Character 162===>¢
ASCII Character 163===>£
ASCII Character 164===>¤
ASCII Character 165===>¥
ASCII Character 166===>¦
ASCII Character 167===>§
ASCII Character 168===>¨
ASCII Character 169===>©
ASCII Character 170===>ª
ASCII Character 171===>«
ASCII Character 172===>¬
ASCII Character 173===>
ASCII Character 174===>®
ASCII Character 175===>¯
ASCII Character 176===>°
ASCII Character 177===>±
ASCII Character 178===>²
ASCII Character 179===>³
ASCII Character 180===>´
ASCII Character 181===>µ
ASCII Character 182===>¶
```

```
ASCII Character 183===>7
ASCII Character 184===>8
ASCII Character 185===>9
ASCII Character 186===>:
ASCII Character 187===>;
ASCII Character 188===><
ASCII Character 189===>=
ASCII Character 190===>>
ASCII Character 191===>?
ASCII Character 192===>@
ASCII Character 193===>A
ASCII Character 194===>B
ASCII Character 195===>C
ASCII Character 196===>D
ASCII Character 197===>E
ASCII Character 198===>F
ASCII Character 199===>G
ASCII Character 200===>H
ASCII Character 201===>I
ASCII Character 202===>J
ASCII Character 203===>K
ASCII Character 204===>L
ASCII Character 205===>M
ASCII Character 206===>N
ASCII Character 207===>O
ASCII Character 208===>P
ASCII Character 209===>Q
ASCII Character 210===>R
ASCII Character 211===>S
ASCII Character 212===>T
ASCII Character 213===>U
ASCII Character 214===>V
ASCII Character 215===>W
ASCII Character 216===>X
ASCII Character 217===>Y
```

```
ASCII Character 218===>Z
ASCII Character 219===>[
ASCII Character 220===>\
ASCII Character 221===>]
ASCII Character 222===>^
ASCII Character 223===>_
ASCII Character 224===>`
ASCII Character 225===>a
ASCII Character 226===>b
ASCII Character 227===>c
ASCII Character 228===>d
ASCII Character 229===>e
ASCII Character 230===>f
ASCII Character 231===>g
ASCII Character 232===>h
ASCII Character 233===>i
ASCII Character 234===>j
ASCII Character 235===>k
ASCII Character 236===>l
ASCII Character 237===>m
ASCII Character 238===>n
ASCII Character 239===>o
ASCII Character 240===>p
ASCII Character 241===>q
ASCII Character 242===>r
ASCII Character 243===>s
ASCII Character 244===>t
ASCII Character 245===>u
ASCII Character 246===>v
ASCII Character 247===>w
ASCII Character 248===>x
ASCII Character 249===>y
ASCII Character 250===>z
ASCII Character 251===>{
ASCII Character 252===>|
```

```
ASCII Character 253===>}
ASCII Character 254===>~
ASCII Character 255===
```

A more convenient version of this program — which takes up less time and space — is as follows:

```
10 REM Character Generation
25 FOR A=32 TO 255 STEP 4
30 PRINT ;A"==>";CHR$(A);"   ";A+1"==>
";CHR$(A+1);"   ";A+2"==>";CHR$(A+2);"   "
;A+3"==>";CHR$(A+3)
40 NEXT
```

You can see from the printout, that some codes affect the printer output...

```
32==>     33==>!    34==>"    35==>£
36==>$    37==>%    38==>&    39==>'
40==>(    41==>)    42==>*    43==>+
44==>,    45==>-    46==>.    47==>/
48==>0    49==>1    50==>2    51==>3
52==>4    53==>5    54==>6    55==>7
56==>8    57==>9    58==>:    59==>;
60==><    61==>=    62==>>    63==>?
64==>@    65==>A    66==>B    67==>C
68==>D    69==>E    70==>F    71==>G
72==>H    73==>I    74==>J    75==>K
76==>L    77==>M    78==>N    79==>O
80==>P    81==>Q    82==>R    83==>S
84==>T    85==>U    86==>V    87==>W
88==>X    89==>Y    90==>Z    91==>[
92==>\    93==>]    94==>^    95==>_
96==>`    97==>a    98==>b    99==>c
```

```
100==>d  101==>e  102==>f  103==>g
104==>h  105==>i  106==>j  107==>k
108==>l  109==>m  110==>n  111==>o
112==>p  113==>q  114==>r  115==>s
116==>t  117==>u  118==>v  119==>w
120==>x  121==>y  122==>z  123==>{
124==>|  125==>}  126==>~  127==>
128==>  129==>  130==>  131==>
132==>  133==>  134==>  135==>
136==>  137==>  138==>
  139==>

140==>
  141==>  142==>  143==>
144==>  145==>  146==>  147==>
148==>  149==>  150==>  151==>
152==>  153==>  154==>  155==>
156==>  157==>  158==>  159==>
160==>  161==>¡  162==>¢  163==>£
164==>¤  165==>¥  166==>¦  167==>§
168==>¨  169==>©  170==>ª  171==>«
172==>¬  173==>  174==>®  175==>¯
176==>°  177==>±  178==>²  179==>³
180==>´  181==>µ  182==>¶  183==>·
184==>¸  185==>¹  186==>º  187==>»
188==>¼  189==>½  190==>¾  191==>¿
192==>À  193==>Á  194==>Â  195==>Ã
196==>Ä  197==>Å  198==>Æ  199==>Ç
200==>È  201==>É  202==>Ê  203==>Ë
204==>Ì  205==>Í  206==>Î  207==>Ï
208==>Ð  209==>Ñ  210==>Ò  211==>Ó
212==>Ô  213==>Õ  214==>Ö  215==>×
216==>Ø  217==>Ù  218==>Ú  219==>Û
220==>Ü  221==>Ý  222==>Þ  223==>ß
224==>à  225==>á  226==>â  227==>ã
```

```
228==>d  229==>e  230==>f  231==>g
232==>h  233==>i  234==>j  235==>k
236==>l  237==>m  238==>n  239==>o
240==>p  241==>q  242==>r  243==>s
244==>t  245==>u  246==>v  247==>w
248==>x  249==>y  250==>z  251==>{
252==>|  253==>}  254==>~  255==
```

TESTING YOUR CHARACTER

Our next program is a reaction tester like the one you experienced earlier. However, you are not just being tested on speed. In this program, you have to try and find the *right key* on the keyboard as quickly as possible.

A letter will appear in the middle of the screen. As quickly as you can, find that letter on the keyboard and press it. You will be told how long it took you, and this time compared with your best time. Notice how the letter which is printed on the screen uses CHR$ in line 100 (find the character of the number, chosen at random in line 70, represented by the variable A), and A$ is compared with CHR$ A in line 120 to see if you were correct.

```
   10 REM Keyboard Instructor
   20 REM
   40 BEST=2000
   50 MODE 7
   60 VDU 23;8202;0;0;0;
   70 A=64+RND(26)
   80 B=0
   90 CLS
  100 PRINT TAB(7,3)CHR$(136);CHR$(A);CH
R$(137)
  110 A$=INKEY$(0)
  120 IF A$=CHR$(A) VDU 7 : GOTO 180
```

```
130 B=B+1
140 PRINT TAB(17,3)B
150 IF B<2000 GOTO 110
160 PRINT ''"Sorry,Time's up !.."
170 GOTO 200
180 PRINT ''"Well Done... You scored "
;B
190 PRINT "On that one..."
200 IF B<BEST THEN BEST=B
210 PRINT ''"So the best so far is ";B
EST
220 FOR T=1 TO 2000
230 NEXT
240 GOTO 70
```

CUTTING THEM UP

One of the really great aspects of your computer is the way it can be used to manipulate strings. It is called 'string slicing' and works as follows. The information which follows the string determines which parts of the string will be printed. For example, if the string was "TRIUMPH" telling the computer to PRINT MID$("TRIUMPH",1,1) would get it to print "T" since T is at position 1 and is one character long. Similarly, PRINT MID$("TRIUMPH",3,4) would get the computer to print up a "IUMP", since this is the third letter of the word and goes on for four letters.

When the keyword MID$ is used — the string to be sliced is placed in brackets with some information to tell the computer how to slice the string, MID$ (TEST STRING, start character, number of characters). The first number tells the computer which character to start printing from, and the second number tells it how many characters to print out.

There are two other 'string slicing' commands — they are RIGHT$ and LEFT$. They too have the string to be sliced in brackets, but with one number only:

LEFT$ (TEST STRING, number of characters)

RIGHT$ (TEST STRING, number of characters)

In each case the keywords take the specified number of characters from the LEFT or RIGHT of the string;

For example LEFT$ ("Ustkamenogorsk",7) = "Ustkame"
RIGHT$ ("Ustkamenogorsk",7) = "nogorsk"

The next program should help make this clear. The string variable A$ is set equal to "FLOCCILATION" in line 10, and various portions of this are selected by the lines of the program. Run it as it is, to see if you can follow through what is happening in each line:

```
   10 A$="FLOCCILLATION"
   20 PRINT A$;"-->RIGHT$(A$,10):";RIGHT
$(A$,10);
   30 PRINT A$;"-->RIGHT$(A$,6):";RIGHT$
(A$,6)
   40 PRINT A$;"-->RIGHT$(A$,3):";RIGHT$
(A$,3)
   50 PRINT A$;"-->LEFT$(A$,4):";LEFT$(A
$,4)
   60 PRINT A$;"-->LEFT$(A$,6):";LEFT$(A
$,6)
   70 PRINT A$;"-->LEFT$(A$,9):";LEFT$(A
$,9)
   80 PRINT A$;"-->MID$(A$,3,4):";MID$(A
$,3,4)
   90 PRINT A$;"-->MID$(A$,5,5):";MID$(A
$,5,5)
  100 PRINT A$;"-->MID$(A$,6,5):";MID$(A
$,6,5)
```

You'll see this when you RUN the program

```
FLOCCILLATION-->RIGHT$(A$,10):CCILLATION
FLOCCILLATION-->RIGHT$(A$,6):LATION
FLOCCILLATION-->RIGHT$(A$,3):ION
```

```
FLOCCILLATION--->LEFT$(A$,4):FLOC
FLOCCILLATION--->LEFT$(A$,6):FLOCCI
FLOCCILLATION--->LEFT$(A$,9):FLOCCILLA
FLOCCILLATION-->MID$(A$,3,4):OCCI
FLOCCILLATION--->MID$(A$,5,5):CILLA
FLOCCILLATION--->MID$(A$,6,5):ILLAT
```

Now change A$ in line 10 to your name, or another word of your choice, and run the program again.

PUTTING THEM BACK TOGETHER

Strings can be added together on your computer. The process of adding strings is called the frightening-looking word *concatenation*. You can *concatenate* two complete strings together, or just add bits of them, as our next program shows:

```
 10 A$="FLOCCILLATION"
 20 B$="SITOPHOBIA"
 30 C$=A$+B$
 40 PRINT "A$--->";A$
 50 PRINT "B$--->";B$
 60 PRINT "C$--->";C$
 70 D=RND(15)
 80 E=RND(10)
 90 D$=MID$(C$,D,E)
100 PRINT "D$--->";D$
110 E$=MID$(A$,E,D)
120 PRINT "E$--->";E$
125 F=RND(D+E)
130 F$=LEFT$(E$+D$,F)
140 PRINT "F$--->";F$
```

When you run this program, which creates C$ in line 60 by concatenating A$ and B$, you'll see results like these two:

When you run this program, which creates strings by concatenating A$ and B$, you see results like these:

```
A$-->FLOCCILLATION
B$-->SITOPHOBIA
C$-->FLOCCILLATIONSITOPHOBIA
D$-->ITOPHOBIA
E$-->TION
F$-->T

A$-->FLOCCILLATION
B$-->SITOPHOBIA
C$-->FLOCCILLATIONSITOPHOBIA
D$-->LATIONSITO
E$-->TION
F$-->TION

A$-->FLOCCILLATION
B$-->SITOPHOBIA
C$-->FLOCCILLATIONSITOPHOBIA
D$-->IONSITOPH
E$-->ATION
F$-->ATIONIONSITOPH
```

PLAYING AROUND

You can do a number of things with string slicing, as our next program demonstrates. NAME PYRAMID allows you to enter your name to produce an interesting display. Once you've seen the program running, you'll understand why the program has been given the name it has.

```
   10 REM Name Pyramid
   20 REM Using String Slicing..
   30 CLS
   40 INPUT "Type in your name please.."
" A$
   50 IF LEN(A$)>21 A$=LEFT$(A$,20)
```

```
 70 A=LEN(A$)
 80 PRINT TAB(19,3)"^^^"
 90 FOR G=1 TO A
100 PRINT TAB(20-G);
110 FOR H=1 TO 2*G
120 PRINT MID$(A$,G,1);
130 NEXT
140 PRINT
150 NEXT
```

```
                    AA
                   LLLL
                  EEEEEE
                 XXXXXXXX
                AAAAAAAAAA
               NNNNNNNNNNNN
              DDDDDDDDDDDDDD
             RRRRRRRRRRRRRRRR
            EEEEEEEEEEEEEEEEEE

          GGGGGGGGGGGGGGGGGGGGGG
         OOOOOOOOOOOOOOOOOOOOOOOO
        LLLLLLLLLLLLLLLLLLLLLLLLLL
       LLLLLLLLLLLLLLLLLLLLLLLLLLLL
      NNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
     EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
    RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR

IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
```

PLAYING IT BACK

Our final program in this chapter shows one very effective use of string slicing, in which a string is progressively reduced by one element. When you run ECHO GULCH, you'll see a letter appear on the screen. It will then vanish. *Once it has vanished*

```
230 GOTO 90
240 PRINT "You Scored ";S
250 END
260 PRINT "*****"Well Done!!! You're th
Champion!!"
```

The variable S, which holds your score, is set to zero in line 70, and line 80 sets the string variable A$ to a long line of letters. If you look at line 90, you'll see a complicated looking statement beginning with INT, and including RND. This chooses a random character from the string so you will get different letters each time you run it.

Line 110 prints C$, as the random letter from the string, on the screen, and line 120 inserts a short delay loop, which uses the LEN function. This is another string function, and returns the *length* of a string, that is, the number of characters which make it up. LEN does not make any distinction between letters, numbers, symbols or spaces, as you'll discover if you enter a number of PRINT LEN (A$) statements, after setting A$ to equal various words, symbols and sentences. Because A$ is reduced by one character by line 190 each time the program cycles, LEN A$ is a smaller number each time, so the delay produced by line 35 (which dictates how long the character will be on the screen before it vanishes) becomes shorters.

INKEY$

Line 40 uses INKEY$ to *read* the keyboard. INKEY$, as you know does not demand that you press RETURN after touching a key. INKEY$ always returns the key you have pressed as a *string*.

Line 160 looks at B$, the variable which is set equal to whichever key is being pressed. As you can see in line 160, you can use the greater than (>) and less than (<) symbols, which

(and not before) you will have a limited amount of time in which to press the key yourself.

If you have pressed the right key, a short tune will sound, and the letter will be replaced with a new one. This will stay on the screen for a shorter time. Each time a new letter appears, you will be given less time to see it, before you'll have to type it into the computer. If you make a mistake, the "SORRY, THAT'S WRONG" message will appear, along with your score. If you manage to get all of a list of letters right, you'll be rewarded with a "YOU'RE THE CHAMP!!" message. Here's the listing:

```
   10 REM Echo Gulch
   20 REM
   60 REM
   70 S=0
   80 A$="LIUHJKDGKCKUYTBKUYBJUYTRMJHGJ
FGYDFJDFKU"
   90 C$=MID$(A$,RND(LEN(A$)),1)
  100 CLS
  110 PRINT TAB(18,5) C$
  120 FOR R=1 TO 100*LEN(A$):NEXT
  130 CLS
  140 *FX 15
  150 B$=INKEY$(0)
  160 IF B$<"A" OR B$>"Z" THEN GOTO 150
  170 PRINT TAB(18,5) B$
  180 IF B$=C$ THEN SOUND 1,-15,LEN(A$)
5,3 : ELSE GOTO 220
  190 A$=MID$(A$,2,43)
  200 IF LEN(A$)=1 THEN 260
  210 S=S+1
  220 PRINT '''''"Oops...You made a mis
 ake..."
```

we discussed in chapter eight, in connection with strings. These look at all elements of a string, and compare them in terms of alphabetical order (so "ZEBRA" is *less than* "AARDVARK", and "BEAST" is *greater than* "BEAUTY"). You can compare strings using equals (=) and not equals (<>) as is shown in the next few lines of the program. Line 180 compares the key you've pressed (B$) with C$, and if they are the same, continues through the multi-statement line to play the sound.

Line 190 strips the string A$ of its fist character. Line 200 checks to see if the length of A$ equals one (that is, IF LEN(A$)=1) and if it finds that it is, goes to line 260 to print out the "YOU'RE THE CHAMP!!" message. If not, line 210 adds one to variable S, your score. The program then cycles back to line 90.

CHAPTER ELEVEN
READING DATA

In this chapter we'll be looking at three very useful additions to your programming vocabulary: READ, DATA and RESTORE. They are used to get information *stored* in *one part* of the program to *another part* where it can be used.

Enter and run this program, which should make this a little clearer:

```
10 REM READ, DATA
20 DIM A(5)
30 FOR I=1 TO 5
50 READ A(I)
60 PRINT A(I)
70 NEXT
80 DATA 3.14159,.26535,.89793,.23846,.264
33
```

Using line 50, the program READS through the DATA statement in line 80 in order, printing up each item of DATA (with line 60).

RESTORE moves the computer back to the *first* item of DATA in the program, as you'll discover if you modify the above program by adding line 65, so it reads as follows:

```
10 REM READ, DATA, RESTORE
20 DIM A(5)
```

```
30 FOR B=1 TO 5
50 READ A(B)
60 PRINT A(B)
65 IF B>3 THEN RESTORE
70 NEXT
80 DATA 3.14159,26535,89793,23846,26433
```

It does *not* matter where in the program the DATA is stored. The computer will seek it out, in order from the first item of DATA in the program to the last, as our next program (which scatters the DATA about in an alarming way) convincingly demonstrates:

```
 5 DATA 42
10 REM READ,DATA
15 DATA 3.14159
20 DIM A(5)
25 DATA 26535
30 FOR B=1 TO 5
35 DATA 89793
50 READ A(B)
55 DATA 23846
60 PRINT A(B)
70 NEXT
```

READ and DATA work just as well with string information. You can mix numeric and string DATA within the same program, so long as you take care to ensure that when the program wants a numeric item that a number comes next in the program, and when it wants a string item, it finds it:

```
10 REM READ,DATA
20 DIM A$(5),A(5)
30 FOR B=1 TO 5
50 READ A$(B),A(B)
60 PRINT A$(B)TAB(30),A(B)
```

```
70 NEXT
80 DATA "Ovoviviparous",4495,"Stamini
ferous",082883,"Phenanthrenequinone",215
13,"Trifluorochloromethane",1513,"Heter
ophyllous",51.502
```

You can also use variables in DATA statements but as you will see when you run this next program strings need not be in quote marks. This means that only numeric variables can be included in the DATA list as Ovovivip$ will be read as "Ovovivip$" and not as the string assigned in line 20.

```
20 Ovovivip$="Floccinaucinihilipilifi
catrix"
30 CORF=5566
40 DIM A$(5),A(5)
50 FOR I=1 TO 5
60 READ A$(I),A(I)
70 PRINT A$(I)TAB(30),A(I)
80 NEXT
90 DATA Ovovivip$,4495, Staminiferous
,082883, Phenanthrenequinone ,CORF, Tri
fluorochloromethane ,1513, Heterophyllou
s ,51.502
```

In our next program, GALACTIC GROCER, written by Tim Rogers, the goodies you are trading with start off their lives in DATA statements in lines 1870 to 1960. The relative values of these strange objects are stored in the DATA statements which follow.

In this program you are the Galactic Grocer, trading such interstellar goodies as coal dust, reject soap, aspidistras and zinc hub-caps. As you'll discover, these items are in great demand throughout the galaxy. You start out with some

money, and a stock of goods, provided by Galactic Grocery Central, and you are attempting to zap around space, trading here and bartering there, to return to Central with more goods. On arrival, the goods in your hold will be sold for twice the amount you paid for them, and a tenth of the profits will go to you. The program is set over some 20 years, compressed into a quarter of an hour or so by the computer time continuum.

Out there, in the cut and thrust of interstellar commerce, you'll meet other hardnosed Grocers from other firms, planets and star systems. They'll be on their way to other points in the Known Universe, and will be willing to trade with you. Keep in mind — when considering purchasing anything from wily traders — that the point of the game, so far as you're concerned, is to try and stockpile the really pricey merchandise. If you don't want to buy from, or sell to, a particular trader, then just press RETURN (that is, you do not have to press Ø, then RETURN).

```
 10 REM Galactic Grocer
 20 REM
 60 REM
 70
 80 MODE 7
 90
100 PROCinit
110
120 PROCassign
130 REPEAT
140
150 REPEAT
160 PROCloop
170 SOUND 0,-15,2,6
180 UNTIL W=2120
```

```
  190 PROCend
  200
  210 UNTIL 0
  220
  230 DEF PROCloop
  240 VDU 24,30
  250 PRINTrl$;nb$;dl$;bl$;"     GALACTIC
GROCER"
  260 PRINTrl$;nb$;dl$;bl$;"     GALACTIC
GROCER"
  270 PRINT"              ";cl$;"Money:£";M
  280 REM Check Value Of Goods
  290 PROCji
  300 PRINT rl$;" Value of goods:£";J
  310 REM Print Out List Of Goods
  320 FOR A=1 TO 20 STEP 2
  330 PRINT y1$0(A);" ";C$(A) TAB(18),0(
(1);" ";C$(A+1)
  340 NEXT A
  350 VDU 28,0,24,39,15
  360 B$="buy"
  370 PROCbarter
  380 CLS
  390 B$="sell"
  400 PROCbarter
  410 REM Increment Year counter
  420 W=W+1:VDU 7
  430 ENDPROC
  440
  450
  460 DEF PROCbarter
  470 N=RND(4)+1
  480 A=0
  490 REM Make Traders wish to sell when
          player wishes to buy and vice
```

```
          versa....
  500 S$="buy"
  510 IF B$="buy" THEN S$="sell"
  520 CLS
  530 PRINT "There are "¦N¦" traders wis
hing to "¦S$¦'"some goods"
  540 DUMMY=GET
  550 REM Set F to let Player choose
          what he wants to sell/buy 1/3
          of the times he has dealings.
  560 F=RND(3)
  570 A=A+1
  580 REM Set X to random position for
          goods from the C$ array...
  590 X=RND(21)
  600 IF F=1 THEN PROCtrader
  610 IF X=0 THEN 860
  620 REM Set Limits of No. of Units
  630 Y=RND(10000)
  640 P=RND(Y)
  650 REM Set Cost of each Unit of Goods
  660 Z=RND(100)
  670 PRINT''
  680 Z=INT(Z*P(X))/100
  690 PRINT "Trader "¦A¦" wishes to "¦S¦
" "¦
  700 PRINT P¦" to "¦Y¦" "¦C$(X)¦" at £'
¦Z
  710 PRINT "each..."
  720 REM Inform Player of Resources..
  730 PRINT''"(You have £"¦M¦")"
  740 PRINT "(You have "¦O(X)¦" "¦C$(X)¦
")"
  750 PRINT''"How many to "¦B$¦
  760 INPUT U$
```

```
  770 REM Calculate New Values changed
          by preceding deal....
  780 U=VAL(U$)
  790 IF U=0 THEN 840
  800 IF U<P OR U>Y THEN 700
  810 IF G$="sell" THEN U=-U
  820 IF U<-O(X) THEN 750
  830 IF M-U*Z<=Q THEN 730
  840 M=M-U*Z
  850 O(X)=O(X)+U
  860 NEXT
  870 IF A=N THEN SOUND 1,-15,10,6:FOR H
=1 TO 1500:NEXT:ELSE GOTO 570
  880 ENDPROC
  890
  900
  910 DEF PROCtrader
  920 PRINT "trader ";A;" says:-'What do
 you want""to ";G$;"?"
  930 REM Ask Player to choose goods to
          buy/sell....
  940 INPUT Z$
  950 IF Z$="" THEN ENDPROC
  960 REM Search for goods chosen in
          goods list....
  970 B=0
  980 X=0
  990 REPEAT
 1000 B=B+1
 1010 UNTIL  Z$=C$(B) OR B>20
 1020 IF B>20 THEN 920
 1030 IF RND(1)<.8 THEN X=B:ENDPROC
 1040 PRINT "Sorry trader ";A;" says:-'s
orry I do""not want to ";G$;" any "C$(B
)
```

```
1050 REPEAT
1060 X=RND(21)
1070 UNTIL X<>3
1080 DUMMY=GET
1090 ENDPROC
1100
1110
1120
1130
1140 DEF PROCassign
1150 RESTORE 1870
1160 REM Assign lists of Data via READ
       and set up 'renewable' vari-
       ables.....
1170 REM Put list of goods into C$(
1180 FOR A=1 TO 21
1190 READ C$(A)
1200 NEXT
1210 REM Put list of multiples into M(
1220 FOR A=1 TO 21
1230 IF A<6 THEN READ M(A) :NEXT
1240 NEXT
1250 FOR A=1 TO 21
1260 REM Set 35% O() to RND(10000)....
1270 READ F(A)
1280 IF RND(1)>.65 THEN O(A)=RND(10000)
1290 NEXT A
1300 W=2100
1310 M=60000+RND(10000)
1320 OM=M
1330 PROCji
1340 J1=J
1350 ENDPROC
1360
1370
```

```
1380 DEF PROCji
1390 REM Work out Value of Goods and
     add up money...
1400 J=0
1410 FOR L=1 TO 21
1420 J=J+O(L)*P(L)*2
1430 NEXT
1440 J=J+M
1450 ENDPROC
1460
1470
1480 DEF PROCend
1490 SOUND 2,-15,250,6
1500 REM Work out Profit and give
     the option to restart game...
1510 PROCji
1520 J2=J
1530 PRINT "You have arrived in port.."
1540 PRINT
1550 Z=J1
1560 PRINT "You started out with £";OM
1570 PRINT "You now have £";M
1580 PRINT
1590 V=INT((M-OM)/OM*100)
1600 PRINT "You made ";V;"%profit"
1610 IF I<V THEN I=V
1620 PRINT "Best so far is ";I;"%"
1630 PRINT
1640 PRINT "Any key to re-run except ES
CAPE"
1650 DUMMY=GET
1660 W=2100
1670 ENDPROC
1680
1690
```

```
1700 DEF PROCinit
1710 REM Initialise all arrays which
         need not be initialised when
         the program is re-run....
1720 I=0
1730 DIM C$(21),D(21),L(21),M(21),P(21)
1740 REM Turn Cursor off....
1750 VDU 23;8202;0;0;0;
1760 @%=4
1770 bl$=CHR$ 132
1780 rl$=CHR$ 129
1790 dl$=CHR$ 141
1800 yl$=CHR$ 131
1810 cl$=CHR$ 134
1820 nb$=CHR$ 157
1830 W=2100
1840 ENDPROC
1850
1860
1870 DATA "tons dust","kg grain"
1880 DATA "cwt peel","TV-sets"
1890 DATA "gal ink","fish"
1900 DATA "bars soap","phasers"
1910 DATA "test-tubes","ex-tyrants"
1920 DATA "aspidistras","paper bags"
1930 DATA "match heads","chipped mugs"
1940 DATA "missiles","badges","noises"
1950 DATA "zinc hub-caps","hittiles"
1960 DATA "toenails","pen-tops"
1970
1980 DATA 6,9,4,9,6
1990 DATA 6,4,5,30,9,.3,6,10,3,10,6
2000 DATA 9,1,5,10,.67,7,9,12,10,.9
2010 DATA 10,10,3,30,.3,.1,15,1,250
2020 DATA 3,1000,4,.1,5,5.5,.2,0.10
```

```
2030 DATA 3,23
2040
2050
>
>
```

CHAPTER TWELVE
ENHANCING PROGRAMS

You have a colour computer on your hands, and you should make the most of it. We'll start our investigation of ways of improving your programs by referring back to the FAST FOOD program which was first introduced in chapter four.

We have been using colour statements in many of the programs you've encountered so far in this book, so you may well have a pretty good idea of how to use the facilities.

Here's a version of FAST FOOD which chooses colours randomly before the PRINT statements:

```
   10 REM FAST FOOD
   20 MODE 7
   30 A=RND(4)
   40 PRINT "YOU'VE ORDERED"
   45 PRINT CHR$(128+RND(6));
   50 IF A=1 THEN PRINT "A HAMBURGER WIT
H ALL THE TRIMMINGS"
   60 IF A=2 THEN PRINT "A LARGE PORTION
 OF STEAMING FRENCH FRIES"
   70 IF A=3 THEN PRINT "A HUGE PLATEFUL
 OF FRESHLY COOKED RIBS"
   80 IF A=4 THEN PRINT "A PAIR OF 100%
MEAT HOT DOGS WITH SAUCE"
   90 PRINT
```

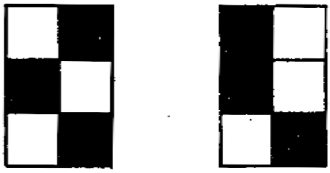
```
 95 DUMMY=GET
100 GOTO 30
```

As you can see this program runs in MODE 7. MODE 7 is the odd one out of all the modes, in that it has a different method of selecting colours. To change colour in MODE 7 a control code needs to be put onto the screen — this takes up the space of a character on the screen, which determines the output on the rest of the line. This means that if you want red letters on the screen you find out which character produces 'red alphanumerics', CHR$(129). If you print that character on the screen anything after it on that line is printed in red. Here is a list of MODE 7

	Alphanumerics	Graphics	
Red	CHR$(129)	CHR$(145)	
Green	CHR$(130)	CHR$(146)	
Yellow	CHR$(131)	CHR$(147)	
Blue	CHR$(132)	CHR$(148)	
Magenta	CHR$(133)	CHR$(149)	
Cyan	CHR$(134)	CHR$(150)	
White	CHR$(135)	CHR$(151)	
Flashing Characters		CHR$(136)	CHR$(137)
Double Height Characters		CHR$(141)	CHR$(140)
Separated Graphics		CHR$(154)	CHR$(153)
Background Colour		CHR$(157)*	CHR$(156)

* precede code with alphanumeric colour code

When preceded by a graphics code from 145 to 151 the punctuation of the teletex mode character set produces graphics. These graphics are made up of the combinations of a grid of six blocks like these:



'Separated Graphics' means that the graphics will look like this:

COLOUR

You have probably worked out how to use some of the BBC micro's colour facilities by looking at the manual and the programs in this book so far. Here is how it works:

> COLOUR X selects a colour — in which subsequent printing will be done. If the number is greater than 127 the colour of the background is changed. This means, in MODE 1 for example COLOUR 1 would select red and COLOUR 3 would select white.
>
> Ø Black
> 1 Red
> 2 Yellow
> 3 White
>
> COLOUR 4 would be black and 5 would be red and so on until colour 127.
>
> 128 Black background
> 129 Red background
> 13Ø Yellow background
> 131 White background
>
> so to get the appropriate background colour the syntax is COLOUR 128 + text colour.

MODE 1 has only four colours, but as you know you have 8 colours available on the BBC micro (and 8 flashing colours). The colours in MODE 1 are actual colours Ø, 1, 3 and 7. 'Actual' colours are the numbers the computer gives to each of the 16 colour variations. It would be confusing if the MODE 1

colours were called 0, 1, 3 and 7 so for each MODE the colours are re-numbered and are known as "logical colours".

Here's a version of the FAST FOOD program in MODE 1, using the COLOUR command to produce random colours for text and background colours.

```
    10 REM FAST FOOD
    20 MODE 1
    30 A=RND(4)
    40 PRINT "YOU'VE ORDERED"
    42 B=RND(4)-1
    44 COLOUR 135-B
    46 COLOUR B
    50 IF A=1 THEN PRINT "A HAMBURGER WIT
H ALL THE TRIMMINGS"
    60 IF A=2 THEN PRINT "A LARGE PORTION
 OF STEAMING FRENCH FRIES"
    70 IF A=3 THEN PRINT "A HUGE PLATEFUL
 OF FRESHLY COOKED RIBS"
    80 IF A=4 THEN PRINT "A PAIR OF 100%
MEAT HOT DOGS WITH SAUCE"
    90 PRINT
    95 DUMMY=GET
   100 GOTO 30
```

But the colours that are available in MODE 1 may not be to your liking. You can change the logical colour to any actual colour you want using the VDU 19 command:

VDU 19, logical colour, actual colour, 0,0,0.

This next FAST FOOD program makes use of this command:

```
10 REM FAST FOOD
20 MODE 1
30 A=RND(4)
```

```
   40 PRINT "YOU'VE ORDERED"
   42 A=RND(3)-1
   44 VDU 19,0,7-B,0,0,0,0,0
   46 VDU 19,3,B,0,0,0,0,0
   50 IF A=1 THEN PRINT "A HAMBURGER WIT
H ALL THE TRIMMINGS"
   60 IF A=2 THEN PRINT "A LARGE PORTION
 OF STEAMING FRENCH FRIES"
   70 IF A=3 THEN PRINT "A HUGE PLATEFUL
 OF FRESHLY COOKED RIBS"
   80 IF A=4 THEN PRINT "A PAIR OF 100%
MEAT HOT DOGS WITH SAUCE"
   90 PRINT
   95 DUMMY=GET
  100 GOTO 30
```

Note that in the above program, the logical colour of the text was never changed — throughout the program the text was printed in logical colour 3 on logical colour Ø — the computer makes the text and screen *look* like different colours. This means if it is printing on white and the colour is redefined to be red not only the subsequent text will be red, the previously printed text also changes.

Though the computer works very fast, it cannot do animation very well, as you can see the things being drawn or plotted. But if you draw the thing you want to animate on the screen in the 16 different colours like this:

and if the colours were all previously defined to be black, nothing would be seen on a black screen. Then if using a

FOR/NEXT loop the 16 colours are redefined to be white in colour order, the ball will be seen to go round in a circle.

When the computer is composing a colour — from black through to white it mixes three colours: red, green and blue. There are eight combinations following a logical structure:

	blue	green	red		
Ø	Off	Off	Off	=	Black
1	Off	Off	On	=	Red
2	Off	On	Off	=	Green
3	Off	On	On	=	Yellow (red + green)
4	On	Off	Off	=	Blue
5	On	Off	On	=	Magenta (blue + red)
6	On	On	Off	=	Cyan (blue + green)
7	On	On	On	=	White

So when a colour is flashed, what actually happens is that the colours will switch on if they are off, and will switch off if they are on: the opposite of red (where only red was on) will be the red off and the other two colours which gives cyan. You will find that this is less complicated in practice than you think. These combinations of opposites are pleasing to the eye and are good choices for display colours (red background/cyan foreground in this case).

Now, to explain the use of things like colour, we are going to take a simple program, and gradually elaborate it, showing how adding such things as flashes and user-defined graphics can add a great deal of interest to your programs. At the end of this section, we'll give you a couple of suggestions to apply if you wish to keep improving and elaborating the program.

The program we're going to use as the core of our development work is a standard 'Duck Shoot' program, in which little objects fly across the screen, and you have to try and shoot them down. In the first version of the program, the little objects are letters chosen at random, and you are the letter "X". You

fire at the "ducks" by pressing the "F" key, and you move yourself left using the "Z", and right using the "X" keys (you move in the direction of the arrows on those keys).

Although there is no time limit on this program, there is a limit on the number of shots you can fire. In all of the versions of this game in this part of the book you'll see (line 90) the program starts with a limit of 15 shots. The number of shots is deliberately kept low so you will not be able to get a high score just by leaving your finger on the "F" key, and waiting for the ducks to fly into the line of fire.

Here, then is the first program. Type it into your computer, and type RUN then press RETURN, to get it underway.

```
  10 REM DUCK SHOOT
  20 REM
  60 REM
  70 CLS
  80 SCORE=0
  90 SHOTS=15
 100 A$="K  LKUGI IUV QI TU    II JHT
   H   GF"
 110 ACROSS=15
 120 DOWN=14
 130 PRINT TAB(0,7);A$
 140 PRINT TAB(ACROSS-2,DOWN);" X "
 150 IF (INKEY(-68)) THEN SHOTS=SHOTS-1
:IF MID$(A$,ACROSS,1)<>" " THEN SCORE=SC
ORE+57:A$=LEFT$(A$,ACROSS-1)+" "+RIGHT$(
A$,40-ACROSS)
 160 PRINT TAB(0,0);"SCORE:";SCORE," SH
OTS LEFT:";SHOTS;"    "
 170 IF SHOTS<1 THEN PRINT TAB(0,10)"
 THAT'S THE END OF THE GAME  ":END
```

```
180 ACROSS=ACROSS+(INKEY(-98))-(INKEY(
-67))
190 A$=RIGHT$(A$,39)+LEFT$(A$,1)
200 FOR WAIT=0 TO 120:NEXT
210 GOTO 130
```

You'll see the letters which are held in A$ (see line 100) moving across near the top of the screen. You (the "X") will be about the middle of the screen when the program starts. You can — as I mentioned a few moments ago — move yourself back and forth using the "Z" and "X" keys to get yourself into the position which you think gives you the best possible chance. When you judge a "duck" is directly overhead, press the "F" key to fire your patented anti-duck missile. The number after the words SHOTS LEFT (in the top right hand corner of the screen) will decrease, and if you have been accurate, the number after the word SCORE in the top left hand corner of the screen will increase.

Note, by the way, that we have deliberately used *explicit names* for the variables within this program. That is, the variable name for the scores is SCORE, for shots left it is SHOTS, for your position down the screen DOWN, and for your position across the screen, the variable name is ACROSS. Even though it takes a little longer to type long variable names into a program and (of course) they use up more memory than do shorter names, running out of the memory is rarely a problem on your computer, and the advantages of using explicit names to keep the purpose of various parts of the listing clear outweighs the extra time it takes to type them in. If, for example, you were writing a program like this and you decided that it would be better if the "X" was printed slightly further down the screen, you would not have to search through the program to work out which variable held your "down" co-ordinate. If you had used the explicit names as in this case, you would find it very easy to locate the variable you were looking for.

Run the DUCK SHOOT program a few times, then return to the book for the first part of our discussion on it.

Line 100 defines the string variable A$ as a long series of letters and spaces. The letters can be anything you like; don't feel you need to copy ours. The important thing, however, is that the string is 40 characters long. You can check this by running the program briefly, stopping it with ESCAPE, then typing in as a direct command, PRINT LEN(A$). If your string is the correct length, PRINT LEN A$, followed by ENTER will give you the answer 40.

The appearance of movement given to the ducks is created by use of BBC BASIC's string-handling commands which were explained a little earlier in the book. Refer back to this section now if you need to remind yourself of how they work. The vital line for the movement is line 190, which resets A$ equal to all of the string without its first character, that is, RIGHT$(A$,39), and then adds to the very end of it the character of the string which was at the beginning, LEFT$(A$,1). The string is reprinted, over and over again, as the program runs (by line 130) in the same position, at 0,7 (eight lines down, and starting hard in the left hand margin), and because the string is in effect being 'shifted along' one character at a time before it is reprinted, the elements in the string appear to move smoothly along. Using strings in this way is one of the simplest ways there is on your computer to create smoothly moving graphics.

Look at line 150. When the computer comes across an IF/THEN statement — as you know — it checks to see if it is true. If it finds that it is not true, then it moves along to the next line in the program, without bothering to carry out any further instructions which may be on the same line. If the computer finds, at the start of line 150, that INKEY does not equal "F" (that is, you are not pressing the "F" key) then it proceeds to line 160, missing all the information and instructions which follow the IF INKEY (−68) line. If, however, you are pressing "F" when the computer gets to line 150, it continues working

through the line, and decrements the variable SHOTS by one. Then it hits another IF/THEN condition, which makes use of the ability of the BBC BASIC to isolate an element of a string instantly. It looks at MID$(A$,ACROSS,1). The "X" which is you is printed at ACROSS (actually as you see in line 140, a three-element string, with a space either side of the "X" is printed at ACROSS *minus two,* which has the effect of printing the "X" at the position referred to by the variable ACROSS, so MID$(A$,ACROSS,1) lies directly above you.

If line 150 discovers that MID$(A$,ACROSS,1) is anything but a space, you have hit a duck, so the computer continues working through the line. The variable SCORE is incremented by 57 and, finally in line 150, that element of A$ is set to a blank, so the "duck" disappears.

Now all this takes some time to explain, but you'll find the computer does it apparently instantaneously. You press "F" the score increases by 57 (if you're a good shot), the number of shots left drops by one, and the duck disappears. You'll see (line 170) that the program stops when you run out of shots. Take note of your score at this point, and see if you can beat it in subsequent runs.

Once you have the program running to your satisfaction, and you have a pretty good idea of how it works, modify it to read like the following program. You do not have to NEW the computer. Just compare the program you have in your computer, line by line, with the next listing and make any changes you need to, by adding complete new lines at 75, 125, 131 and 155 and modifying certain other lines.

```
10 REM DUCK SHOOT V1
20 REM
60 REM
70 MODE 1
75 VDU 23;8202;0;0;0;
90 SCORE=0
90 SHOTS=15
```

```
  100 A$="K  LKUGI IUV OI TU     JI JHT
    H   OF"
  110 ACROSS=15
  120 DOWN=14
  125 COLOUR 2
  130 PRINT TAB(0,7);A$
  131 COLOUR 3
  140 PRINT TAB(ACROSS-2,DOWN);" X "
  150 IF (INKEY(-68)) THEN SHOTS=SHOTS-1
:IF MID$(A$,ACROSS,1)<>" " THEN SCORE=SC
ORE+57:A$=LEFT$(A$,ACROSS-1)+" "+RIGHT$(
A$,40-ACROSS)
  155 COLOUR 1
  160 PRINT TAB(0,0);"SCORE:";SCORE," SH
OTS LEFT:";SHOTS;"    "
  170 IF SHOTS<1 THEN PRINT TAB(0,10)"
 THAT'S THE END OF THE GAME  ":END
  180 ACROSS=ACROSS+(INKEY(-98))-(INKEY(
-67))
  190 A$=RIGHT$(A$,39)+LEFT$(A$,1)
  200 FOR WAIT=0 TO 120:NEXT
  210 GOTO 125
```

Now when you run this, you'll see an immediate and quite striking improvement. Colour certainly adds a lot to any program on this computer.

Even if we had done nothing else, there would be a significant improvement in the program. You'd have the score and shots left, ducks and the "X" all printed in different colours, which is far more interesting than just plain old black and white.

You are probably aware that, in moving graphics programs, everything you get the computer to do — from making an IF/THEN decision, adding two numbers together, raising one to the power of the other, to generating a random number —

takes time, and the more you get the computer to do before each subsequent frame of a moving graphics program is printed, the more slowly the graphics will move, and the more jerky they will appear.

Apart from the colour changes we've discussed, the program is the same as the first listing. However, you can see that the few changes we have introduced have improved it considerably. We'll now continue with the improvements, and eventually we will add some sound, and getting the screen to flash when a duck is shot.

Enter this next version of the program, run it to decimate a few flocks of ducks, then return to your book for a discussion.

```
10 REM DUCK SHOOT V3
20 REM
30 REM
40
50 MODE 1
60 PROCinitialise
100 REPEAT
110 PROCloop
120 UNTIL 0
130
140
150 DEF PROCloop
160 COLOUR 2
170 PRINT TAB(0,7);A$
180 COLOUR 3
190 PRINT TAB(ACROSS-2,DOWN);" X "
200 IF (INKEY(-68)) PROCfire
210 COLOUR 1
220 PRINT TAB(0,0);"SCORE: ";SCORE," SH
OTS LEFT: ";SHOTS;"    "
230 IF SHOTS<1 THEN PROCend
```

```
  240 ACROSS=ACROSS+(INKEY(-98))-(INKEY(
-67))
  250 A$=RIGHT$(A$,39)+LEFT$(A$,1)
  260 FOR WAIT=0 TO 50:NEXT
  270 ENDPROC
  280
  290
  300 DEF PROCfire
  310 SHOTS=SHOTS-1
  320 NEWA$=LEFT$(A$,ACROSS-1)+" "+RIGHT
$(A$,40-ACROSS)
  330 IF MID$(A$,ACROSS,1)<>" " THEN SCO
RE=SCORE+57:A$=NEWA$
  340 ENDPROC
  350
  360
  370 DEF PROCend
  380 PRINT TAB(0,10) "    THAT'S THE END
 OF THE GAME   "
  390 *FX 15 0
  400 END
  410
  420
  430 DEF PROCinitialise
  440 SCORE=0
  450 SHOTS=15
  460 A$="A    AAA AAAA AA    AAAA     A  A
A    AAA A"
  470 ACROSS=15
  480 DOWN=14
  490 ENDPROC
```

That version of DUCKSHOOT was operationally exactly the same. The difference is the way the program is executed. It has been partly structured in that some procedures have been

defined. They have been created in order to delete the multistatement IF... lines by having these statements separate in a PROC, only called IF the condition is satisfied. Also an infinite REPEAT/UNTIL LOOP has been added at lines 100 and 120.

In the next version of DUCK SHOOT the program has been divided into workable procedures.

```
   10 REM DUCK SHOOT V4
   20 REM
   60 REM
   70
   80 MODE 1
   90 PROCinitialise
  100 REPEAT
  110 PROCloop
  120 UNTIL 0
  130
  140
  150 DEF PROCloop
  160 COLOUR 2
  170 PRINT TAB(0,7);A$
  180 COLOUR 3
  190 PRINT TAB(ACROSS-2,DOWN);" X "
  200 IF (INKEY(-68)) PROCfire
  210 COLOUR 1
  220 PRINT TAB(0,0);"SCORE:";SCORE," SH
OTS LEFT:";SHOTS;"    "
  230 IF SHOTS<1 THEN PROCend:ENDPROC
  240 ACROSS=ACROSS+(INKEY(-98))-(INKEY(
-67))
  250 A$=RIGHT$(A$,39)+LEFT$(A$,1)
  260 FOR WAIT=0 TO 50:NEXT
```

```
  270 ENDPROC
  280
  290
  300 DEF PROCfire
  310 VDU 19,0,0,0,0,0,0
  320 SHOTS=SHOTS-1
  330 GCOL 4,1
  340 X=(ACROSS*32)-18
  350 Y=1024-(DOWN*32)
  360 MOVE X,Y
  370 DRAW X,Y+200
  380 NEWA$=LEFT$(A$,ACROSS-1)+" "+RIGHT
$(A$,40-ACROSS)
  390 IF MID$(A$,ACROSS,1)<>" " THEN SCO
RE=SCORE+57:A$=NEWA$
  400 SOUND 17,-15,200,2
  410 DRAW X,Y
  420 VDU 19,0,4,0,0,0,0
  430 ENDPROC
  440
  450
  460 DEF PROCend
  470 *FX 15
  480 FOR T=1 TO 5000:NEXT
  490 PRINT TAB(0,12) "  THAT'S THE END
OF THIS GAME..."
  500 PRINT TAB(0,20) "   ANY KEY TO RUN
 AGAIN...."
  510 *FX 15
  520 DUMMY=GET
  530 RUN
  540
  550
  560 DEF PROCinitialise
  570 VDU 23;8202;0;0;0;
```

```
  580 VDU 19,0,4,0,0,0,0
  590 VDU 23,232,0,4,73,222,62,8,0,0
  600 SCORE=0
  610 SHOTS=15
  620 FOR L=1 TO 39
  630 IF RND(12)<6 A$=A$+CHR$(232) ELSE
A$=A$+CHR$(32)
  640 NEXT
  650 ACROSS=15
  660 DOWN=14
  670 ENDPROC
>
```

The nature of that version now makes it very simple to add to the programs changing the logical colour — creating a laser blast and a screen flash for PROCfire — adding sound and a blue background simple because we can see immediately where to put the enhancements.

Line 80 — Sets MODE
90 — Calls PROC initialise which initialises the screen and variables
100 — Start indefinite REPEAT/UNTIL loop
110 — Call PROC loop as it is the procedure in the loop
120 — End of REPEAT/UNTIL loop
150 — Defines PROC loop
160 — Sets colour of ducks
170 — Prints ducks
180 — Sets 'x' colour
190 — Sets position of 'x' and prints it
200 — If F is being pressed then go to the 'fire' section of the program. INKEY with a negative number in it s the same as INKEY (0) except that it is only true or false and if the key denoted by that number has been pressed it is true. Look in the

USER GUIDE to find out which keys have which value. This means the translation of IF INKEY (−69) THEN.....is: If the 'Y' key is being pressed then....

210 — Sets the colour of the scores
220 — Prints out the score
230 — If there is less than one shot then PROCend
240 — As INKEY with a negative number produces −1 if the specified key is pressed the horizontal co-ordinate is decremented by the difference between Z being pressed and X being pressed. When the sum is the difference between −1 and 0 which is −1 the horizontal co-ordinate is decreased. If X is pressed the sum is 0− −1 which is +1 so the horizontal co-ordinate is increased
250 — Makes A$ equal to itself with its front lopped off and added onto the end
260 — The computer pauses with a FOR/NEXT loop
270 — Ends the Definition of PROCloop
300 — Starts the definition of PROCfire
310 — Redefines the background colour of screen from blue to black
320 — Decrements SHOTS as one is about to be fired
330 — Sets graphics colour to inverse anything that is already there (see chapter 14)
340 — Calculates x co-ordinates of line to represent gun blast, adjusts the value of ACROSS to that of the number of graphics points across the middle of the 'x' from the left side of the screen
350 — Calculates y co-ordinates of the top of 'x'
360 — MOVEs graphics cursor to the position near the x from where the 'gun blast' is to come
370 — Draws 200 positions up from the 'x' to the line of the ducks
380 — Makes the position fired at a space
390 — If the position fired at is not a space (i.e. a duck) this line increases the score by 57

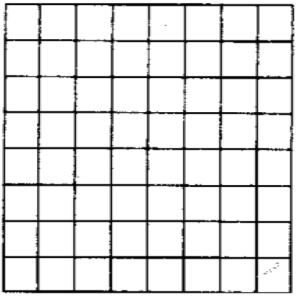
400 — Outputs a SOUND to inform the player that the 'gun' has been fired
410 — DRAWs back to the x and as line 330 sets the drawing to inverse anything tht was there before, the DRAWing deletes the gun-blast line
420 — Redefines the black background to blue again. The computer works so fast that the time the computer to run through the lines between the line which redefined the blue screen to be black (line 310) is so short, that the screen seems to flash black with a white line briefly appearing on the screen
430 — Ends the definition of PROCfire called by line 200, which only stops the program's animation for a moment, as the computer works so fast
460 — Starts the Definition of the procedure which is called when all the SHOTS are used up
470 — This command clears the keyboard buffer. The keyboard buffer is a little part of the computers memory which holds the letter you type in if it is too busy to accept them — in this program the program needs no input but the F's (for the firing of the gun) stored in the computer
480 — Delay loop
490 — Prints the 'end of game' message
510 — Clears the keyboard buffer so players can continue at their leisure
520 — Waits for a key press (really it actually sets the variable DUMMY to ASCII value of the key pressed)
530 — Reruns the program
560 — Starts defining the initialisation procedure
570 — Turns off the cursor
580 — Redefines logical black as actual colour blue
590 — This line defines the character to look like the duck in the display.

The duck has been 'user-defined'. User-defined

graphics are one of the really great features of your computer, and — although certain other computers have similar feature — few, of any, are as simple to use as the one on your machine.

It is very simple to define graphics. We'll take you through the way we created the alleged duck, and from this explanation you should be able to produce anything you like.

The key to the user-defined graphics is an eight by eight grid like this one:



To work out your graphics characters you simply fill the squares on this grid which you want as solid dots in the final graphic with ones and those you don't with zeros. Our duck (in grid form) looks like this:

0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	1	0	0	1	0	0	1
1	1	0	1	1	1	1	0
0	0	1	1	1	1	1	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Reading across each row separately we get what are called binary numbers. For example the third row would be Ø1ØØ1ØØ1. These eight digit binary numbers, or bytes as they are normally called, can all be represented by decimal numbers between Ø and 255.

The way binary works is similar to decimal in that each column has a different value. The column values (from the right) with decimal numbers are 1, 1Ø, 1ØØ, 1ØØØ and so on. These column values are used in reading numbers. We look at a number and multiply each digit by its column value. For example:

1ØØ's	1Ø's	1's				
3	7	8	3 x 1ØØ	=	3ØØ	
378 =			7 x 1Ø	=	7Ø	+
			8 x 1	=	8	+
					378	

With the binary numbering system the column values are (from the right) 1, 2, 4, 8, 16, 32, 64 and in this case 128. So to convert binary to decimal we simply multiply each binary digit (or bit) by its column value as in this example:

128's	64's	32's	16's	8's	4's	2's	1's
1	Ø	1	1	Ø	1	Ø	Ø

1 x 128	=	128	
Ø x 64	=	Ø	
1 x 32	=	32	
1 x 16	=	16	
Ø x 8	=	Ø	
1 x 4	=	4	
Ø x 2	=	Ø	
Ø x 1	=	Ø	
		18Ø	

1Ø11Ø1ØØ (binary) = 18Ø

Using these column values we can work out the decimal numbers from those eight binary numbers need to produce the user-defined graphic character.

128's	64's	32's	16's	8's	4's	2's	1's		
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	=	Ø
Ø	Ø	Ø	Ø	Ø	1	Ø	Ø	=	4
Ø	1	Ø	Ø	1	Ø	Ø	1	=	73
1	1	Ø	1	1	1	1	Ø	=	222
Ø	Ø	1	1	1	1	1	Ø	=	62
Ø	Ø	Ø	Ø	1	Ø	Ø	Ø	=	8
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	=	Ø
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	=	Ø

so the syntax of the VDU 23, command is VDU 23, character to be defined, 1st row, 2nd row, 3rd row. . . . 8th row. You have characters 224 to 255 free for redefinition.

600 — Sets the score to Ø
610 — Sets the amount of shots to be 15
620 — Starts a loop to randomly create the string that holds the ducks
650 —
660 — Set co-ordinates of the x start position
670 — Ends the initialisation routine.

There are various things you can do to further develop the program:

1. Define an ENVELOPE to be used in conjunction with sound channel Ø to make a quacking sound.

2. Define a character to replace the x to look like a gun or a man (in line 190 use " "; CHR$(233[number of the redefined charcter]); " " instead of "x".
3. Add "high score" feature so the game will restart preserving a high score for you to beat.
4. When all the shots *do* hit a duck each, make the program play a tune for the winner. You can do this by using the READ statement in conjunction with the SOUND statements.
5. Change the number of shots.
6. Make the stream of ducks go on for ever — i.e. not looping on the screen but an infinite number of ducks to get!
7. Try putting the game in MODE 7 — though it has no user defined graphics.

CHAPTER THIRTEEN
GETTING LISTED

An *array* is used when you want to create a list of items, and refer to the item by mentioning just which position in the list it occupies. You set up an array by using the DIM command. If you type in DIM A(5), the computer will set up a list in its memory called A, and will save space for five items: A(1), A(2), A(3), A(4) and A(5). These, by the way, are called *elements* of the array.

When you *dimension*, or set up, an array, the computer creates the list in its memory, and then fills every item in that list with a zero. So, if you said to the computer, PRINT A(1) or PRINT A(4) it would come back with Ø. You fill items in an array with LET (such as LET A(2) = 1ØØØ) or using READ and DATA as we saw in chapter eleven. Once you've filled a position in the list, whether you filled it with LET or READ, you can get the computer to give you the contents of that element of the array, by simply saying PRINT A(2).

Arrays when dimensioned on the BBC computer have more space than you would at first think. The array has one more element that we have not mentioned, A(Ø). So if we DIM A(5) we set up six elements. The Ø element has not be used in the following examples.

The next program *dimensions* (this, as we said, simply means sets up, or creates) an array called A, with room for sixteen elements. The B loop from line 3Ø to 5Ø fills the array with

random digits between one and nine, and then prints them back for you with the loop from lines 70 to 90.

```
10 REM ARRAYS....
20 DIM A(15)
30 FOR B=1 TO 15
40 A(B)=RND(9)
50 NEXT
60 FOR Z=1 TO 15
70 PRINT "A(";Z;") IS ";A(Z)
80 DUMMY=GET
90 NEXT
```

This is called a *one-dimensional* array, because a *single digit* follows the letter which "labels" the array. You can also have *multi-dimensional* arrays, in which *more than one number* follows the array label after DIM. In the next program, for example, the computer sets up a two-dimensional array called A, consisting of four elements by four elements (that is, it is dimensioned by DIM A(4,4) as you can see in line 20):

```
 10 REM ARRAYS....
 20 DIM A(4,4)
 30 FOR B=1 TO 4
 35 FOR C=1 TO 4
 40 A(B,C)=RND(9)
 50 NEXT ,
 60 PRINT "  1 2 3 4 "
 70 FOR B=1 TO 4
 80 PRINT ;B;CHR$(134);
 90 FOR C=1 TO 4
100 PRINT ;A(B,C);" ";
110 NEXT
120 PRINT
130 NEXT
```

When you run it, you'll see something like this:

```
    1  2  3  4
1   7  1  3  7
2   6  8  7  8
3   4  9  7  9
4   7  7  9  2
```

You specify the element of a two-dimensional array by referring to *both* its numbers, so the first element of this array (the number 7 in the top left hand corner of the printout above) can be referred to as A(1,1). The 7 at the end of the line is A(1,4) and the 9 on the bottom row is A(4,3).

Your computer also supports *string* arrays. Enter and run this routine to see a string array in operation:

```
  10 REM ARRAYS....
  20 DIM A$(5)
  30 FOR B=1 TO 5
  40 READ A$(B)
  50 NEXT
  60 FOR Z=1 TO 5
  70 PRINT "A$(";Z;") IS ";A$(Z)
  80 DUMMY=GET
  90 NEXT
 100 DATA Pogamoggan,Antisurrejoinder,P
iedmontite,Lymphogranuloma,Erythredemapo
lyneuropathy


A$(1) IS Pogamoggan
A$(2) IS Antisurrejoinder
A$(3) IS Piedmontite
A$(4) IS Lymphogranuloma
A$(5) IS Erythredemapolyneuropathy
```

ESCAPING FROM MURKY MARSH

This program demonstrates the use of a two-dimensional array for "holding" an object and for printing it out. The object in this case is a miniature dragon, who is trying to escape from Murky Marsh, indicated in this program by a square of brightly-coloured dots. Our dragon is very stupid, and moves totally at random within the marsh. He is free if he manages to stumble onto the outer right or bottom two rows. The dragon in this program demonstrates *Brownian* motion, the random movement shown by such things as tiny particles in a drop of water viewed under a microscope, or of a single atom in a closed container. Brownian motion explains why a drop of ink gradually mixes in which the water into which it has been placed.

In the second version of the program, you can see how a third dimension added to each co-ordinate lets you have two characters or more at each co-ordinate of the "marsh".

Here is the program listing:

```
 10 REM Dragon's Lair
 20 REM
 60 REM
 65
 70 DIM A(10,10)
 80 CLS
 90 M=0
100 GOSUB 340
105
110 VDU 23;8202;0;0;0;
120 REPEAT
130 IF RND>.35 THEN P=P+1 ELSE P=P-1
140 IF RND>.35 THEN Q=Q+1 ELSE Q=Q-1
150 IF Q<1 THEN Q=1
160 IF Q>10 THEN Q=10
170 IF P<1 THEN P=1
```

```
  180 IF P>10 THEN P=10
  190 M=M+1
  200 VDU 31,15,1
  210 PRINT "Attempt No. ";M;"   ";P;" ";
Q;" "
  220 A(P,Q)=35
  230 COLOUR 132
  240 VDU 31,15,3
  250 FOR X=1 TO 10
  260 FOR Y=1 TO 10
  270 PRINT CHR$(A(X,Y));" ";
  280 NEXT
  290 PRINT''TAB(15);
  300 NEXT X
  310 A(P,Q)=124
  320 UNTIL Q<9 AND P>9
  330 GOTO 410
  335
  340 Q=RND(3)+3
  350 P=RND(3)+3
  360 FOR X=1 TO 10
  370 FOR Y=1 TO 10
  380 A(X,Y)=46
  390 NEXT ,
  400 RETURN
  405
  410 PRINT'TAB(10)"Whew..Free at last!"
```

And another version with teletext control codes:

```
10 REM Dragon's Lair
20 REM
60 REM
70
80
90 DIM A(10,10,1)
```

```
100 MODE 7
110 M=0
120 PROCinit
130
140 VDU 23;8202;0;0;0;
150 REPEAT
160 IF RND>.35 THEN P=P+1 ELSE P=P-1
170 IF RND>.35 THEN Q=Q+1 ELSE Q=Q-1
180 IF Q<1 THEN Q=1
190 IF Q>10 THEN Q=10
200 IF P<1 THEN P=1
210 IF P>10 THEN P=10
220 M=M+1
230 VDU 31,5,1
240 PRINT "Attempt No. ";M;"  ";P;" ";
Q;" "
250 A(P,Q,1)=35 : A(P,Q,0)=130
260 VDU 31,5,3
270 FOR X=1 TO 10
280 FOR Y=1 TO 10
290 PRINT CHR$(A(X,Y,0));CHR$(A(X,Y,1)
);
300 NEXT
310 PRINT''TAB(5);
320 NEXT X
330 A(P,Q,1)=124:A(P,Q,0)=132
340 SOUND 1,-15,20*P+2*Q,3
350 UNTIL Q>9 AND P>9
360 GOTO 520
370
380 DEF PROCinit
390 Q=RND(3)+3
400 P=RND(3)+3
410 FOR X=1 TO 10
420 FOR Y=1 TO 10
```

```
430 A(X,Y,1)=46
440 A(X,Y,0)=134
450 NEXT ,
460 FOR T=0 TO 24
470 VDU 31,0,T,129,157,134
480 NEXT
490 VDU 28,3,23,38,0
500 ENDPROC
510
520 PRINT'TAB(10)"Whew..Free at last!"
```

DAVEY JONES' LOCKER

Time now to relax with our next game — FULL FATHOM FIFTY — which uses two arrays (B and D) to store the player's scores and dice rolls respectively. It is a simple game. You and the computer are in a race to roll a total of 50 or more. You roll two dice at once, but the only time you score and get the total of the dice roll added to your accumulating total, is when both dice come up with the same number.

You should know enough about how programs work at this stage to be able to determine what each section does, so we will not explain this relatively simple program. Get it up and running, and then decide for yourself how each section works. You'll possibly gain more from the program by working it out yourself rather than having it explained in detail by us.

Here is the program listing for Full Fathom Fifty:

```
 10 REM Full Fathom Fifty
 20 REM
 60 REM
 70
 80 MODE 7
 90 PROCinit
100 CLS
```

```
   110 ROUND=1
   120 FOR A=1 TO 2
   130 CLS
   140 FOR D=0 TO 1
   150 PRINT TAB(0,0)redlet$;newbar$;"
   "doublet$;velvet$;"    Full Fathom Fift
y"
   160 NEXT
   170 PRINT TAB(11,10)"  Round No.:";ROU
ND
   180 PRINT TAB(11,12)"B.B.C. Micro:";B(
1)
   190 PRINT TAB(11,14)"    Human :";B(2)
   200 IF B(1)>39 OR B(2)>39 THEN PROCend
   210 IF A=1 PRINT TAB(6,20)"Now I'll ro
ll the dice..." : GOTO 250
   240 PRINT TAB(5,20)flalet$;"Press 'R'
to roll the dice..."
   250 COLOUR=SET
   260 PRINT TAB(3,20) STRING$(35," ")
   265 FOR C=1 TO 2
   270 PRINT TAB(1,6)"Rolling Die No. ";C
   275 PRINT TAB(20,6);STRING$(19," ")
   280 FOR T=1 TO 50:SOUND 1,-15,RND(200)
,1:PRINT TAB(3*T,6);RND(6):FOR TT=1 TO T*
10:NEXT,
   290 D(C)=RND(6)
   295 SOUND 1,-15,(D(C)*25)+50,10
   300 PRINT TAB(24,6)"It came up:";D(C)
   310 FOR T=0 TO 2000:NEXT
   320 NEXT C
   330 PRINT TAB(1,6) STRING$(38," ")
   340 IF D(1)<>D(2) THEN PRINT TAB(7,12)"
";"Those Rolls do not count     "
   350 IF D(1)=D(2) THEN R(A)=R(A)+3*D(1)
```

```
:PRINT TAB(8,17);"The Roll was ";D(1);"
and ";D(1)
  360 FOR T=0 TO 3000:NEXT
  370 NEXT A
  380 ROUND=ROUND+1
  390 FOR T=0 TO 2000:NEXT
  400 GOTO 120
  405
  406
  410 DEF PROCend
  420 FOR D=3 TO 4
  430 PRINT TAB(1,0)blulet$;doulet$;"The
 Winner is the ";
  440 IF B(1)>B(2) THEN PRINT "B.B.C. Mi
cro"
  450 IF B(1)<B(2) THEN PRINT "human"
  460 NEXT
  470 REPEAT UNTIL FALSE
  480
  485
  490 DEF PROCinit
  500 DIM B(2),D(2)
  510 B(1)=0:B(2)=0
  520 VDU 23;8202;0;0;0;
  530 newbac$=CHR$(157)
  540 redlet$=CHR$(129)
  550 yellet$=CHR$(131)
  560 blulet$=CHR$(132)
  570 doulet$=CHR$(141)
  580 flalet$=CHR$(136)
  590 ENDPROC
```

CHAPTER FOURTEEN
LET THE GOOD GRAPHICS ROLL

Your computer has a very wide range of graphics facilities and in this chapter we'll introduce you to some of them. After the introduction to each function, we'll give a program or two to illustrate it.

These demonstrations should be seen only as *introductions* to the graphics potential of your computer. Experimentation will show just how far you can stretch your computer.

THE SCREEN

The graphics are displayed on the 'graphics screen' based on a grid, similar to the grid used when the PRINT TAB statement is used, but with two main differences: the numbers in the brackets after TAB are co-ordinates with the origin (Ø Ø), i.e., the starting point, at the top left-hand corner of the screen. The graphics are based on a grid starting at the bottom left-hand corner of the screen. But the main difference between PRINT TAB and the graphics commands is the fact that the graphics grid is much finer, numbered from Ø to 1279 on the x-axis and Ø to 1Ø23 on the y-axis.

The way the graphics work depends on which MODE you are in. Of the 8 modes, three modes, 3, 6 and 7, have no graphics capability. The reason there is more than one kind of graphics

mode (modes Ø, 1, 2, 4 and 5) is because different modes use up a different amount of memory. The first three modes all take up 2ØK, but each has a different number of colours. Basically the thinner the lines on the screen the lower the number of colours, because the computer needs more memory for thinner lines (as more can fit on one screen) and more memory for a larger amount of colours.

Although the grids for these modes accommodate a different number of vertical lines, their numbering is the same, so programs need not be changed to work in different modes. Imagine that the screen is divided into a grid 128Ø units long and 1Ø24 units across. Thus each point on the screen has co-ordinates and using these co-ordinates we can join different points with lines. They keyword DRAW x,y helps you do this. 'x' and 'y' represent the co-ordinates to which you wish to draw to, using the aforementioned grid. When using the graphics you start at the origin, Ø,Ø, at the bottom left-hand corner of the screen. Run the next program, entering numbers between zero and 1279 for x, and zero and 1Ø23 for y, to see DRAW in action:

```
 10 REM DRAW X,Y
 20 REM
 40 MODE 1
 50 PRINT TAB(0,0)"ENTER X:";
 60 INPUT X
 70 PRINT TAB(0,0)"                    "
 80 PRINT TAB(0,0)"ENTER Y:";
 90 INPUT Y
100 PRINT TAB(0,0)"                    "
110 PRINT TAB(0,1)"DRAW ";X;",";Y
120 DRAW X,Y
130 GOTO 50
```

So to summarise, DRAW X,y plots a line from the last point

visited to co-ordinates x,y where the value of x is between Ø and 1279 and y between Ø and 1Ø24.

PLOT

When creating graphics, you may not wish to have a drawing made up of a continuous line, starting at the bottom left of the screen. Fortunately there are other ways of joining points on the screen using the PLOT k,x,y statement. x and y are the co-ordinates to which you wish to plot and k defines the way the PLOTting is done. Two of the most used PLOT statements are PLOT 4,x,y and PLOT 5,x,y — these have their own BASIC keywords: MOVE x,y and DRAW x,y respectively.

As you might expect MOVE x,y moves the graphics cursor to position x,y so the next DRAW or PLOT statement will store the x,y as "the last point visited". There are over 5Ø PLOT variations, i.e. over 5Ø different values for k — each of which behaves differently.

To describe plotting inevitably involves some graphics jargon. "Line absolute" means a line whose co-ordinates are measured from the origin (at the bottom left-hand of the screen) while "line relative" means that the destination co-ordinates (x and y at the end of the statement) are measured from the previously visited point. This means that if the graphics cursor was at (5ØØ,5ØØ) i.e. the last point visited, and the point you wanted to draw to was (4ØØ,45Ø) you could draw a "line absolute" by using the statement PLOT 5, 4ØØ,45Ø or DRAW 4ØØ,45Ø by using PLOT 1,x,y which is the 'draw line relative' statement, in this case it would be PLOT 1, – 1ØØ, – 5Ø because the graphics cursor moves minus 1ØØ positions to the right and minus 5Ø positions up (which comes to the same as 1ØØ to the left and 5Ø down the screen).

COLOUR

You will remember that when you start typing in any MODE you have a screen which is white text on a black screen — this is

the case until you use a COLOUR statement — this is the same with graphics: you start with white lines on a black screen until you use a GCOL a,b statement. GCOL is short for Graphics COLour and the two numbers following it define 1) how the colour is plotted and 2) which colour is used. In GCOL Ø,x 'x' is the colour you wish to use. If x is greater than 127 then that colour will be the graphics background colour which will show after the graphics screen is cleared using the command CLG. You may come across the words 'logical inverse colour' — the 'logical inverse of a colour is the colour which is its opposite:

In two-colour modes the inverse of colour Ø is colour 1, while in four-colour modes it is like this:

Colour	inverse
Ø	3
1	2
2	1
3	Ø

and in the sixteen-colour mode the opposite of a colour will also flash alternating with its logical opposite (see chapter 12).

PLOT Ø,x,y	moves the graphics cursor relative to the last position (x units to the right and y units up).
PLOT 1,x,y	draws a line relative to the last position.
PLOT 2,x,y	draws a "line relative" in the logical inverse colour, that is a line drawn in the logical inverse colour of the colour in the GCOL statement.
PLOT 3,x,y	draws a line relative in the background colour in use, so that if it is drawn over some foreground colour it can be seen — otherwise it can be used to move the graphics cursor relative to the last position.
PLOT 4,x,y	moves to absolute position (i.e. x units to the

right from the origin and y units up). MOVE x,y can also be used: it sets the graphics position.

PLOT 5,x,y — draws to absolute position from last graphics point.

PLOT 6,x,y — draws line absolute in logical inverse colour.

PLOT 7,x,y — draws line absolute in background colour defined by GCOL 0, 127 + x (where x is normal foreground colour).

There are some more plotting statements which are useful although they may not seem vital at first glance:

PLOT 21,x,y — draws a line absolute using a dotted line.

PLOT 69,x,y — places a point absolute at x,y.

PLOT 85,x,y — draws a triangle between the last two points and x,y.

Here are some progams using PLOT statements to create graphic displays:

```
 10 REM PLOT 1,X,Y + 1
 20 REM
 40 MODE 1
 50 PROCmd
 60 MOVE X,Y
 70 PLOT 1,X,1023-Y
 80 FOR T=0 TO 1000:NEXT
 90 MOVE X,Y
100 PLOT 1,1279-X,Y
110 FOR T=0 TO 1000:NEXT
120 MOVE X,Y
130 PLOT 1,1279-X,1023-Y
140 FOR T=0 TO 1000:NEXT
```

```
150 GOTO 50
160
170 DEF PROCrnd
180 X=RND(1280)-1
190 Y=RND(1024)-1
200 ENDPROC
```

```
 10 REM PLOT 1,X,Y - 2
 20 REM
 40 MODE 1
 50 PROCrnd
 60 MOVE X,Y
 70 PLOT 1,X,1023-Y
 80 MOVE X,Y
 90 PLOT 1,1279-X,Y
100 MOVE X,Y
110 PLOT 1,1279-X,1023-Y
120 GOTO 50
130
140 DEF PROCrnd
150 X=RND(1280)-1
160 Y=RND(1024)-1
170 GCOL 0,RND(3)
180 ENDPROC
```

```
 10 REM PLOT 1,X,Y - 2
 20 REM
 40 MODE 1
 50 PROCrnd
 60 MOVE 640,512
 70 PLOT 1,X,Y
 80 MOVE 640,512
```

```
 90 PLOT 1,X,-Y
100 MOVE 640,512
110 PLOT 1,-X,Y
120 MOVE 640,512
130 PLOT 1,-X,-Y
140 GOTO 50
150
160 DEF PROCrnd
170 X=RND(1280)-1
180 Y=RND(1024)-1
190 GCOL 0,RND(3)
200 ENDPROC
```

```
 10 REM BUTTERFLY PLOT
 20 REM
 40 MODE 1
 50 X=RND(640)-1
 60 Y=RND(512)-1
 70 PLOT 69,X,Y
 80 PLOT 69,1280-X,Y
 90 PLOT 69,X,1024-Y
100 PLOT 69,1280-X,1024-Y
110 GOTO 50
```

```
 10 REM BUTTERFLY PLOT V. 2
 20 REM
 40 MODE 1
 50 X=RND(640)-1
 60 Y=RND(512)-1
 70 GCOL 0,RND(3)
 80 PLOT 69,X,Y
 90 PLOT 69,1280-X,Y
```

```
100 PLOT 69,X,1024-Y
110 PLOT 69,1280-X,1024-Y
120 GOTO 50
```

```
 10 REM BUTTERFLY PLOT V. 3
 20 REM
 40 MODE 2
 50 X=RND(640)-1
 60 Y=RND(512)-1
 70 GCOL 0,RND(15)
 80 PLOT 69,X,Y
 90 PLOT 69,1280-X,Y
100 PLOT 69,X,1024-Y
110 PLOT 69,1280-X,1024-Y
120 GOTO 50
```

```
 10 REM OVAL
 20 REM
 40 MODE 1
 50 FOR A=1 TO 1305 STEP 4
 60 B=PI*A/650
 70 C=481*COS(B)+790
 80 D=470*SIN(B)+553
 90 PLOT 69,C,D
100 PLOT 69,C/2,D
110 PLOT 69,C,D/2
120 PLOT 69,C/2,D/2
130 NEXT A
```

```
 10 REM OVAL 2
 20 REM
```

```
40 MODE 0
50 MOVE 127,555
60 FOR A=1 TO 1305 STEP 8
70 B=PI*A/650
80 C=481*COS(B)+790
90 D=470*SIN(B)+553
100 DRAW C,D
110 DRAW C/2,D
120 DRAW C/2,D/2
130 DRAW C,D/2
140 NEXT A
```

```
10 REM COSINE PLOTTING
20 REM
40 MODE 1
50 FOR A=0 TO 5*PI STEP .05
60 PLOT 69,80*A,240*COS(A)+430
70 NEXT A
```

```
10 REM SINE PLOTTING
20 REM
40 MODE 1
50 FOR A=0 TO 5*PI STEP .05
60 PLOT 69,80*A,240*SIN(A)+430
70 NEXT A
```

```
10 REM SINE PLOTTING 2
20 REM
40 MODE 1
50 FOR A=0 TO 10*PI STEP .05
60 PLOT 69,40*A,240*SIN(A)+430
70 NEXT A
```

```
10 REM SINE PLOTTING 3
20 REM
40 MODE 1
50 FOR T=0 TO 80 STEP 8
60 FOR A=0 TO 10*PI STEP .15
70 PLOT 69,40*A+T,240*SIN(A)+430
80 NEXT A
90 NEXT T
```

```
10 REM COS/SINE PLOTTING 2
20 REM
40 MODE 1
50 FOR A=0 TO 5*PI STEP .1
60 MOVE 80*A,240*COS(A)+430
70 DRAW 80*A,240*SIN(A)+430
80 NEXT A
```

```
10 REM COS/SINE PLOTTING
20 REM
40 MODE 1
50 FOR A=0 TO 5*PI STEP .05
60 PLOT 69,80*A,240*COS(A)+430
70 PLOT 69,80*A,240*SIN(A)+430
80 NEXT A
```

GCOL

We have not as yet fully explained the operation of the GCOL statement. GCOL 0,x sets the graphics colour as do all the five GCOL statements — but the first number dictates how the graphics drawn affect the things they are plotted over. This means GCOL sets the way colours 'mix'. The next section is

only for the reader who wants to predict very precisely how his colours will mix, and should be skipped by the fainthearted!

The way colours mix is related to "bit-wise operations", a somewhat complex set of operations which relate to the statements OR and AND. Bit-wise operators look at the binary digits of a number comparing them to the binary digits of another number (we have dealt with binary in chapter 12). The best way to describe these operators is by showing you some examples:

```
If we OR number  0 1 0 1 1 0 1 1 0 (number "A")
      andnumber  0 1 1 1 0 1 1 1 0 (number "B")
         we get  0 1 1 1 1 1 1 1 0 (the result)
```

What OR does is compare the two corresponding digits of two binary numbers, taking it column by column (vertically). In the example above, the corresponding digit of the result will be 1, if the first digit of the number identified as "A" or the first digit of number "B" is 1. In this case neither the first digit of "A" nor the first digit of "B" is 1, so the first digit of the result is 0, while the second digit of the result will be 1, because the second digit of "A" OR the second digit of "B" is 1.

```
If we AND     0 1 0 1 1 0 1 1 0 (A)
      and     0 1 1 1 0 1 1 1 0 (B)
   we get     0 1 0 1 0 0 1 1 0
```

What AND does is to compare the two digits. If the first AND and the second are both 1, then the answer is 1.

As well as these 'bit-wise' versions of AND and OR there is another 'bit-wise' operator used in the GCOL statement: EOR (which stands for Exclusive OR).

```
If we EOR     0 1 0 1 1 0 1 1 0 (A)
      and     0 1 1 1 0 1 1 1 0 (B)
   we get     0 0 1 0 1 1 0 0 0
```

EOR registers if digits are different (Ø and 1 or 1 and Ø) and if they are it makes the corresponding result digit 1.

And here we come to the point of the above explanations: take the first number ("A" in each of the above cases) as the screen and the second number ("B") as the dotted line (produced by PLOT 21,x,y) the result of each operation signifies which pixels (i.e. patterns) would be lit if the colour was ORed, ANDed or EORed.

GCOL Ø,x	draw lines with specified colour x
GCOL 1,x	draw lines with x ORed, with the background (i.e. whatever it is drawing over)
GCOL 2,x	draw lines with x ANDed with the screen
GCOL 3,x	draw lines with x EORed with the screen
GCOL 4,x	draw lines inverting the screen colour

Here are two programs by Jeremy Ruston which show what you can do with the graphics on the BBC micro:

```
 10 REM String Art
 20 REM
 30 REM By Jeremy Ruston
 40 REM
 50 REM (c) J. Ruston 1983
 60 REM
 70
 80 REPEAT
 90 MODE 0
100 VDU 29,640;512;
110 A=RND(100)/10
120 B=RND(100)/10
130 C=RND(100)/10
```

```
140 D=RND(100)/10
150 T=0
160 REPEAT
170 A%=SIN(RAD(T*A))*600
180 B%=COS(RAD(T*D))*500
190 C%=SIN(RAD(T*C))*600
200 MOVE A%,B%:DRAW C%,D%
210 MOVE -A%,B%:DRAW -C%,D%
220 T=T+0.5
230 UNTIL INKEY(-1)
240 UNTIL FALSE
```

```
10 REM Tile
20 REM
30 REM by J.Ruston
40 REM
50 REM (c) J. Ruston
60 REM
70
80 MODE 0
90 G%=64
100 VDU 29,640;512;
110 FOR X%=-640 TO 640 STEP G%
120 FOR Y%=-512 TO 512 STEP G%
130 IF ABS((X% DIV G%) MOD 2)=ABS((Y%
DIV G%) MOD 2) THEN PROCsquare(X%,Y%)
140 NEXT Y%,X%
150 END
160
170 DEF PROCsquare(X%,Y%)
180 PROCplot(4,X%,Y%)
190 PROCplot(4,X%+G%,Y%)
200 PROCplot(85,X%,Y%+G%)
210 PROCplot(85,X%+G%,Y%+G%)
220 ENDPROC
```

```
  230
  240 DEF PROCplot(K%,X%,Y%)
  250 LOCAL D%,A%
  260 IF X%=0 THEN X%=1
  270 D%=SQR(X%*X%+Y%*Y%)
  280 IF D%>500 THEN
PLOTK%,X%,Y%:ENDPROC
  290 A%=DEG(ATN(Y%/X%))
  300 IF SGN(X%)=-1 THEN A%=A%-180
  310 D%=SIN(RAD(D% DIV 6))*500
  320 PLOT
K%,COS(RAD(A%))*D%,SIN(RAD(A%))*D%
  330 ENDPROC
```

CHAPTER FIFTEEN
MORE GREAT GAMES

In this final chapter, we'll be giving you the listings of a number of major programs. The notes on these will not be as extensive as you've had with some programs, but the introductions to the programs will highlight some of the more interesting programming techniques used. Examination of the listings will give you a number of ideas you can apply to your own programs.

BAGATELLE

This is a simplified pinball machine, in which you and the computer take it in turns to roll diamond-shaped balls down a frame which contains a series of numbers and letters. Your score tends to increase each time you hit an obstacle on the way down the frame, and the ball will bounce off the obstacle, increasing your chances of hitting more obstacles. Hitting the sides of the frame, or bouncing around on the bottom, can cost you small penalties. In general, though, each time the ball hits something you'll hear it do so, and have the satisfaction of seeing your score increase.

You start your ball rolling down the frame from the numbers 1 to 9. You and the computer have five balls each to roll down the bagatelle frame, and you take it in turns to do so. You have the first roll. Then the computer will tell you which ball it intends to roll down the frame, and then roll this ball. The most noteworthy part of this program is the FNscr, which is used in lines 32Ø, 34Ø and 36Ø. The numbers following the word FNscr

are like the numbers which follow the PRINT TAB command. However, instead of printing something at that location on the screen, FNscr checks to see what is there. In each case in this program, it is checking *around* the current position of the ball, to see if it is about to strike something. If it finds anything except a space, the score is incremented by the CODE of the character, minus 48. You'll see why, if you look up the code of the ":" which is used for the walls, and the ASCII code of "l". Good rolling.

```
   10 REM Bagatelle
   20 REM
   60 REM
   70
   80 MODE 7
   90 PROCinit
  100
  110 PROCscreen
  120 REM Loop Five Times....
  130 FOR Y%=1 TO 5
  140 PRINT TAB(35,11);Y%
  150 REM Loop Twice at first the Player
          then the computer....
  160 FOR X=1 TO 2
  170 REM Start-up options for both huma
n         and computer....
  180 IF X=1 THEN PRINT TAB(28,17)"Your
Turn"
  190 IF X=2 THEN PRINT TAB(28,17)"My Tu
rn   "
  200 IF X=1 THEN REPEAT:PRINT TAB(22,19
)"Which No. to start";VDU 31,28,21:INPUT
 K%:UNTIL K%>0 AND K%<10
  210 IF X=2 THEN PRINT TAB(22,19)"I wil
l start with ";VDU 31,28,21:K%=RND(9):PR
INT;K%
```

```
290 REM Add 9 To Number to place it
        below appropriate number....
300 BA=BA+9:EBA=BA
340 BD=1:IA=1:ID=1
350 EBD=1
360 REM Start Printing " " and "@" to
        animate ball-" " deleting the
        previous ball...
370 PRINT TAB(EBA,EBD);" ":EBA=BA
380 EBD=BD
390 PRINT TAB(BA,BD);"@"
500 FLAG=0
510 REM Check one position ahead of
        ball in different directions
        and set flag and variables...
520 A=FNscr(BA,BD+1)
530 IF A<>32 THEN C(X)=C(X)+A:FLAG=1:G
OTO 580
540 A=FNscr(BA+1,BD)
550 IF A<>32 THEN C(X)=C(X)+A:FLAG=1:G
OTO 580
560 A=FNscr(BA-1,BD)
570 IF A<>32 THEN C(X)=C(X)+A:FLAG=1:G
OTO 580
580 IF FLAG=1 THEN SOUND 1,-12,10,3:IF
 RND(1)<.5 THEN IA=-IA:ID=-ID:SOUND 1,-12,1
0*RND(1)+4*RND(1),3
590 REM Print out Scores...
600 PRINT TAB(35,4);C(1)
610 PRINT TAB(35,6);C(2)
620 REM Change direction of ball...
630 BD=BD+(ID*RND(1))+.75
640 REM Check position of ball and
        take action if it is about to
        leave board....
```

```
  450 IF BA<7 THEN IA=-IA:BA=8
  460 IF BA<1 THEN ID=-ID:BD=1
  470 IF BA>21 THEN IA=-IA:BA=19
  480 BA=BA+(IA*RND(1))
  490 REM Check position of ball,if it
          has not reached the bottom of
          the board return to ball move
          ment..
  500 IF BD<19 THEN 270
  510 SOUND 1,-14,250,8
  520 PROCscreen
  530 NEXT,
  540 PROCend
  550 GOTO 100
  560
  570 DEF FNscr(X%,Y%)
  580 REM Function To find out character
          position at (X,Y)
  590 A%=135
  600 VDU 31,X%,Y%
  610 C%=((USR(&FFF4) AND &FFFF) DIV &10
0)
  620 IF C%<128 THEN =C%
  630=(C% MOD 32)+224
  640
  650 DEF PROCscreen
  660 REM Sets up screen...
  670 PRINT TAB(0,0)B$;A$;"!!!!123456789
!!!! "$G$
  680 FOR G=1 TO 19
  690 PRINT TAB(0,G)B$;A$;"
   ! "$B$
  700 NEXT
  710 PRINT TAB(7,2) "    i  i  i"
  720 PRINT TAB(7,4) " Z X  Z X  Z X"
```

```
  730 PRINT TAB(7,6) "  1 11   11 1"
  740 PRINT TAB(7,9) " 7   7   7   7"
  750 PRINT TAB(7,11)"  5   5 5   5"
  760 PRINT TAB(7,14)"9      9     9"
  770 PRINT TAB(7,16)"    8     8"
  780 PRINTTAB(0,20)B$;A$;"!!!!!!!!!!!!!!
!!!! ";B$
  790 PRINTTAB(0,21)B$;A$;"!!!!!!!!!!!!!!
!!!! ";B$
  800 PRINT TAB(0,22)B$;CHR$(141);"
 BAGATELLE"
  810 PRINT TAB(0,23)B$;CHR$(141);"
 BAGATELLE"
  820 PRINT TAB(27,4) "  Human:";C(1)
  830 PRINT TAB(27,6) "Machine:";C(2)
  840 PRINT TAB(27,11)"Ball No.:";
  850 PRINT TAB(28,21)"    "
  860 ENDPROC
  870
  880 DEF PROCend
  890 FOR G%=1 TO 19
  900 PRINT TAB(0,G%)B$;A$;"!
    ! ";B$
  910 NEXT
  920 PRINT TAB(11,3)"G A M E"
  930 PRINT TAB(11,5)"O V E R"
  940 PRINT TAB(7,7)"The Winner is.."
  950 VDU 31,7,9
  960 DUMMY=GET
  970 IF C(1)>C(2) THEN PRINT "   The Hu
man ":ELSE PRINT "   The Computer"
  980 PRINT TAB(7,12)"PRESS ANY KEY TO"
  990 PRINT TAB(7,13)"   RUN AGAIN.."
 1000 DUUMY=GET
 1010 ENDPROC
```

```
1020
1030 DEF PROCinit
1040 REM If ERROR occurs...End Program
1050 ON ERROR CLS:PRINT A$;CHR$ 141;"
     PROGRAM END...":PRINT A$;CHR$ 141;"
      PROGRAM END...":END
1060 DIM C(2)
1070 REM Set A$ to make a red backgrour
d
1080 A$=CHR$(129)+CHR$(157)+CHR$(131)
1090 REM Set B$ to make the background
          blue
1100 B$=CHR$(132)+CHR$(157)+CHR$(134)
1110 REM Turn Cursor off
1120 VDU 23;8202;0;0;0;
1130 ENDPROC
1140
```

FOLLOW ME

In this game, you have to duplicate the colour and tone pattern generated by the computer. There is a catch. At first, you will have only one sound/colour combination to remember. Then the computer will add a new combination to the first one, and play both. You will have to repeat both. Then the sequence will become three sound/colour combinations long, and you have to repeat all three . . . and so on.

Use "R" for red, "B" for blue, "Y" for yellow, "C" for green, to copy the colours.

```
10 REM Sound...Follow Me..
20 REM
60 REM
70
80 MODE 1
90
```

```
100 PROCinit
110
120 REPEAT
130   PROCcalc
140   PROCdisplay
150 PROCcheck
160 UNTIL end
170
180 RUN
190
200 DEF PROCsquare(pos)
210 VDU 19,0,pos,0,0,0,0,0
220 CLS
230 VDU 20,26
240 SOUND 17,-15,50+20*pos,DEL/9
250 FOR T=0 TO 10*DEL:NEXT
260 ENDPROC
270
280
290 DEF PROCdisplay
300 FOR L=1 TO LEN(A$)
310 P=VAL(MID$(A$,L,1))
320 PROCsquare(P)
340 CLS
350 VDU 26
360 NEXT
370 ENDPROC
380
390
400 DEF PROCcalc
410 A$=A$+STR$(RND(4))
420 ENDPROC
430
440
450 DEF PROCassignA
```

```
  460 IF G=82 THEN P1=1
  470 IF G=67 THEN P1=2
  480 IF G=89 THEN P1=3
  490 IF G=66 THEN P1=4
  500 ENDPROC
  510
  520
  530 DEF PROCcheck
  540 L=0
  550 PRINT TAB(0,20)" Now Input that la
st sequence....."
  560 L=L+1
  570 G=GET
  580 PROCassignA
  590 PROCsquare(P1)
  600 P=VAL(MID$(A$,L,1))
  610 IF P<>P1 THEN PROCwrong:ENDPROC
  620 PRINT TAB(0,20)" Good...Input Next
 note..."
  630 IF L<LEN(A$) THEN GOTO 560
  640 PRINT TAB(0,20)" Well done ! - Any
 key to Continue..."
  650 DEL=DEL-(DEL/12)
  660 *FX 21 0
  670 DUMMY=GET
  680 ENDPROC
  690
  700
  710 DEF PROCwrong
  720 PRINT TAB(0,20)" Oops !! Wrong-Any
 key to repeat tune"
  730 IF LEN(A$)>7 THEN PRINT "...But yo
u did get "$LEN(A$)$" notes..."
  740 DUMMY=GET
  750 PROCdisplay
```

```
  760 PRINT TAB(0,20)"   ...Any key to re
run"
  770 DUMMY=GET
  780 end=TRUE
  790 ENDPROC
  800
  810
  820 DEF PROCinit
  830 VDU 23;8202;0;0;0;
  840 DEL=120
  850 Score=0
  860 Hi=10
  870 end=FALSE
  880 ENDPROC
  890
  900
```

ECOLOGICAL DISASTER

Here's your chance to make decisions of national importance. The program explains the starting scenario. See if you can get the lake to survive longer than 10 weeks.

```
   10 REM Ecological Disaster
   50 REM (c) A. Goliner 1983
   60 REM
   70
   80 MODE 7
   90
  100 PROCinit
  110
  120 REPEAT
  130 PROCloop
  140 PROCend
  150 UNTIL 0
  160 END
  170
```

```
  180
  190 DEF PROCloop
  200 CLS
  210 PROCtitle(0,"       Ecological Disa
ster")
  220 PRINT'''"You are the president's a
dvisor on nat- ural resources...And part
 of your job is to stock a new lake in Mi
chigan with    fish and eels..."
  230 PRINT'"The eels only eat the fish.
.and the fishare are only eaten by the e
els"
  240 PRINT'"You must select starting nu
mbers of fish and eels which will ensure
the longest possible survival of the la
ke."
  250 PRINT''flalet$;"PRESS ANY KEY TO S
TART...."
  260 DUMMY=GET
  270 CLS
  280 PROCtitle(0,"       Ecological Disa
ster")
  290 PRINT'''"Enter the starting No. of
 fish.."
  300 INPUT FISH
  310 PRINT'''"Enter the starting No. of
 eels.."
  320 INPUT EELS
  330 CLS
  340 PROCtitle(0,"       Ecological Disa
ster")
  350 WEEK=0
  360 REPEAT
  370 FOR G=0 TO 30
```

```
  380 A$=CHR$(129+RND(6))+"THE LAKE IS E
VOLVING.........."
  390 PROCtitle(21,A$)
  400 SOUND 1,-7,70+(1.5*G),2
  410 NEXT
  420 WEEK=WEEK+1
  430 PRINT TAB(3,3)"REPORT TO THE N.R.A
....."
  440 PRINT TAB(3,4)"At the end of week
";WEEK
  450 CON=10
  460 SEELS=(EELS-(FISH/CON))
  470 IF SGN(SEELS)=-1 THEN SEELS=0
  480 EELS=EELS-SEELS
  490 KFISH=EELS*CON
  500 FISH=FISH-KFISH
  510 EELS=EELS+((EELS/2)*3)
  520 FISH=FISH+((FISH/2)*(CON/2))
  530 PRINT''INT(FISH);" Fish "
  540 PRINT INT(EELS);" Eels "
  550 DUMMY=GET
  560 UNTIL EELS<2 OR FISH<2 OR EELS>200
0 OR FISH>20000
  570 CLS
  580 ENDPROC
  590
  600
  610 DEF PROCend
  620 PROCtitle(0,"       Ecological Disa
ster")
  630 PRINT''"It's all over now, Natural
 Resources     Advisor..."
  640 IF WEEK>BESTWEEK THEN BESTWEEK=WEE
```

```
  650 PRINT''" The Lake stayed alive for
 ";WEEK;" weeks,"''"so your best to date
is ";BESTWEEK;" weeks...."
  660 DUMMY=GET
  670 CLS
  680 ENDPROC
  690
  700
  710 DEF PROCinit
  720 ON ERROR IF ERR=17 PROCe:END : ELS
E REPORT:PRINTERL
  730 VDU 23;8202;0;0;0;
  740 redlet$=CHR$ 129
  750 newbac$=CHR$ 157
  760 yellet$=CHR$ 131
  770 doulet$=CHR$ 141
  780 flalet$=CHR$ 136
  790 BESTWEEK=0
  800 ENDPROC
  810
  820 DEF PROCtitle(D,A$)
  830 FOR DOWN=D TO D+1
  840 PRINT TAB(0,DOWN)redlet$;newbac$;y
ellet$;doulet$;A$
  850 NEXT
  860 ENDPROC
  870
  880
  890 DEF PROCe
  900 VDU 22,7
  910 PROCtitle(0,"            Program End
")
  920 END
```

APPENDIX: USING MATH

Here is a summary of the mathematical symbols on your computer.

Usual symbol:		Computer symbol:
+	(plus)	+
−	(minus)	−
X	(multiply)	*
	(divide)	/
m^n	(raise to power)	m↑n

The mathematical functions are:

Computer word:	Meaning:
ASN	ARCSINE
ACS	ARCCOSE
ATN	ARGTANGENT
SIN	SINE
COS	COSINE
TAN	TANGENT
INT	Reduce to next lowest whole number
SGN	Sign (returns −1 if negative, Ø is zero, 1 if positive)
ABS	Returns number without its sign (so ABS − 5 is 5)
SQR	Square root
LN	Natural log
PI	The constant 3.14...
VAL	Returns the numerical value of a string
DEF FN	Defines a function, to be called up with FN

GETTING STARTED ON YOUR BBC MICRO is one of a series designed by experts and put together by actual users which will allow the first-time buyers of a personal computer to make effective, creative and constructive use of their new acquisition with the minimum of time-wasting false starts and the maximum of personal satisfaction and fulfilment. Free of jargon, this is a simple to read, easy to use, step by step guide to proficiency in the use of the BBC MICRO which will enable readers to obtain the maximum of results from their machine in the least possible time.

Also in this series

GETTING STARTED ON YOUR ZX81
GETTING STARTED ON YOUR SPECTRUM
GETTING STARTED ON YOUR TRS 80/DRAGON
GETTING STARTED ON YOUR COMMODORE/VIC 20
GETTING STARTED ON YOUR ATARI
GETTING STARTED ON YOUR ORIC

Futura Publications
Non-fiction
0 7088 24439
U.K. £2.95

GB £ NET +002.95
ISBN 0-7088-2443-9
00295
9 780708 824436