FUTURA

GETTING STARTED ON YOUR COMMODORE/ VIC 20

Tim Hartnell and Mark Ramshaw

Also in this series

GETTING STARTED ON YOUR ORIC
GETTING STARTED ON YOUR ATARI
GETTING STARTED ON YOUR BBC MICRO
GETTING STARTED ON YOUR DRAGON
GETTING STARTED ON YOUR ZX81
GETTING STARTED ON YOUR SPECTRUM

Tim Hartnell & Mark Ramshaw

Getting Started on Your Vic 20

Futura
Macdonald & Co
London & Sydney

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 2445 5

Reproduced, printed and bound in Great Britain by
Hazell Watson & Viney Ltd,
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

This book is dedicated to Ellen McLean, Esther Ramshaw, Mum and Dad.

Contents

Introduction

A note on the authors

Chapter One — DISCOVERING THE KEYBOARD
The shift keys; delete; Commodore shift; normal shift; control key; the cursor; quotes/insert mode; abbreviations

Chapter Two — PUTTING THINGS ON THE SCREEN
PRINT; strings; the first program ('Jack and Jill');REM

Chapter Three — RINGING THE CHANGES
Clearing the screen; input prompts; editing; using the printer

Chapter Four — DESCENT INTO CHAOS
Random events; generating random numbers; seeding the random numbers generator; FAST FOOD program; speed

Chapter Five — ROUND AND ROUND WE GO
FOR/NEXT loops; use of STEP; nested loops; MULTIPLICATION TABLES program; CODE BREAKER; multi-statement lines

Chapter Six — CHANGING IN MID-STREAM
GOTO; IF/THEN; GO SUB; computed destinations; dice rolls program

Chapter Seven GETTING INTO THE MUSIC BUSINESS
Registers; a sound subroutine; SOUND ADVICE program

Chapter Eight MAKING COMPARISONS
BRICKBAT program

Chapter Nine TWO GAMES AND A SPEED TEST
SOLITAIRE program; REACTION TEST program; HASAMI SHOGI program; introduction to structured programming

Chapter Ten STRINGING ALONG
The character set; ASC and CHR$; KEYBOARD INSTRUCTOR program; string slicing; concatenation; NAME PYRAMID program; making string comparisons

Chapter Eleven READING DATA
READ, DATA and RESTORE; GALACTIC GROCER program

Chapter Twelve WONDERS OF THE VIC CHIP
An introduction to binary; PEEK and POKE; AND, OR and NOT; the VIC registers; living more colourfully; FAST FOOD WITH COLOUR program; background and borders; POKEing to the screen; DUCK SHOOT program

Chapter Thirteen GETTING LISTED
arrays and DIM; single-dimensional and multi-dimensional arrays; DRAGON'S LAIR program; FULL FATHOM FIFTY program

Chapter Fourteen — LET THE GOOD GRAPHICS ROLL
user defined graphics; multi colour mode; animation; high resolution graphics

Chapter Fifteen — MORE GREAT GAMES
BAGATELLE; REPTILE; STARS AND STRIPES; CHECKERS; FOLLOW ME; ECOLOGICAL DISASTER; CASUALTY; SLALOM; OPERATION DESPERATION

APPENDIX — USING MATHS
Summary of symbols used

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 VIC 2Ø, we hope to share with you — step by simple step — the secrets of computer 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 VIC 2Ø. Representing the leading edge of computer technology, the VIC 2Ø is a computer that was designed to be *friendly*, and easy and rewarding to use. You've picked the best computer, and this book is the most direct route to programming expertise.

Come on and join us now, as together we discover how you can make the most of your VIC 2Ø.

Tim Hartnell
London, July, 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, ZX81 and ZX Spectrum. He is founding editor of the British bimonthly magazine ZX Computing, and is involved in user-group activities in the US and UK.

Mark Ramshaw is a schoolboy, with an active interest in VIC games. He has already written one book of VIC games, and is co-author of a games book for the Commodore 64.

CHAPTER ONE
DISCOVERING THE KEYBOARD

You've just bought a great computer, and in this book we're going to show you how to get the most out of it. Don't worry if this is the very first computer you have ever owned. We're going to take things slowly, and in small steps, so you'll have no trouble in keeping up with us.

READ THE BOOK WITH YOUR COMPUTER TURNED ON

It is vital that you have the computer turned on at all times when reading this book (or at least the first time you work through it). This is an 'action book', and unlike a novel, it is not designed simply to be read. This book is like a book on how to drive. You could never learn about changing gears, and how it feels to handle an auto in traffic, without actually going out and taking command of a vehicle. So it is with your computer and this book. Try out each new command and function when it is explained to you, experiment with the games and other programs, and you'll find you're learning to program without even trying.

THE KEYBOARD

First we have to find our way around the keyboard. It looks pretty frightening the first time you look at it, but once you've

used it for an hour or so, you'll be surprised at how familiar it will become to you.

THE SHIFT KEYS

The three most important keys on the keyboard are located in the bottom left hand corner (the key marked shift, and the key marked with the Commodore logo) and in the bottom right hand corner (marked shift). The two keys marked shift act the same. Find those three keys.

Now connect up your computer as explained in the manual, and get the Commodore statement at the top of the screen. Press any of the keys and the main symbol printed on the key will appear on the screen.

DELETE

By now the screen may be looking pretty crowded, and something like a disaster area. If you look to the top right hand corner of the keyboard you'll see a key marked INST, DEL — the delete key. The important word for now is the word DEL. If you hold down the DELETE key, and *continue to hold it down*, you'll find it will move across the printed material to the left, erasing it as it goes.

Another way of clearing up 'rubbish' like this from the bottom of the screen is to turn off the power for a short while, but as this causes the computer to forget the program which is in the computer, this is not advised.

COMMODORE SHIFT

If you hold down the Commodore shift key and continue to hold it down as you press another key, you'll find you get the left hand graphic character written on the front of the key appearing on the screen. For example, if you press the F key with the Commodore key, you get the bottom left hand corner graphic symbol printed on the screen. In the same way, pressing

the G key and holding down the Commodore shift will get you the left hand line graphic.

Try pressing a number of different keys. You'll see, in each case, that you get the graphic symbol on the left of the key printed on the screen. The Commodore shift, when used with a key without a graphic symbol on the left, acts like an ordinary shift key.

NORMAL SHIFT

Holding down the shift key and a normal key with a right hand graphic symbol will print the right hand graphic symbol on the screen (in much the same way as the Commodore shift produced the left hand graphic symbols). When used with the keys without graphic symbols on them, it will print the symbol above the main character on the key, e.g. shift with 2 gives the quotes symbol on the screen.

Pressing the shift and Commodore keys together will switch between upper and lower case modes. Your VIC is in upper case mode when switched on. In lower case mode, all program statements must be typed in lower case. The Commodore key will still print graphic symbols, but the normal shift will give capital letters instead of right hand graphic characters.

There is still more. Don't worry if this seems confusing. You'll be surprised at how quickly it will fall into place within half an hour or so of playing at the keyboard.

CONTROL KEY

The control key is the one labelled CTRL at the left of the keyboard. Pressing this with one of the numeric keys will give the colour typed on the key, or in the case of the 9 and Ø keys, will give reverse on and reverse off modes. All future text will be printed in that colour until it is changed. Reverse on means that the letter's foreground and background colours are swapped —

try it and see. Reverse off goes back to normal text. Remember, you can't see white writing on a white background.

THE CURSOR

The small square that flashes on and off on the screen is known as the cursor. This shows you where the next thing that you type will be printed on the screen. The two keys at the bottom right of the keyboard are the cursor control keys. They move the cursor down and right, and up and left with shift key. This is useful because it means that if you have typed a command and you want the computer to repeat it again, all you have to do is move the cursor back up to the command you typed and press return again. The computer will think that you have typed the line again.

At the top right of the keyboard is the INST/DEL key. We have already seen how this deletes the character to the left of the cursor. When used with the shift key, it inserts a space to the left of the cursor to allow you to enter more text into an existing line. Remember the computer will not accept a line consisting of more than eighty eight characters (four lines). To the left of this key is the CLR/HOME key. Pressing this makes the cursor go to the top left hand corner of the screen. Pressing it with the shift key clears the screen as well.

QUOTES/INST MODE

After typing a quotes character or inserting one or more spaces, the computer goes into a special mode which is mainly used in programs to format displays. When in this mode, pressing a colour, reverse on, reverse off, clear screen, cursor home or the cursor control keys will not perform the normal function. Instead, a weird graphic character is printed, each key having a different character. They also have numbers corresponding to them known as CONTROL CODES. When used in a program these control symbols/codes can be used to change the colour, format, etc. of the printout. The cursor can be moved around using the cursor controls inside a program to allow the

position where the next piece of writing will take place to be altered. This effect can be achieved by putting the symbols inside quotes or by using the statement known as CHR$(n) (where n is the control code needed). This means 'character whose code is', and is quite often used in a PRINT statement. More on this later. This mode is ended when another quotes symbol is typed, return is pressed or all the inserted spaces have been used. Note that delete will not work in INSERT mode.

Here is a program to show you how the control symbols/ codes would be used in a print statement. It is intended to be used for reference, not to be typed in.

```
10 PRINTCHR$(144);"CHANGE TO BLACK"
15 PRINT"           - CTRL 1"
20 PRINT CHR$ (5);"CHANGE TO WHITE"
25 PRINT"           - CTRL 2"
30 PRINT CHR$(28);"CHANGE TO RED"
35 PRINT"           - CTRL 3"
40 PRINTCHR$(159);"CHANGE TO CYAN"
45 PRINT"           - CTRL 4"
50 PRINTCHR$(156);"CHANGE TO PURPLE"
55 PRINT"           - CTRL 5"
60 PRINT CHR$(30);"CHANGE TO GREEN"
65 PRINT"           - CTRL 6"
70 PRINT CHR$(31);"CHANGE TO BLUE"
75 PRINT"           - CTRL 7"
80 PRINTCHR$(158);"CHANGE TO YELLOW"
85 PRINT"           - CTRL 8"
90 PRINT CHR$(18);"REVERSE ON"
95 PRINT"           - CTRL 9"
100 PRINTCHR$(146);"REVERSE OFF"
105 PRINT"          - CTRL 0"
110 PRINT CHR$(20);"DELETE"
115 PRINT"          - - INST/DEL"
120 PRINTCHR$(148);"INSERT"
```

```
125 PRINT"          ■ - SHIFT INST/DEL"
130 PRINT CHR$(19);"CURSOR HOME"
135 PRINT"          ▒ - CLR/HOME"
140 PRINTCHR$(147);"CLEAR SCREEN"
145 PRINT"          ▒ - SHIFT CLR/HOME"
150 PRINT CHR$(17);"CURSOR DOWN"
155 PRINT"          ▒ - CRSR DOWN"
160 PRINTCHR$(145);"CURSOR UP"
165 PRINT"          ▒ - SHIFT CRSR DOWN"
170 PRINT CHR$(29);"CURSOR RIGHT"
175 PRINT"          ▒ - CRSR RIGHT"
180 PRINTCHR$(157);"CURSOR LEFT"
185 PRINT"          ▒ - SHIFT CRSR RIGHT"
190 PRINT CHR$(14);"LOWER CASE MODE"
195 PRINT"          - - COMMODORE AND SHIFT
KEYS"
200 PRINTCHR$(142);"UPPER CASE MODE"
205 PRINT"          - - COMMODORE AND SHIFT
KEYS"
210 PRINTCHR$(133);"F1"
215 PRINT"          ▒ - F1"
220 PRINTCHR$(134);"F3"
225 PRINT"          ▒ - F3"
230 PRINTCHR$(135);"F5"
235 PRINT"          ▒ - F5"
240 PRINTCHR$(136);"F7"
245 PRINT"          ▒ - F7"
250 PRINTCHR$(137);"F2"
255 PRINT"          ▒ - SHIFT F1"
260 PRINTCHR$(138);"F4"
265 PRINT"          ▒ - SHIFT F3"
270 PRINTCHR$(139);"F6"
275 PRINT"          ▒ - SHIFT F5"
280 PRINTCHR$(140);"F8"
285 PRINT"          ▒ - SHIFT F7"
```

The function keys have no effect on the computers output. They can be checked from within a program, e.g. for use as controls in a game.

Note that delete, upper case mode and lower case mode cannot be used in quotes but can be used in a CHR$(n) statement.

The previous program showed how the symbols look in upper case mode, the symbols in lower case mode are shown here.

```
10 Printchr$(144);"change to black"
15 Print"           - ctrl 1"
20 Print chr$ (5);"change to white"
25 Print"           - ctrl 2"
30 Print chr$(28);"change to red"
35 Print"           - ctrl 3"
40 Printchr$(159);"change to cyan"
45 Print"           - ctrl 4"
50 Printchr$(156);"change to purple"
55 Print"           - ctrl 5"
60 Print chr$(30);"change to green"
65 Print"           - ctrl 6"
70 Print chr$(31);"change to blue"
75 Print"           - ctrl 7"
80 Printchr$(158);"change to yellow"
85 Print"           - ctrl 8"
90 Print chr$(18);"reverse on"
95 Print"           - ctrl 9"
100 Printchr$(146);"reverse off"
105 Print"          - ctrl 0"
110 Print chr$(20);"delete"
115 Print"          - inst/del"
120 Printchr$(148);"insert"
125 Print"          - shift inst/del"
130 Print chr$(19);"cursor home"
135 Print"          - clr/home"
```

```
140 Printchr$(147);"clear screen"
145 Print"          - shift clr/home"
150 Print chr$(17);"cursor down"
155 Print"          - crsr down"
160 Printchr$(145);"cursor uP"
165 Print"          - shift crsr down"
170 Print chr$(29);"cursor right"
175 Print"          - crsr right"
180 Printchr$(157);"cursor left"
185 Print"          - shift crsr right"
190 Print chr$(14);"lower case mode"
195 Print"          - - commodore and shift
keys"
200 Printchr$(142);"uPPer case mode"
205 Print"          - - commodore and shift
keys"
210 Printchr$(133);"f1"
215 Print"          - f1"
220 Printchr$(134);"f3"
225 Print"          - f3"
230 Printchr$(135);"f5"
235 Print"          - f5"
240 Printchr$(136);"f7"
245 Print"          - f7"
250 Printchr$(137);"f2"
255 Print"          - shift f1"
260 Printchr$(138);"f4"
265 Print"          - shift f3"
270 Printchr$(139);"f6"
275 Print"          - shift f5"
280 Printchr$(140);"f8"
285 Print"          - shift f7"
```

ABBREVIATIONS

The following is a list of abbreviations for many of the program statements and functions. These may seem strange but you will get to know them later on. The abbreviations can be typed in instead of the full command, but note that some functions include the first bracket (parenthesis) and some do not (all functions are followed by a number in brackets in a BASIC program).

Command	Abbreviation	Command	Abbreviation
AND	A shift N	PRINT #	P shift R
NOT	N shift O	READ	R shift E
CLOSE	CL shift O	RESTORE	RE shift S
CLR	C shift L	RETURN	RE shift T
CMD	C shift M	RUN	R shift U
CONT	C shift O	SAVE	S shift A
DATA	D shift A	STEP	ST shift E
DEF	D shift E	STOP	S shift T
DIM	D shift I	SYS	S shift Y
END	E shift N	THEN	T shift H
FOR	F shift O	VERIFY	V shift E
GET	G shift E	WAIT	W shift A
GOSUB	GO shift S	ABS	A shift B
GOTO	G shift O	ASC	A shift S
INPUT #	I shift N	ATN	A shift T
LET	L shift E	CHR$	C shift H
LIST	L shift I	EXP	E shift X
LOAD	L shift O	FRE	F shift R
NEXT	N shift E	LEFT$	LE shift F
OPEN	O shift P	MID$	M shift I
POKE	P shift O	PEEK	P shift E
PRINT	?	RIGHT$	R shift I
RND	R shift N	STR$	ST shift R
SGN	S shift G	TAB(	T shift A
SIN	S shift I	USR	U shift S
SPC(	S shift P	VAL	V shift A
SQR	S shift Q		

Note that PRINT # and INPUT # are not the same as PRINT and INPUT. '? #' will not work for PRINT #.

CHAPTER TWO
PUTTING THINGS ON THE SCREEN

We pointed out in chapter one, when talking about the keyboard, that the shift keys (Commodore and normal shift) were the most important ones on the keyboard. We're now going to introduce the third most important one, RETURN. Whenever you're typing on the screen 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 on the screen, with the cursor flashing patiently beside it, until you press RETURN.

The same goes for the PRINT 2 you 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 below your command followed by 'READY.'. 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 (you'll find the plus key to the right of the number keys on the top row of the keyboard):

```
PRINT 5+3
```

You should see the figure 8 appear underneath. 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 in this, and press RETURN to see subtraction (and PRINT) at work (the − sign is to the right of the + key):

```
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 (/). The asterisk (*) for multiplication is just below the − sign and the slash (/) for division is to the left of the right hand shift key.

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 a division. Type it in, then press RETURN to see the computer evaluate it:

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

This seems pretty simple. Just type in 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
```

That doesn't look too good. Instead of the word *TESTING* we've got a 0 (zero). The computer thought you were referring to a variable with the name 'TESTING'.

We won't try to explain what has happened 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 very stupid 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 on the screen. 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 *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 into the computer, then type in PRINT (or type in ? , the abbreviation for PRINT), get the opening quote marks from the 2 key (with the shift key) 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 line being executed, and the writing being printed, the computer stores the line and does not act on it. 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 20) and press RETURN once you have it all in place. If the line is not the same as printed here move the cursor back up to it and DELETE and INSERT as necessary as you were told in the first chapter. e.g. if you have typed JCK AND JILL, move the cursor over the C and press shift INST then press A to insert an A. If you had typed O instead of A for example, you could move the cursor over the O and retype A instead of deleting and then inserting. If you type a new line with the line number of an old one, it will replace the old line. You can also type in a new line, and the computer will insert it in the correct plce in your program. To see this in action, type the following line then press RETURN. Then type LIST, this will list your program so you can examine it:

```
25 REM A LINE IN THE MIDDLE
```

Now, the program will look like this:

```
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 a 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, I 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 in the word NEW. Get the computer to act on that command by pressing RETURN. The computer will reply with the word 'READY.'. Your computer is now empty and ready and waiting for its next task.

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 "VIC"
35 PRINT
40 PRINT "23+34=";23+34
45 PRINT
50 PRINT 2*3
55 PRINT
60 PRINT 315
65 PRINT
70 PRINT "THE ANSWER IS";23-7/6
```

You should get something like this on the screen:

```
 1              2

 1  2  3

VIC

23+34= 57

 6
```

```
 243

THE ANSWER IS 21.8333333
```

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

Firstly, as in the 'Jack and 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, when 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 (which you will find on the bottom row of the keyboard). This has the effect of getting the computer to print the first number (1) on the left hand side of the screen, and the second number (2) 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 divides the screen into two, printing that which follows the comma starting at the next available half of the screen. If the line had read PRINT 1,2,3 it would have printed the 1 at the start of the first line, the 2 at the start of the second half of the first line, and the 3 *underneath* the one, because this was the 'next available half screen'. If you're not too clear what we mean by this, try it yourself.

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. Instead of separating the output of the numbers as the comma did, you'll see that it causes them to be printed with two spaces between each one. You use the semicolon (;) when you want some printed material to follow other printed material without a break.

Line 3Ø is the word VIC and this is a ..... If you mentally said *string* when you came to those dots, then you're learning well. This 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 result 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), so it assumed it was equal to zero. 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 60.

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.

Numbers are printed with a space following them where as strings are not. Positive numbers have a space before them as well, but this is replaced by a ' – ' for negative numbers. This is why there are two spaces between two positive numbers printed by the computer: the trailing space of the first number and the preceding space of the second. Strings are printed with no spaces between them unless they contain spaces themselves.

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 control code, number 147.

CLEAR THE SCREEN

Enter the following program into your computer and run it. You'll find the dollar sign after the A in line 2Ø on the 4 key. You get the dollar sign by holding down the shift key while pressing the four key.

```
10 PRINT "TESTING""
20 INPUT A$
30 PRINT CHR$(147)
```

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

```
TESTING
```

However, underneath this you'll see a question mark and the cursor blinking merrily to itself:

```
? ■
```

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 RETURN, 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 PRINT CHR$(147). 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 RETURN, you can write a program which clears the screen automatically, as our next example demonstrates. Enter this next program into your computer, press RUN, then RETURN, and sit back for the Amazing Flashing Word demonstration.

```
10 PRINT "AUTOTESTING"
20 FOR A=1 TO 100
30 NEXT A
40 PRINT CHR$(147)
50 FOR A=1 TO 100
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 100, before moving on. As you can see, it does this counting pretty quickly.

So, it waits for a moment while counting from one to 100. Then it comes to line 40, 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 from one to 100 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 pressing RUN, followed by RETURN. But RUN is a command, 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 100, clearing the screen, counting to 100 again, and then hitting RUN so that the whole dance starts over.

EDITING

As was discussed in chapter one, your VIC is provided with a screen editor which makes it very simple to change the contents of lines within a program. NEW the current program, then enter the following program into your computer. *Do not type RUN.* We want to explain something about the program before you do:

```
10 REM AN EDIT TEST
20 PRINT "TEST AGAIN"
30 PRINT "AND AGAIN"
```

As always, the cursor is flashing where the computer expects its next input. Move the cursor around the screen using the cursor control keys as described in chapter one. Note that if you keep them held down, they repeat, this is also true of the insert and delete keys and the space bar. NOTE: you can make all the keys repeat with the command POKE 650,255. You can return to normal by typing POKE 650,0. Remember to press RETURN in each case after each line. If you clear the screen, you can obtain a listing of your program with the command LIST (followed by RETURN of course). Using the control keys, move the cursor to the beginning of line 10. You can now change, or edit, this line. Using the cursor right key, move the cursor along line 10, noticing how the symbols underneath it flash between normal and reverse mode. It should look something like:

```
10 REM AN EDIT TEST
```

Move the cursor right to the end of the line and use the DELete key to erase all the letters *after* the word REM. Now type in new words, so the line reads:

```
10 REM AN EDIT CHANGE
```

When you have done this, press RETURN as if you had just finished entering a standard line of program. Then move the cursor down to below the last line of the program. Type LIST and you should see the program with line 10 changed. Before you read on, spend some time changing lines 10, 20 and 30,

until you're completely familiar with the use of the cursor control keys.

GETTING THE PROGRAM BACK

You'll notice that, after you're RUN a program, the listing of the program may have scrolled off the top of the screen. Scroll is a term used to describe what happens when the computer prints past the bottom line of the screen and the display moves up a line to compensate for this. This results in the loss of the top line of the display. As was mentioned before, typing LIST will give you a listing of your program. Try it now with the program you have in your computer.

There is no reason why you must LIST from the start of a program. This is a very useful feature when you have a program which is so long that not all of it will fit on the screen at once. Type in LIST 2Ø and press RETURN. You'll see line 2Ø appear on the screen. You may also type in LIST 2Ø–, this lists from line 2Ø onwards; LIST –2Ø, this lists up to line 2Ø; and LIST 1Ø–2Ø, this lists all the lines between 1Ø and 2Ø. Of course you can use any line numbers with list, so long as they are not negative or too big. If the line number specified after the word LIST does not exist, the computer will go to the next available number, and LIST from there.

USING THE PRINTER

When using the printer, you must make sure it is switched on otherwise you might loose your program altogether from the VIC and have to switch off and on again. First of all, after switching your printer on, you must open a file to the printer. This is done with the command: OPEN 1,4,Ø the first number is known as the logical file number — it can be any number between 1 and 255, but you must always use the same number when you address the printer. The second number is the device number and can be either 4 or 5, depending on the setting of the switch at the back of the printer. The third number is known as the secondary address and can be either Ø or 7. When Ø, printing

occurs in normal mode, and when 7, printing occurs in lower case mode. If you simply want to print something on the printer, you can use the command: PRINT#1, data. The number is the same number as the logical file number in the open command. Anything after the comma, is printed on the printer instead of the screen as in a normal PRINT statement. Note that control codes do not work on the printer, it has its own special control codes which are listed in the printer manual. So a command like: PRINT#1, "HALLO" would print 'HALLO' to the printer (assuming a file has already been opened). If you want to list your program, or simply get output on the printer from a normal PRINT command, you can use the command: CMD 1. This sends all subsequent output from the computer to the printer instead of the screen. The number is, once again, the logical file number. From now on, any lists or prints should go to the printer. If you want to bring output back to the screen use the PRINT# with the logical file number and no data. After this, only data printed using the PRINT# command will be sent to the printer. When you are ready to stop printing altogether, you must close the file with the command: CLOSE 1. This sets the computer as if you had never opened the file. The number is the logical file number used in all previous work. It is a good idea to type PRINT#1 without data before closing the file. Note that the number is, yet again, the infamous logical file number. The next example will open the file whose number is 2 and demonstrate what you have learnt:

```
10 REM PRINTER
20 OPEN 2,4,0
30 PRINT#1,"HI THERE,I'M VIC"
40 PRINT#1,CHR$(13)
50 PRINT#1,"THIS IS THE VIC PRINTER"
60 CMD2
70 PRINT"THIS IS FROM A NORMAL"
80 PRINT"STATEMENT"
90 PRINT#2,"BUT THIS IS FROM A"
100 PRINT#2,"PRINT# STATEMENT"
```

```
110 PRINT#2
120 CLOSE 2
```

```
HI THERE,I'M VIC


THIS IS THE VIC PRINTER

THIS IS FROM A NORMAL
STATEMENT
BUT THIS IS FROM A
PRINT# STATEMENT
```

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, and in due course, all of them will be covered. 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 programs 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 the 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 word to do this is RND(1). The number in the brackets can be any positive number but there must be a number there. It is usually Ø or 1.

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 GOTO 10
```

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

```
.185564016
.0468986348
.827743801
.554749226
.897233831
.572916248
.838893164
.931229627
```

```
.138382009
.97293994
.776433747
.417980108
.829277551
.809331095
.967128073
.426060658
.364234364
.770340712
```

As you can see, RND 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. Enter the next program and run it. You'll get the parenthesis on the 8 and 9 keys (and you get them by holding down the SHIFT key while pressing the 8 and 9). There is a single space between the quote marks at the end of line 10. You get the space just by pressing the space bar.

```
10 PRINT INT(RND(1)*6+1);" "
20 GOTO 10
```

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

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

Even though we can create vast series of numbers between 1 and 6 with a program like this, it is not particularly interesting. And, if you ran the program over and over again, you might find that the sequence of numbers may start to become a little familiar. The random numbers may stop looking so random.

This is because the computer does not really generate random numbers, but only looks as if it is doing so. Inside its electronic head, the computer holds a long, long list of numbers, which it prints in order when asked for random numbers. The list is so long, it is almost impossible to see any pattern or repetition in it. However, this is often not quite good enough. Some programs demand numbers which are as close as possible to genuinely random.

SEEDING THE RANDOM NUMBER GENERATOR

Fortunately, we can do this on our computer. This is done by putting a negative number in between the brackets in the RND statement. What it does is choose a starting position somewhere in this list, instead of starting at the beginning of the list. There is a number stored in the computer called TI. This is the time the computer has been on in 60ths of a second. Thus typing LET A = RND(−TI) will choose a starting position in the list unknown to anyone. We can seed the random number generator in three ways:

1 Just enter LET A = RND(−TI) *before* you run a program, which will seed the random number generator (by 'seed' we mean start it off in a random position within the list).

2 Include the above line somewhere in a program (preferably at the beginning), so it occurs every time you run a program.

3 Include it within a program, and get the program to ask for a 'seed'. A seed, in this case, is just a number the user enters. Each time you enter the same seed, you'll get the same result. Be careful to make the seed negative if it is positive and if it is too small then get the program to ask for another seed. The lines might look something like this (the statements will be explained later):

```
50 INPUT "SEED";SE:IF SE>0 THEN SE=-SE
55 SE=RND(SE)
```

Now this possibly seems confusing. But, don't worry, it will become clear, and the next two programs are designed to help make it clear.

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 LET A=INT(RND(1)*4)+1
40 PRINT"YOU'VE ORDERED"
50 IF A=1 THEN PRINT"A HAMBURGER WITH THE LOT"
60 IF A=2 THEN PRINT"A LARGE FRENCH FRIES"
70 IF A=3 THEN PRINT"A SERVE OF RIBS"
80 IF A=4 THEN PRINT"TWO HOT DOGS WITH KETCHUP"
90 PRINT
100 GOTO30
```

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

```
YOU'VE ORDERED
TWO HOT DOGS WITH KETCHUP

YOU'VE ORDERED
TWO HOT DOGS WITH KETCHUP

YOU'VE ORDERED
A LARGE FRENCH FRIES

YOU'VE ORDERED
A SERVE OF RIBS
```

Notice how, as was mentioned previously, the screen scrolls (or rolls upwards) when the writing gets to the bottom of the screen. You may slow this 'scrolling' down by keeping a finger on the CTRL key at the far left of the keyboard while the program is running. To stop the program use the key marked RUN/STOP just below the CTRL key. The program will stop and the computer will say something like 'BREAK IN LINE 30'. Incidentally you can also stop the program by keeping one finger on the RUN/STOP key and another on the key marked RESTORE at the right hand side of the keyboard. This restores your VIC without losing your program. Notice how the program sets the letter A to the value of the random number in line 30. 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 30) the computer has *assigned* the value of the random number to the variable A. And, as you can see in line 50, 60, 70 and 80, the value assigned to A determines which food order you place. Read this over if it seems incomprehensible the first time.

PLANTING A SEED

Now, we are going to modify the program, using the cursor control keys which we discussed earlier. Change your original program, so it looks like this, modifying some lines and adding others. You'll find the single apostrophe (') on the 7 key:

```
10 REM FAST FOOD
15 PRINT "ENTER RANDOMIZE"
20 INPUT "SEED";A:IF A>0 THEN A=-A
25 A=RND(A)
30 LET A=INT(RND(1)*4+1)
35 PRINT "SEED";A
40 PRINT "YOU'VE ORDERED"
50 IF A=1 THEN PRINT"A HAMBURGER WITH THE LOT"
60 IF A=2 THEN PRINT"A LARGE FRENCH FRIES"
70 IF A=3 THEN PRINT"A SERVE OF RIBS"
80 IF A=4 THEN PRINT"TWO HOT DOGS WITH KETCHUP"
90 FOR I=1 TO 100:NEXT
100 GOTO 20
```

When you run this, the computer will stop and the words ENTER RANDOMIZE SEED will appear on the screen. This is an *input prompt*. When you place words in quote marks after the word INPUT, they appear on the screen to tell you what kind of input the computer expects. In this case, because the letter A after the semicolon at the end of line 20 does *not* have a dollar sign after it (that is, it is *not a string*), the computer is expecting you to enter a number. You can enter any number you like, and you'll soon discover which numbers to enter to get the food you want.

The display on the screen looks like this when the program is running, with the random number generated in line 30 printed by line 35.

```
ENTER RANDOMIZE
SEED 4
YOU'VE ORDERED
TWO HOT DOGS WITH KETCHUP
SEED 1
YOU'VE ORDERED
A HAMBURGER WITH THE LOT
SEED 2
```

```
YOU'VE ORDERED
A LARGE FRENCH FRIES
SEED 2
YOU'VE ORDERED
A LARGE FRENCH FRIES
SEED 4
YOU'VE ORDERED
```

Notice how easy it was to modify the program. Have a look at line 90. This is called a LOOP and will be discussed in greater depth in the next chapter. All it does is count from the first number to the second, producing a delay. The bigger the second number the longer the delay. This is very useful for *pacing* the running of a program to suit you.

CHAPTER FIVE
ROUND AND ROUND WE GO

In this chapter we'll be introducing a very useful part of your programming vocabulary — FOR/NEXT loops. You'll recall that we mentioned FOR/NEXT loops when demonstrating clearing 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 200
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  12  13
```

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  7
69  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 10, producing this result:

```
 10  20  30  40  50  6
0  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
```

```
?NEXT WITHOUT FOR
 ERROR IN 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:

```
 1 TIMES 1 IS 1
 1 TIMES 2 IS 2
 1 TIMES 3 IS 3
 1 TIMES 4 IS 4
 1 TIMES 5 IS 5
 1 TIMES 6 IS 6
 1 TIMES 7 IS 7
 1 TIMES 8 IS 8
 1 TIMES 9 IS 9
 1 TIMES 10 IS 10
 1 TIMES 11 IS 11
```

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
```

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. (Note that the word BLACK in line 38Ø is printed in black and reversed by the use of control codes. These are used throughout the program. Note also the PRINT statements with nothing following them (e.g. line 2Ø). These make the computer miss a line, they also make the computer print in normal mode if it was in reverse before. The practical upshot of all this is that there is no need, in this instance, for a REVERSE OFF control code.

```
5 POKE36879,138
10 PRINTCHR$(147);"■**********************"
20 PRINT
30 PRINT
40 PRINT"     CODEBREAKER"
45 PRINT"  BY A.J.B. AND T.S."
50 PRINT"■■■■■■■■■■■■■■■■■■■■■■"
60 PRINT
70 PRINT"  WHEN YOU ARE TOLD TO";
80 PRINT"DO SO, ENTER A 4-DIGIT";
90 PRINT"NUMBER AND  THEN PRESS";
100 PRINT"RETURN. DIGITS CAN  BE";
110 PRINT"REPEATED  YOU  HAVE  8";
120 PRINT" GOES  TO  BREAK  THE ";
130 PRINT"DIFFICULT CODE."
140 PRINT:PRINT"**********************";
145 TI$="000000"
147 IFTI<420THEN147
150 PRINTCHR$(147)
160 DIMB(4)
170 DIMC(4)
180 LETH=0
190 FORA=1TO4
200 LETB(A)=INT(10*RND(1))
210 NEXTA
220 FORC=1TO8
230 PRINT"■ENTER GUES NUMBER ";C:INPUTX
240 IFX>9999THENGOTO230
250 PRINT
260 IFINT(X/1000)=0THENPRINT"0";
270 PRINTX;" ";
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 FORE=1TO4
370 IFD(E)<>B(E)THENGOTO420
380 PRINT"■BLACK";
390 B(E)=B(E)+10
400 D(E)=D(E)+20
410 H=H+1
420 NEXTE
430 IFH=4THEN630
440 FORF=1TO4
450 D=D(F)
460 FORG=1TO4
470 IFD<>B(G)THEN510
480 PRINT"■WHITE";
490 B(G)=B(G)+10
500 GOTO520
510 NEXTG
520 NEXTF
530 FORG=1TO4
540 IFB(G)<10THEN560
550 B(G)=B(G)-10
560 NEXTG
570 H=0
580 PRINT
590 NEXTC
600 PRINT
602 PRINT
603 PRINT"■YOU DIDN'T GET IT..."
610 PRINT"THE ANSWER IS ";
615 H=B(1)*1000+B(2)*100+B(3)*10+B(4)
616 PRINTH
620 GOTO999
```

```
630 PRINT
635 PRINT"■YOU GOT THE ANSWER IN ";C;" GOES"
999 END
```

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

```
 1111  ■■■■■■
ENTER GUESS NUMBER  6

 1222  ■■■■■■
ENTER GUESS NUMBER  7

 1333  ■■■■■■
ENTER GUESS NUMBER  8

 1444  ■■■■■■■■■■■■

YOU DIDN'T GET IT...
THE ANSWER IS  1974
```

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 140 print a number of asterisks to rule off the title and instructions, with blank lines printed by lines 20, 30, 60 and 250. After the title is printed by line 40, it is 'underlined' by a series of graphics available from the I key. You'll probably remember from chapter one how to get graphics. All you do is press the I key in conjunction withthe Commodore shift. It might be wise to type POKE 650,255 followed by RETURN before entering the program. This allows the keys to repeat.

Line 145 resets the VICs string time variable which counts in hours, minutes and seconds. Line 147 waits until the numeric time variable which counts in 60ths of a second is over 600 (i.e. 10 seconds). Two *arrays* are dimensioned in lines 160 and 170. 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 line 180. In line 410, 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 630 to print up the congratulations.

The lines from 190 to 210 work out the number which you will have to try and guess. Line 200 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 190 equals one the first time the loop is passed through, two the second time, and so on, so that the A in line 200 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 230 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 240 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 230 to accept another guess.

The next section of the program, right through to line 580, works out how well you've done, using a number of

FOR/NEXT loops (36Ø to 42Ø, 44Ø to 52Ø, 46Ø to 51Ø and 53Ø to 56Ø). Line 59Ø 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 23Ø, but 'falls through' line 59Ø to tell you that you have not guessed the code in time, and to tell you what it is. Line 61Ø prints out the code.

If you do manage to guess it, so that H equals four in line 43Ø, then the program jumps to line 63Ø to print out the congratulatory message.

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 slightly condensed version of CODEBREAKER produced by putting more than one statement on a line in places. Compare the two listings, and see how a little bit of space can be saved with multi-statement lines. Do *not* put a second statement in a line after the word IF (such as line 43Ø). The reason for this will be outlined in the next chapter.

```
5 POKE36879,138
10 PRINTCHR$(147);"[illegible]******************
*****"
20 PRINT:PRINT
40 PRINT"      CODEBREAKER":PRINT"  BY
 A.J.B AND T.S."
50 PRINT"[illegible]":PRIN
T
70 PRINT"  WHEN YOU ARE TOLD TO";
80 PRINT"DO SO, ENTER A 4-DIGIT";
```

```
90 PRINT"NUMBER AND  THEN PRESS";
100 PRINT"RETURN. DIGITS CAN  BE";
110 PRINT"REPEATED  YOU  HAVE  8";
120 PRINT" GOES  TO  BREAK  THE ";
130 PRINT"DIFFICULT CODE"
140 PRINT:PRINT"**********************
*"
145 TI$="000000"
147 IFTI<420THEN147
150 PRINTCHR$(147)
160 DIMB(4),C(4):LET H=0:
190 FORA=1TO4:LETB(A)=INT(10*RND(1)):
NEXT A
220 FORC=1TO8
230 PRINT"■ENTER GUESS NUMBER ";C:INP
UTX
240 IFX>9999THENGOTO230
250 PRINT
260 IFINT(X/1000)=0THENPRINT"0";
270 PRINTX;" ";
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:D(2)=Q:D(3)=R:D(4)=S
360 FORE=1TO4
370 IFD(E)<>B(E)THENGOTO420
380 PRINT"■BLACK";
390 B(E)=B(E)+10
400 D(E)=D(E)+20
410 H=H+1
420 NEXTE
430 IFH=4THEN630
440 FORF=1TO4
450 D=D(F)
```

```
460 FORG=1TO4
470 IFD<>B(G)THEN510
480 PRINT"■■WHITE";
490 B(G)=B(G)+10
500 GOTO 520
510 NEXTG
520 NEXTF
530 FORG=1TO4
540 IFB(G)<10THEN560
550 B(G)=B(G)-10
560 NEXTG
570 H=0
580 PRINT
590 NEXTC
600 PRINT"■■YOU DIDN'T GET IT..."
610 PRINT"THE ANSWER IS ";
615 H=B(1)*1000+B(2)*100+B(3)*10+B(4)
:PRINTH
620 STOP
630 PRINT"■■WELL DONE,CODEBREAKER"
635 PRINT"■YOU GOT THE ANSWER IN ";C;
" GOES"
```

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 GO TO command (it can be typed as either one or two words, the VIC will still recognise it) sends action through a program in which ever 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"
90 GOTO50
```

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 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
  THIS IS 50
THIS IS 20
```

The program starts at line 10, and finding GOTO 50 there, moves on to line 50 to print out "THIS IS 50". It then continues on to line 60, where it finds the command "GOTO 20". Without question, it zips back to line 20 to print out "THIS IS 20" then goes to line 30 which directs it to line 80. At line 80, it finds the instruction to print out "THIS IS 80" which it does. From there it gets to line 90, and finds "GOTO 50" 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 pair of words generally found together on the computer are IF and THEN. IF something is true, THEN do something else. IF you are hungry, THEN order a hamburger. IF you want some candy, THEN behave. IF something THEN do something. 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 LET A=INT(RND(1)*6+1)
110 IF A=1 THEN 30
120 IF A=2 THEN 40
130 IF A=3 THEN 50
140 IF A=4 THEN 60
150 IF A=5 THEN 70
160 IF A=6 THEN 80
```

After THEN if you want to jump to a line you can simply type the number of the line or type GOTO the number of the line. You will find later on that that other things can follow a THEN statement.

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

```
FIVE
FOUR
FIVE
FOUR
ONE
TWO
TWO
ONE
FOUR
```

```
FIVE
THREE
FIVE
SIX
SIX
SIX
FOUR
THREE
SIX
```

So we've look 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 ouside 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 it for you. The second time it is for itself. 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 I=1 TO 500:NEXT I:PRINT
20 FOR C=1 TO 2
30 GOSUB 100
40 IF C=1 THEN LET A=D
50 IF C=2 THEN LET 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 SUBROUTINE
```

```
110 LET D=INT(RND(1)*6)+1
120 IF C=1 THEN PRINT"I ROLLED A";D
130 IF C=2 THEN PRINT"YOU ROLLED A";D
140 FOR I=1 TO 300:NEXT I
150 RETURN
```

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

```
I ROLLED A 3
YOU ROLLED A 4
YOU WIN

I ROLLED A 6
YOU ROLLED A 4
I WIN

I ROLLED A 1
YOU ROLLED A 5
YOU WIN

I ROLLED A 4
YOU ROLLED A 4

I ROLLED A 3
```

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"
100 LET A=INT(RND(1)*6)+1
110 IF A=1 THEN GOSUB 30
120 IF A=2 THEN GOSUB 40
```

```
130 IF A=3 THEN GOSUB 50
140 IF A=4 THEN GOSUB 60
150 IF A=5 THEN GOSUB 70
160 IF A=6 THEN GOSUB 80
170 GOTO 100
```

Although THEN GOTO 30 can be abbreviated to THEN 30, after an IF statement, if you want to call a subroutine, the GOSUB command must be included after the THEN statement.

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 110 to 170 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"
100 LET A=INT(RND(1)*6)+1
110 ON A GOSUB 30,40,50,60,70,80
120 GOTO 100
```

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

The program can be further condensed with the use of multiple-statement lines, producing this effect:

```
10 REM DICE ROLLS
20 GOTO 100
30 PRINT "ONE":RETURN
40 PRINT "TWO":RETURN
50 PRINT "THREE":RETURN
60 PRINT "FOUR":RETURN
70 PRINT "FIVE":RETURN
80 PRINT "SIX"
100 LET A=INT(RND(1)*6)+1
110 ON A GOSUB 30,40,50,60,70,80
120 GOTO 100
```

Notice the word ON. What this does is to look at the variable after it, and then choose a line number to jump to, or execute a subroutine from depending on this number. This number must be between 0 and 255, otherwise the computer will stop with an error message. If the number is 1, the computer goes to the first line after the ON statement; if it is 2 the computer goes to the second line etc. If the number is 0 or higher than the number of lines given in the program line the computer continues onto the next line without jumping anywhere. This may sound a bit confusing so experiment for yourself by changing line 100 to set the variable A equal to various values such as:

100 LET A = 5

CHAPTER SEVEN
GETTING INTO THE MUSIC BUSINESS

Your VIC has an excellent sound facility. This is a great way to add life to your program. It is amazing what a little sound can do to enhance a program.

The only problem is, the VIC has no commands to control the sound so this must be done with a rather strangely named command POKE. This has two numbers after it separated by a comma. What it does is take the second number (which must be between Ø and 255) and place it in a memory location whose number is the first number (which must be between Ø and 65535). This command has a function related to it called PEEK. What this does is return the value of what is stored in the memory location whose number is in the brackets after the PEEK. So POKE 768Ø,32 will put 32 in memory location 768Ø and LET A = PEEK(768Ø) will give A the value of whatever is in memory location 768Ø. This may seem a little complicated, but you should understand it if you persevere.

The VIC's sound is controlled by five of these memory locations, because they are special they have a special name: REGISTERS. They are as follows:

Memory location 36874 — this is the alto voice (low).
Memory location 36875 — this is the tenor voice (medium).
Memory location 36876 — this is the soprano voice (high).

Memory location 36877 — this is the noise channel. This is useful for gun shots, explosions etc.

Memory location 36878 — this is the volume control. It can be any setting from 0 (no noise) to 15 (loud).

All of the first four registers can hold any number between 128 and 255. The higher the number, the higher the pitch. They can hold numbers less than 128 but this will not produce any noise. Notice how you can have up to three musical notes and one noise playing at the same time. You can POKE these registers directly or from within a program. Remembering that you must POKE the volume register with a number bigger than 0 to hear anything (and have your television turned up). Perhaps the best method for creating sound effects from within a program is to call a subroutine like the one listed below. Before you call it, put the value you want register 1 to be in variable S1, the value you want register 2 to be in variable S2 etc. Put the start and end volumes in variables V1 and V2. Then call the routine using GOSUB 9000. If you don't want any of the voices to play, put a 0 in the corresponding variable. Also remember to put the duration in variable DU, making sure it is at least 1 and also not a decimal. You can use this routine in your own programs:

```
10 V1=15:V2=15:S1=240:S2=0:S3=0:S4=0
20 DU=250:GOSUB 9000
30 GOTO 30
8999 REM SOUND ROUTINE
9000 POKE 36878,PEEK(36878) AND 240
9005 POKE 36874,S1:POKE 36875,S2:POKE
 36876,S3:POKE 36877,S4
9010 DV=(V2-V1)/DU
9015 FOR I=2 TO DU
9020 V1=V1+DV:POKE 36878,PEEK(36878)
AND 240 OR V1
9030 NEXT I:POKE 36878,PEEK(36878) AN
D 240
9040 POKE 36874,0:POKE 36875,0:POKE 3
```

```
6876,0:POKE 36877,0
9050 RETURN
```

That was only one note playing. The following program plays three notes which all get quieter.

```
10 V1=15:V2=0:S1=133:S2=195:S3=225:S4
=0
20 DU=500:GOSUB 9000
30 GOTO 30
8999 REM SOUND ROUTINE
9000 POKE 36878,PEEK(36878) AND 240
9005 POKE 36874,S1:POKE 36875,S2:POKE
 36876,S3:POKE 36877,S4
9010 DV=(V2-V1)/DU
9015 FOR I=2 TO DU
9020 V1=V1+DV:POKE 36878,PEEK(36878)
AND 240 OR V1
9030 NEXT I:POKE 36878,PEEK(36878) AN
D 240
9040 POKE 36874,0:POKE 36875,0:POKE 3
6876,0:POKE 36877,0
9050 RETURN
```

SOUND ADVICE

The control numbers held in the variables can of course be the result of calculations instead of just numbers. In the next program, you have to guess the number between 1 and 50 which the computer has thought of. The feedback on each guess — which will help you home in on the correct number in the shortest possible number of guesses — is in the form of notes from your VIC. Once you've played it for a few rounds, you'll learn to interpret the output from the computer.

When you manage to guess the answer, you'll get this on the screen:

```
YES,I WAS THINKING OF
 12


YOU GOT IT IN 6
GUESSES
```

Here is the listing and note the POKE in line 30 is to set another register, this one controls background and border colour and will be discussed later. Note also, the control code in the PRINT statement to get white writing:

```
10 REM SOUND ADVICE
20 LET R=INT(RND(1)*50+1)
30 POKE 36879,42:PRINT "■"
40 LET GUESS=1
50 PRINT "■■■■■■■■■■■■THIS IS GUESS #";
GUESS
60 PRINT "WHAT NUMBER AM I":INPUT"THI
NKING OF";A
80 IF A=R THEN 130
90 PRINT "■■■■NO,";A;"IS NOT CORRECT"
100 S1=0:S3=0:S4=0:V1=15:V2=0:DU=10*(
8-GUESS/8):S2=130+2*ABS(R-A)
110 GOSUB 9000:LET GUESS=GUESS+1
120 IF GUESS<40 THEN 50
125 PRINT "■■■■■■■■TOO SLOW,I WAS":PRI
NT"THINKING OF";R:GOTO 200
130 PRINT "■■■■YES,I WAS THINKING OF"
:PRINT R
140 PRINT "■■■■YOU GOT IT IN";GUESS:PR
INT "GUESSES"
150 END
8999 REM SOUND ROUTINE
9000 POKE 36878,PEEK(36878) AND 240
9005 POKE 36874,S1:POKE 36875,S2:POKE
```

```
 36876,S3:POKE 36877,S4
9010 DV=(V2-V1)/DU
9015 FOR I=2 TO DU
9020 V1=V1+DV:POKE 36878,PEEK(36878)
AND 240 OR V1
9030 NEXT I:POKE 36878,PEEK(36878) AN
D 240
9040 POKE 36874,0:POKE 36875,0:POKE 3
6876,0:POKE 36877,0
9050 RETURN
```

The first line is, of course, just a REM statement to tell you the name of the program. The next line generates a random number between one and fifty, and sets the numeric variable R equal to this number. In line 30, the computer turns the background to red, the colour in which the computer prints to white, and the area surrounding the centre of the screen, the BORDER, to red, the same colour as the background.

Line 40 sets a numeric variable called GUESS (the VIC only recognises variables by the first two characters of their name — so that GUESS and GUN are considered to be the same variable — but you can use longer ones to aid readability) to one, and line 50 prints up the number of your guess. Line 60 accepts your guess, as variable A. If A (the number you've guessed) is the same as R (the number the computer thought of back in line 20), as the computer checks in line 80, then the computer knows you have guessed its number.

If you are not right, then line 90 tells you so, and the following line (100) gives you your audio feedback, with both the length of the note, and its pitch, being related to the difference between your number and the computer's number. The variable GUESS is *incremented* by one (that is, one is added to the value of the variable GUESS) before the computer returns to line 50 to accept your next guess.

If you have guessed the number correctly (detected in line 8Ø) the program moves to line 13Ø. The screen is cleared and the computer tells you the number it is thinking of, and then (in line 14Ø) tells you how many guesses you took.

CHAPTER EIGHT
MAKING COMPARISONS

We all know the equals sign (=) and we've seen it in use in several programs 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.

```
1 POKE650,255:POKE36878,15:DEF FNA(A)
=PEEK(7680+Y*22+X):CO$="
5 Y$="
7 X$="
10 POKE36879,125:PRINT"
          ";
15 PRINT"                              ";
```

```
20 PRINT" ▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮ ";
30 PRINT" ▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮ ";
40 FOR A=1TO5:PRINT" ";MID$(CO$,RND(1
)*4+1,1);
50 PRINT"▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮ ";
60 NEXTA
70 FORA=1TO12
80 PRINT" ▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮ ";
90 NEXTA
100 T=10:SC=0
105 PRINT"▮LIVES";T;"▮▮ ",
107 PRINT"SCORE";SC;"▮▮ ";
110 C=-1:D=-1:Y=13:X=INT(RND(1)*7)+6:
M=X-2
120 IFFNA(1)=207 THENSC=SC+1:PRINT"▮▮
";TAB(16);SC:C=-C:FR=255:LE=2:GOSUB10
00
130 IFY>20THEN230
140 PRINTLEFT$(Y$,Y+1);LEFT$(X$,X);
145 PRINT"▮●";LEFT$(Y$,22);LEFT$(X$,M
);
150 PRINT" ▮▬▬▮ ";
160 GETA$:M=M-(A$="M"ANDM<17)+(A$="Z"
ANDM>0)
170 IFY+C<2THENC=-C:FR=191:LE=1:GOSUB
1000
180 IFY=20THENIFX>=MTHENIFX<=M+4THENC
=-C:LE=1:FR=195:GOSUB1000
190 IFX+D<10RX+D>20THEND=-D:FR=195:LE
=1:GOSUB1000
200 PRINTLEFT$(Y$,Y+1);LEFT$(X$,X);
210 PRINT"▮ ";:Y=Y+C:X=X+D
220 GOTO 120
230 T=T-1:C=-C:FORR=1TO6*T:FR=195+R:L
E=2:GOSUB1000:FR=255-R:GOSUB1000:NEXT R
```

```
240 PRINT"■■LIVES";T;"■■ ",
245 PRINT"SCORE";SC;"■■ ";LEFT$(Y$,22)
;LEFT$(X$,M);
250 PRINT"■    ";
260 IFT>0THEN110
270 FR=RND(1)*127+128:LE=2:GOSUB1000:
GOTO270
1000 POKE36876,FR:FORI=1TOLE:NEXTI:PO
KE36876,0:RETURN
```

In BRICKBAT, you use the M and Z 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 squares, it will strike them, making a sound, adding to 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 260) you still have a 'life' (that is, ball) left. The program ends in line 270 with an endless series of randomly chosen beeps. You'll need to press the RUN/STOP key (sometimes called the BREAK key) to get out of the program.

The words AND and OR are also used in comparison lines, *chaining* two or more tests together, as in lines 160 and 190. 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 16Ø, the computer checks to see IF you are pressing the M key AND if M is less than 17, and if *both* these conditions are true, moves the slide one square to the right. Line 19Ø bounces the ball off the side walls. It checks to see if the ball is about to hit the left wall (IF X + D< 1) OR is about to hit the right wall (IF X + D> 2Ø) and if *either* of these conditions is found to be true, the computer 'bounces' the ball, by changing the value of D, the variable which determines the change in the position of the ball across the screen from move to move.

Notice the extensive use of control strings to position printing on the screen. Commands such as LEFT$ will be explained later on, suffice to say that they are used for the extraction of various parts of strings (known as string slicing).

Note also line 16Ø. If a condition is true in VIC BASIC, a −1 will be returned, otherwise a Ø. Therefore if A$ = "M"ANDM < 17 is true then −1 is subtracted from M otherwise a Ø is subtracted. Note that subtracting −1 is the same as adding +1 (i.e. moving 1 to the right). If the second condition in the line is true, −1 is added to M (i.e. +1 is subtracted, moving 1 to the left).

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 we have decided to supply you with these programs at this point, and hope you'll enter them '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 continue with the learning, then move straight along to chapter ten.

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.

The little man-like figures represents the pegs, and the black square in the middle of the screen is the empty spot. 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, colourfully and musically, and you are then offered another move.

This is the listing of the Solitaire program:

```
5 POKE55,255:POKE56,27
10 REM SOLITAIRE
15 CO$="■▒▓■◣":POKE36869,255:POKE368
78,15
20 GOSUB260
30 GOSUB180
40 REM ACCEPT MOVE
45 PRINT"▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓WHICH PE
G DO YOU WANT"
50 INPUT"▓TO MOVE";A
60 POKE36876,175+RND(1)*60:IFA(A)<>64
 THEN 50
65 POKE36876,0
70 INPUT"▓TO WHERE";B
```

```
80 POKE36876,175+RND(1)*60:IFA(B)<>ET
HEN70
85 POKE36876,0
90 A((A+B)/2)=E:A(A)=E:A(B)=64
100 MO=MO+1
110 CO=0
120 FORF=11TO75
130 IFA(F)=64THENCO=CO+1
140 NEXTF
150 GOSUB180
160 IFCOUNT>1 ORA(44)<>64THEN170
165 PRINT"■■YOU DID IT IN";MO:PRINT"■
MOVES":POKE36869,240:STOP
170 PRINT"■■■■■■■■■■■■■■■■■■■■THERE ARE"
;CO
175 PRINT"■PEGS ON THE BOARD":GOTO 40
180 REM PRINT OUT
190 PRINT"■■■■■■■■■■■■■■■■■■■■1234567";
195 PRINT"■■■■■■■■-7■■■■■■■■■■■■■■■■■";
200 FORD=11TO75
210 IFD-10*(INT(D/10))=8 THEND=D+2:PR
INT"■■";-INT(D/10)+1:PRINTTAB(6);:GOT
O230
215 IFA(D)=64THENPRINT"■";:POKE36876,
200
216 IFA(D)=ETHENPOKE36876,150+RND(1)*
20
218 IFA(D)<>64THENPRINT"■";
220 PRINTMID$(CO$,1-(RND(1)*5+1)*(A(D
)=64),1);CHR$(A(D));
230 NEXTD
240 PRINT"■■■■■■■■■■  ";MO;"MOVES SO FA
R ";
250 POKE36876,0:RETURN
260 POKE36879,26:PRINT"■■";:DIMA(87)
```

```
270 E=166
280 FORD=11TO75:IFD-10*(INT(D/10))=8T
HEND=D+3
290 READA(D)
300 NEXTD
310 MO=0
320 FORZ=0TO7
330 READX
340 POKE7168+Z,X
345 NEXTZ
350 RETURN
360 DATA32,32,64,64,64,32,32
370 DATA32,32,64,64,64,32,32
380 DATA64,64,64,64,64,64,64
390 DATA64,64,64,166,64,64,64
400 DATA64,64,64,64,64,64,64
410 DATA32,32,64,64,64,32,32
420 DATA32,32,64,64,64
430 DATA28,93,73,62,28,54,99,0
```

The program works as follows. Line 20 sends action to the subroutine from line 260 which sets up the requirements for the start of the game. The BORDER is set to red, the PAPER to white, and the print colour is set to black. The screen is cleared, and the A array is dimensioned to hold the board. One variable is initialised: E for the black, central square. The loop is lines 280 to 300 reads the DATA from lines 360 to 420 to get the positions for the start of the game.

Character 64 is the first user-defined graphic ("@") and this is set up with the loop from lines 320 to 345, using the DATA from line 430.

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 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. "BEST SO FAR IS..." appears on the screen, and the computer then waits for you to take your hands off the keyboard, to prevent cheating (as if you'd try to cheat on your computer), before the whole thing begins again. As well as printing out your best score, the computer also prints out a long red bar, the length of which is related to your best score, to give you an easy-to-read indicator of how well you're doing. The game continues until you manage to get your reaction time down to below five, which is not an easy task.

Here is the listing of REACTION TEST:

```
5 REM REACTION TEST
10 HI=1000:POKE36878,15
15 PRINT"■";:A=RND(-TI):B$="
                "
20 PRINT"■■■■■■■■STAND BY
  ";
30 FORA=1TO200+RND(1)*500
35 GETA$:IFA$<>""THEN35
40 NEXTA
45 POKE36876,0
50 PRINT"■■■■■■■■OK,PRESS THE 'Z' KEY
";
60 CO=0
70 CO=CO+1
90 GETA$
100 IFA$<>"Z"THEN70
110 PRINT"■■YOUR SCORE WAS ";CO;"■■ "
```

```
120 IFCO<HITHENHI=CO:POKE36876,130-(C
O*(CO<60))*2
130 PRINT"■";B$;"■■ BEST SO FAR IS";H
I;"■■ ■■";:X=HI:IFX>44THENX=44
132 PRINTLEFT$(B$,X/2)
135 IFHI<5 THEN160
140 GETA$:IFA$<>""THEN140
150 GOTO 20
160 POKE36879,56:PRINT"■■";
170 PRINT"■■■■■   YOU'RE THE CHAMP!!"
175 POKE36876,0
```

Line 10 sets the variable HI to 1000, after the random number generator is seeded in line 15, a string (B$) is set up. Following this with a command to PRINT B$ would get the computer to print up a line of 22 spaces. The only way you could see would be to PRINT them with REVERSE, so that a solid bar would be printed. In line 130, the computer uses B$ to blank out everything on the top row, before the words BEST SO FAR IS... are printed on the following line. Then, the end of that line prints a solid red bar (using LEFT$(B$,X/2) as you can see) which prints a bar related to your best score, so long as this best score is lower than 45.

The variable CO is set to zero in line 60 and incremented by one every time line 70 is revisited, because you have not yet managed to get to the Z key. Line 100 checks to see if you have touched the Z and if not, sends the program back to 70 where CO is incremented.

Once you have managed to get to Z, the program 'falls through' to line 110 where you are told your score. This is compared with the best score (variable name HI) in the following line, and HI is adjusted to CO if CO is the lower of the two. A beep sounds, related to your score on that particular round. The lower the tone, the better you did.

The next line (135) checks to see if you have managed to get a score lower than five, and if you have, directs the computer to line 160 where the YOU'RE THE CHAMP!!! message is displayed. If not, line 140 waits until you have taken your fingers off the keyboard and line 150 directs the program to line 20, where the next round is played out. You'll have few problems getting a score of 15 or so, but to get it below five is extremely difficult.

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 a 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 are rounded, and the computer's stones are square. 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 round last a long, long time. This may be satisfactory when you're playing against another human being, and can chat as the game proceeds, but you'll find it takes far too long to complete a game when playing against the machine. Therefore, we've changed the aim of the game so that each of you is trying to capture seven of your opponent's pieces.

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.

Observing a game in action is perhaps the best way to understand it.

Next is the listing for Hasami Shogi, and after the listing we'll explain how the program works.

```
10 REM HASAMI SHOGI
20 REM MAKE SURE YOUR COMPUTER IS IN
UPPER CASE MODE
30 GOSUB790
40 GOSUB90
50 GOSUB460
60 GOSUB630
70 GOSUB460
80 GOTO40
90 REM CAPTURE
100 A=99
110 IFA(A)<>CTHEN190
120 IFA(A-10)=ETHENIFA(A-9)=HTHENIFA(
A-8)=CTHENB=A-10:GOTO350
130 IFA(A-10)=ETHENIFA(A-11)=HTHENIFA
(A-12)=CTHENB=A-10:GOTO350
```

```
140 IFA(A-10)=ETHENIFA(A+11)=HTHENIFA
(A+12)=CTHENB=A-10:GOTO350
150 B=1
160 IFA+2*C(B)<110RA+2*C(B)>99THEN180
170 IFA(A+C(B))=EANDA(A+2*C(B))=HANDA
(A+3*C(B))=C THEN A(A+2*C(B))=E:CS=CS
+1:GOTO340
180 IFB<4THENB=B+1:GOTO160
190 IFA>11THENA=A-1:GOTO110
200 REM NON-CAPTURE
210 CO=0
220 CO=CO+1
230 A=RND(1)*89+11
240 IFA(A)=CTHEN270
250 IFCO<200THEN220
260 PRINT"■HASAMI SHOGI MASTER":PRINT
"I GIVE YOU THE         VICTORY!":END
270 B=1
280 IFA+2*C(B)<11THEN300
290 IF(A(A+C(B))=CORA(A+C(B))=H)ANDA(
A+2*C(B))=ETHENB=A+2*C(B):GOTO350
300 IFA(A+C(B))=ETHEN330
310 IFB<4THENB=B+1:GOTO280
320 GOTO250
330 REM COMPUTER MOVES
340 B=A+C(B)
350 B1=B-10*(INT(B/10))
360 A(B)=C:A(A)=E
370 IFB1>7THEN390
380 IFA(B+1)=HANDA(B+2)=CTHENA(B+1)=E
:CS=CS+1
390 IFB1<3THEN410
400 IFA(B-1)=HANDA(B-2)=CTHENA(B-1)=E
:CS=CS+1
410 IFA>89THEN430
```

```
420 IFA(B+10)=HANDA(B+20)=CTHENA(B+10
)=E:CS=CS+1
430 IFA<29THENRETURN
440 IFA(B-10)=HANDA(B-20)=CTHENA(B-10
)=E:CS=CS+1
450 RETURN
460 REM BOARD PRINTOUT
470 PRINTCHR$(147);
480 PRINTTAB(3);"■1 2 3 4 5 6 7 8 9"
490 FORM=90TO10STEP-10
500 PRINT"■ "CHR$(M/10+64);" ■";
510 FORN=1TO9
520 PRINTCHR$(A(M+N));" ";
530 NEXT
540 PRINT"■"CHR$(M/10+64)
550 NEXT
560 PRINTTAB(3);"1 2 3 4 5 6 7 8 9"
570 PRINT"■■COMPUTER:"CS"HUMAN:"HS
590 IFCS>60RHS>6THEN610
600 RETURN
610 IFCS>HSTHENPRINT:PRINT"■I WIN!":E
ND
620 PRINT:PRINT"■YOU WIN!":END
630 REM PLAYER MOVE
640 INPUT"■FROM (LETTER,NO)";A$
650 IFA$="S"THENEND
660 IFLEN(A$)<>2THEN640
670 PRINT"FROM "A$" TO ";:INPUTB$
680 IFLEN(B$)<>2THEN640
690 A=10*(ASC(A$)-64)+VAL(RIGHT$(A$,1
))
700 B=10*(ASC(B$)-64)+VAL(RIGHT$(B$,1
))
710 Y=VAL(RIGHT$(B$,1))
720 A(B)=H:A(A)=E
```

```
730 IFA(B+1)=CANDA(B+2)=HANDY<=7THENA
(B+1)=E:HS=HS+1
740 IFA(B-1)=CANDA(B-2)=HANDY>=3THENA
(B-1)=E:HS=HS+1
750 IFB>79THEN770
760 IFA(B+10)=CANDA(B+20)=HTHENA(B+10
)=E:HS=HS+1
770 IFB>=31THENIFA(B-10)=CANDA(B-20)=
HTHENA(B-10)=E:HS=HS+1
780 RETURN
790 REM INITIALISE
800 REM SEED RANDOM NUMBER GENERATOR
810 PRINTCHR$(147)"PRESS A KEY"
820 GETA$:IFA$<>""THEN820
830 GETA$:IFA$=""THEN830
850 N=RND(-TI)
870 PRINTCHR$(147)
880 DIMA(129),C(4)
890 H=113:C=166:E=42
900 FORZ=11TO29
910 IFZ=20THENZ=21
920 A(Z)=H
930 NEXTZ
940 FORZ=31TO79
950 IF10*INT(Z/10)=ZTHENZ=Z+1
960 A(Z)=E
970 NEXTZ
980 FORZ=81TO99
990 IFZ=90THENZ=91
1000 A(Z)=C
1010 NEXTZ
1020 HS=0
1030 CS=0
1040 FORZ=1TO4
1050 READC(Z)
```

```
1060 NEXT
1070 DATA-10,-1,1,10
1080 GOSUB460
1090 RETURN
```

The program listing may seem very long, but it is somewhat simpler to understand than it may appear at first sight. This is because it has been written in 'modules', each of which is called by a GOSUB near the start of the program. Programming in this way is a good idea if you are developing a program which could be fairly long and involved, as it enables you to keep track of what each section is doing. As well, it is easy to track down *bugs* (the common computer word for a program error, and named after a malfunction in an early IBM computer which was found to have been caused by a moth which was caught inside the cabinet) if the program is in modules, as the module which contains the bug should be relatively easy to isolate.

The idea of programming in 'modules' in this way is perhaps easier to understand if we show how it has worked in this program.

Line 30–GOSUB 790: This sends the program to the subroutine which sets up all the starting variables. This kind of a subroutine is often called the *initialisation subroutine*. On returning from the subroutine, the computer gets the instruction to GOSUB 90.

Line 40–GOSUB 90: This is the subroutine which determines the computers move. The computer moves first in this game. Firstly, in the lines from 90 to 180, it goes through the board, square by square, looking for a potential capture. It does not take very long to do this.

If a capture has not been found, the computer then moves into its routine for looking for a non-capture move. Here, it sets the variable CO, which counts the number of moves it has selected, to zero to start with, and then adds one to it in line 220. Line 230 chooses a square on the board at random — the squares are held in an array, the A array, and are numbered from 11 to 89 — and

then checks to see if it has one of its own pieces on that square. The variable C indicates one of its own pieces. If it finds one there, it moves to line 27Ø to make the move.

If not, the program checks that it has not yet selected 2ØØ moves (that is, it checks that the variable CO is less than 2ØØ) and if it finds that it is, sends the action back to line 22Ø, where one is added to CO, and a new square is selected in line 23Ø. If it tries 2ØØ numbers at random, and does not find one of its own pieces — a most unlikely situation — it will concede the game with line 26Ø.

However, if it finds one of your pieces (in line 24Ø) the program goes to the routine from line 27Ø, where it tests the square located for a move, using elements of the C array, which represent potential moves from each square. Jumps are given priority over single moves.

If a move has been found, the program then goes to line 33Ø. If not, line 32Ø sends the program back to line 22Ø, where another piece is located to consider for moves.

The actual move, and the result of that move, is made in the lines from 33Ø. The computer makes its move in line 36Ø. Lines 37Ø through to 44Ø process the move, taking any captured pieces off the board. Line 45Ø returns the program to line 5Ø.

Line 5Ø–GOSUB 46Ø: The routine from line 46Ø prints out the board. This subroutine is, of course, visited after each move, whether by the computer or the player, to update the board. Line 59Ø checks to see if either player has managed to capture more than six of the opponent's pieces, and if so, goes to line 61Ø to print out the name of the winner. If not, the board printout subroutine is exited via line 6ØØ, and the program returns to line 6Ø.

Line 6Ø–GOSUB 63Ø: We hope by now you are working out how the *modular* idea is applied. A series of subroutines, each of which carries out a specific task, is called over and over again as the program progresses from a small number of lines, which the program loops through, time and again. Even if it's not

100% clear now, it should be by the time we finish the explanation. Anyway, this subroutine from line 630 — as the REM statement explains — allows the player to enter his or her move (terminating the program if the player enters "S" for stop) in lines 640 and 670, translates the letter and numbers of the players "to" and "from" squares into numbers which the computer can apply to the A array, and then makes the relevant changes in the loop through to line 780, where the RETURN sends the program back to line 70.

Line 70—GOSUB 460: This line sends the program back to the subroutine which prints out the board, to reflect the changes introduced by the players move.

Line 80—GOTO 40: This sends the program back to line 40, where a subroutine call is directed to 90, to get the computer to make a move.

We hope by now you can understand the structure. This is what is happening:

GOSUB 790	—	initialised (this subroutine is not revisited)
GOSUB 90	—	computer makes a move
GOSUB 460	—	board is printed out
GOSUB 630	—	human makes a move
GOSUB 460	—	board is printed out again
GOTO 40	—	return to the line to allow the computer to make a move

This sequence is cycled over and over again while the program is running until one player or the other manages to capture seven or more of the opponents pieces. You can see that if we had, for example, a problem with the way the board was being printed on the screen, it could be overcome relatively easily, because we would know the error lay between lines 460 and 630. Similarly, if we wished to alter the style of play exhibited by the computer, all we'd have to do is look at the sequence which began at line 90. Programming in this 'modular' way is very useful.

The only routine we have not looked at is that starting at line 790, the initialisation routine. Line 880 initialises two arrays: A to hold the board and C to hold the potential moves in four directions from each square. Variables H (for the human piece), C (for the computer piece) and E (for an empty square) are assigned at the end of this line. The three loops (900 to 930, 940 to 970 an 980 to 1010) 'fill up' the board with the starting positions. In lines 1020 and 1030 the variables which hold the score, HS for human scores, CS for computer score, are set to zero. Line 1080 calls the board subroutine so you can look at the board as the computer considers a move, and then 1090 sends the program back to line 40, where the whole 'game cycle' begins.

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* (such as A$, B1$, FC$ etc). 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 code. This is known as the ASCII code and is pretty standard for most micro computers (your VIC also has a lot of extra, none standard codes as well as some of its symbols having none standard codes). Telling the computer to print the *character* of that code produces the character, and is done using the function CHR$ (N) where N is the code of the character. You may remember seeing this in chapter one. It is easy to understand this. The 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 the answer of 65. Now 65 is the ASCII code of the letter "A". We can turn it back to an "A" by asking the

computer to print the character which responds to ASCII code 65. We do this with the function CHR$.

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

You can get the computer to print out every code and character with the next short program (except the control codes which would upset the display), which has — as you can see — a very long print out. When SCROLL? appears, just hit the RETURN key to continue.

```
10 REM SHOWING ASC AND CHR$
20 FOR A=32 TO 191 STEP 20
30 FOR B=A TO A+19
35 IF B=128 THEN B=132:GOTO 45
37 IF B=192 THEN B=200:GOTO 45
40 PRINT"ASC";B;TAB(10);"CHARACTER ";CHR$(B)
45 NEXTB
50 IF A=112 THEN LET A=140
60 INPUT"SCROLL";A$
70 NEXTA
```

Notice the use of two loops so that only 20 codes are printed on the screen at once, thus preventing the information scrolling away before you have a chance to read it. This is what you'll get when you run the program:

```
First in uPPer Case

ASC 32          CHARACTER
ASC 33          CHARACTER !
ASC 34          CHARACTER "
ASC 35          CHARACTER #
ASC 36          CHARACTER $
ASC 37          CHARACTER %
ASC 38          CHARACTER &
ASC 39          CHARACTER '
```

```
ASC 40          CHARACTER (
ASC 41          CHARACTER )
ASC 42          CHARACTER *
ASC 43          CHARACTER +
ASC 44          CHARACTER ,
ASC 45          CHARACTER -
ASC 46          CHARACTER .
ASC 47          CHARACTER /
ASC 48          CHARACTER 0
ASC 49          CHARACTER 1
ASC 50          CHARACTER 2
ASC 51          CHARACTER 3
SCROLL

ASC 52          CHARACTER 4
ASC 53          CHARACTER 5
ASC 54          CHARACTER 6
ASC 55          CHARACTER 7
ASC 56          CHARACTER 8
ASC 57          CHARACTER 9
ASC 58          CHARACTER :
ASC 59          CHARACTER ;
ASC 60          CHARACTER <
ASC 61          CHARACTER =
ASC 62          CHARACTER >
ASC 63          CHARACTER ?
ASC 64          CHARACTER @
ASC 65          CHARACTER A
ASC 66          CHARACTER B
ASC 67          CHARACTER C
ASC 68          CHARACTER D
ASC 69          CHARACTER E
ASC 70          CHARACTER F
ASC 71          CHARACTER G
SCROLL
```

```
ASC 72              CHARACTER H
ASC 73              CHARACTER I
ASC 74              CHARACTER J
ASC 75              CHARACTER K
ASC 76              CHARACTER L
ASC 77              CHARACTER M
ASC 78              CHARACTER N
ASC 79              CHARACTER O
ASC 80              CHARACTER P
ASC 81              CHARACTER Q
ASC 82              CHARACTER R
ASC 83              CHARACTER S
ASC 84              CHARACTER T
ASC 85              CHARACTER U
ASC 86              CHARACTER V
ASC 87              CHARACTER W
ASC 88              CHARACTER X
ASC 89              CHARACTER Y
ASC 90              CHARACTER Z
ASC 91              CHARACTER [
SCROLL

ASC 92              CHARACTER £
ASC 93              CHARACTER ]
ASC 94              CHARACTER ↑
ASC 95              CHARACTER ←
ASC 96              CHARACTER —
ASC 97              CHARACTER ♠
ASC 98              CHARACTER |
ASC 99              CHARACTER —
ASC 100              CHARACTER —
ASC 101              CHARACTER —
ASC 102              CHARACTER —
ASC 103              CHARACTER |
ASC 104              CHARACTER |
ASC 105              CHARACTER \
```

```
ASC 106          CHARACTER ╲
ASC 107          CHARACTER ╱
ASC 108          CHARACTER └
ASC 109          CHARACTER ╲
ASC 110          CHARACTER ╱
ASC 111          CHARACTER ┌
SCROLL

ASC 112          CHARACTER ┐
ASC 113          CHARACTER ●
ASC 114          CHARACTER ─
ASC 115          CHARACTER ●
ASC 116          CHARACTER │
ASC 117          CHARACTER ╱
ASC 118          CHARACTER ╳
ASC 119          CHARACTER ○
ASC 120          CHARACTER ♠
ASC 121          CHARACTER ▕
ASC 122          CHARACTER ♦
ASC 123          CHARACTER ┼
ASC 124          CHARACTER ▒
ASC 125          CHARACTER ▕
ASC 126          CHARACTER π
ASC 127          CHARACTER ◥
SCROLL

ASC 160          CHARACTER
ASC 161          CHARACTER ▮
ASC 162          CHARACTER ▬
ASC 163          CHARACTER ▔
ASC 164          CHARACTER _
ASC 165          CHARACTER ▕
ASC 166          CHARACTER ▒
ASC 167          CHARACTER ▕
ASC 168          CHARACTER ▱
ASC 169          CHARACTER ◤
```

```
ASC 170              CHARACTER ┤
ASC 171              CHARACTER ├
ASC 172              CHARACTER ▮
ASC 173              CHARACTER └
ASC 174              CHARACTER ┐
ASC 175              CHARACTER ▁
ASC 176              CHARACTER ┌
ASC 177              CHARACTER ┴
ASC 178              CHARACTER ┬
ASC 179              CHARACTER ┤
SCROLL

ASC 180              CHARACTER ▏
ASC 181              CHARACTER ▌
ASC 182              CHARACTER ▊
ASC 183              CHARACTER ▔
ASC 184              CHARACTER ▀
ASC 185              CHARACTER ▄
ASC 186              CHARACTER ▟
ASC 187              CHARACTER ▖
ASC 188              CHARACTER ▝
ASC 189              CHARACTER ▗
ASC 190              CHARACTER ▘
ASC 191              CHARACTER ▚
SCROLL

Now in lower case
asc 32               character
asc 33               character !
asc 34               character "
asc 35               character #
asc 36               character $
asc 37               character %
asc 38               character &
asc 39               character '
asc 40               character (
```

```
asc 41              character )
asc 42              character *
asc 43              character +
asc 44              character ,
asc 45              character -
asc 46              character .
asc 47              character /
asc 48              character 0
asc 49              character 1
asc 50              character 2
asc 51              character 3
scroll
asc 52              character 4
asc 53              character 5
asc 54              character 6
asc 55              character 7
asc 56              character 8
asc 57              character 9
asc 58              character :
asc 59              character ;
asc 60              character <
asc 61              character =
asc 62              character >
asc 63              character ?
asc 64              character @
asc 65              character a
asc 66              character b
asc 67              character c
asc 68              character d
asc 69              character e
asc 70              character f
asc 71              character g
scroll
asc 72              character h
asc 73              character i
```

```
asc 74              character j
asc 75              character k
asc 76              character l
asc 77              character m
asc 78              character n
asc 79              character o
asc 80              character p
asc 81              character q
asc 82              character r
asc 83              character s
asc 84              character t
asc 85              character u
asc 86              character v
asc 87              character w
asc 88              character x
asc 89              character y
asc 90              character z
asc 91              character [
scroll
asc 92              character £
asc 93              character ]
asc 94              character ↑
asc 95              character ←
asc 96              character —
asc 97              character A
asc 98              character B
asc 99              character C
asc 100              character D
asc 101              character E
asc 102              character F
asc 103              character G
asc 104              character H
asc 105              character I
asc 106              character J
asc 107              character K
```

```
asc 108          character L
asc 109          character M
asc 110          character N
asc 111          character O
scroll
asc 112          character P
asc 113          character Q
asc 114          character R
asc 115          character S
asc 116          character T
asc 117          character U
asc 118          character V
asc 119          character W
asc 120          character X
asc 121          character Y
asc 122          character Z
asc 123          character +
asc 124          character ▒
asc 125          character |
asc 126          character ▚
asc 127          character ▒
scroll
asc 160          character
asc 161          character ▌
asc 162          character ▄
asc 163          character ▔
asc 164          character _
asc 165          character ▏
asc 166          character ▓
asc 167          character |
asc 168          character ▒
asc 169          character ▒
asc 170          character ▏
asc 171          character ▖
asc 172          character ▪
```

```
asc 173              character └
asc 174              character ┐
asc 175              character ─
asc 176              character ┌
asc 177              character ┴
asc 178              character ┬
asc 179              character ┤
scroll
asc 180              character ├
asc 181              character ▌
asc 182              character ▐
asc 183              character ▔
asc 184              character ▁
asc 185              character ▒
asc 186              character ▕
asc 187              character ◤
asc 188              character ◥
asc 189              character ◣
asc 190              character ◢
asc 191              character ▉
scroll
```

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

```
10 REM SHOWING ASC AND CHR$
20 FOR A=32 TO 191 STEP 40
30 FOR B=A TO A+39 STEP 2
35 IF B=128 THEN B=152:GOTO 45
37 IF B=192 THEN B=220:GOTO 45
40 PRINT TAB(1);B;TAB(6);CHR$(B);
42 PRINT TAB(11);B+1;TAB(16);CHR$(B+1)
45 NEXTB
50 IF A=112 THEN LET A=120
60 INPUT"SCROLL";A$
70 NEXT A
```

The TAB function in line 30 moves printing along to the column in the brackets, 0 being the left most column. If the column has already been passed, the TAB is ignored.

And this is the result of running it:

```
 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
SCROLL
 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 ♠
98 | 99 —
100 — 101 —
102 — 103 |
104 | 105 ╲
106 ╲ 107 ╱
108 └ 109 ╲
110 ╱ 111 ┌
SCROLL
112 ┐ 113 ●
114 — 115 ●
116 | 117 ╱
118 X 119 O
120 ♠ 121 |
122 ♦ 123 +
124 ▒ 125 |
126 π 127 ◥
SCROLL
160 161 ▌
162 ▬ 163 —
164 _ 165 |
166 ▒ 167 |
168 ▒ 169 ◤
170 | 171 ├
172 ▪ 173 └
174 ┐ 175 ▬
176 ┌ 177 ┴
178 ┬ 179 ┤
180 | 181 ▌
182 ▌ 183 —
184 ▬ 185 ▬
186 ┘ 187 ▪
188 ▪ 189 ┘

```
  190 "                191 "
SCROLL
```

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.

Make sure that SHIFT LOCK is *not* engaged when you run this program. 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 40 (find the character of the number, chosen at random in line 20, represented by the variable A), and A$ (which is the key you pressed, using GET which is explained a little later in the book) is compared with CHR$ a in line 60 to see if you were correct.

```
5 POKE 36878,15:PRINT"□"
10 REM KEYBOARD INSTRUCTOR
15 LET BEST=0
20 LET A=65+INT(RND(1)*26)
30 LET B=0
40 PRINT"[redacted]";CHR$(A)
50 GET A$
60 IF A$=CHR$(A)THEN 120
70 LET B=B+1
80 PRINT"[redacted]";B
90 IF B<200 THEN 50
100 PRINT"[redacted]SORRY, TIME IS UP"
110 GOTO 140
120 POKE36876,128+INT(RND(1)*128)
125 FORI=1TO100:NEXT:POKE 36876,0
127 PRINT"[redacted]WELL DONE, YOU SCORED"
```

```
130 PRINT 200-B;"ON THAT ONE"
140 IF 200-B>BEST THEN BEST=200-B
150 PRINT"■■BEST SO FAR IS";BEST
160 FOR G=1 TO 16*B
170 NEXT G
180 PRINT"■"
190 GOTO 20
```

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 string (which can be a word in quotes like "HELLO" or a string variable, such as A$) is used with one of the following three statements: LEFT$, RIGHT$, MID$. For example, PRINT LEFT$(A$,3) would print the three left most characters of the string A$. If the number is greater than the length of the string, then the whole string would be printed. PRINT RIGHT$("HELLO",2) would print the two right most characters of the string ("LO"). PRINT MID$(B$,5) would print the string B$ out from the fifth character onwards. A second number can be added to the statement as follows: PRINT MID$("HELLO THERE",5,4). This will print four characters of the string, starting from the fifth along ("O TH").

This can also be used to extract a single character from a string. Can you see how you would do this? The MID$ statement is used with the first number being the position of the character you want in the string, and the second number being 1 to show that you only want one character. Thus PRINT MID$(A$,3,1) will print the third character of A$. As well as printing the strings, you can of course assign them to other strings. Such as LET A$ = LEFT$(B$,3,2).

The next program should help make this clear. The string variable A$ is set equal to "FIFTH*AVE" 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 LET A$="FIFTH*AVE"
15 PRINT "A$ = ";A$
20 PRINT "MID$(A$,2) = ";MID$(A$,2)
30 PRINT "RIGHT$(A$,4) = ";RIGHT$(A$,4)
40 PRINT "MID$(A$,6,1) = ";MID$(A$,6,1)
50 PRINT "MID$(A$,4,4) = ";MID$(A$,4,4)
60 PRINT "LEFT$(A$,7) = ";LEFT$(A$,7)
```

You'll see this when you run the program:

```
A$ = FIFTH*AVE
MID$(A$,2) = IFTH*AVE
RIGHT$(A$,4) = *AVE
MID$(A$,6,1) = *
MID$(A$,4,4) = TH*A
LEFT$(A$,7) = FIFTH*A
```

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 LET A$="AMERICA"
20 LET B$="COLUMBUS"
30 LET C$=A$+B$
40 PRINT "A$= ";A$
50 PRINT "B$= ";B$
60 PRINT "C$= ";C$
70 LET D=INT(RND(1)*6+1)
80 LET E=INT(RND(1)*6+1)
90 LET D$=MID$(C$,D,E)
```

```
100 PRINT "D$= ";D$
110 LET E$=A$+D$
120 PRINT "E$= ";E$
```

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

```
A$= AMERICA
B$= COLUMBUS
C$= AMERICACOLUMBUS
D$= MERI
E$= AMERICAMERI

A$= AMERICA
B$= COLUMBUS
C$= AMERICACOLUMBUS
D$= ICACOL
E$= AMERICAICACOL

A$= AMERICA
B$= COLUMBUS
C$= AMERICACOLUMBUS
D$= RICA
E$= AMERICARICA
```

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 a very interesting display. Once you've seen the program running, you'll understand why the program has been given the name it has. Notice the string variable CO$ which controls all the colour control codes. This is a useful trick because it allows you to print in colour number N, simply by typing PRINT MID$(CO$,N,1). The G variable is used in line 60 both to choose the colour using this method and also to choose the element out of the name string:

```
1 REM NAME PYRAMID
2 REM SHOWING STRING SLICING
3 CO$="[illegible]":POKE 36878,15
5 POKE 36879,8:PRINT"▛▜";
10 PRINT"WHAT IS YOUR":INPUT"NAME";A$
20 IF LEN(A$)>11THEN A$=LEFT$(A$,11)
30 LET A=LEN(A$)
35 PRINT TAB(10);"▚▪":REM CBM SHIFT 'D'
40 FOR G=1 TO A
45 PRINT TAB(11-G);
50 FOR H=1 TO 2*G
60 PRINTMID$(CO$,G/1.7+1,1);MID$(A$,G,1);
70 NEXT H
80 PRINT
90 POKE 36876,128+4*G
100 NEXT G
110 POKE 36876,0
```

```
          ▪
         MM
        AAAA
       RRRRRR
      KKKKKKKK

    JJJJJJJJJJJJ
   AAAAAAAAAAAAAA
  MMMMMMMMMMMMMMMM
 EEEEEEEEEEEEEEEEEE
SSSSSSSSSSSSSSSSSSSS
```

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

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:

```
5 PRINT "■":POKE36878,15
10 REM ECHO GULCH - SHOWING STRING S
LICING
15 LET S=0
20 LET A$=MID$("ABCSDKDMBUDBCETYWRSGQ
DEJKDGSHBBCDADBCER",INT(RND(1)*6+1))
30 PRINT "■■■■■■■■■■■■■■■■■■■■■■■■■■■";LEFT
$(A$,1)
```

```
35 FORG=1TO14*LEN(A$):NEXT:PRINT"▯"
40 GET B$
50 IFB$<"A" OR B$>"Z" THEN 40
55 PRINT "";B$
60 IF B$=LEFT$(A$,1)THENFORG=1TO10+RN
D(1)*30:POKE36876,128+G:POKE36876,255
-G:NEXTG
62 POKE36876,0
65 PRINT "":FO
R G=1 TO 200:NEXT G
70 IF B$<>LEFT$(A$,1) THEN 110
80 LET A$=MID$(A$,2)
85 IF LEN(A$)=1 THEN 140
90 LET S=S+1
100 GOTO 30
110 PRINT"SORRY, THAT'S WRONG"
120 PRINT"YOU SCORED";S
130 STOP
140 PRINT"YOU'RE THE CHAMP!!"
```

The variable S, which holds your score, is set to zero in line 15, and line 20 sets the string variable A$ to a long line of letters. If you look at those letters in line 20, you'll see a complicated looking statement beginning with MID$ and including RND. This chooses a random starting place for the string (one of the first six characters) so you will get different letters each time you run it (although, of course, they will be the same from the starting point, but the random starting point gives the effect of a new list of characters each run). Line 30 prints LEFT$(A$,1), the first letter only of the string, on the screen, and line 35 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 8Ø 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 shorter.

Line 4Ø uses GET to *read* the keyboard. GET, as you've probably worked out by this point in the book, does not demand that you press RETURN after touching a key. GET does not, in contrast to INPUT, *wait* until you have pressed a key before the program continues. If you are not touching the keyboard when the program reaches a GET it simply passes right through the line, reading your non-touching of the keyboard as the *null string*, (two quote marks with nothing, not even a space, between them, as "").

Line 5Ø looks at B$, the variable which is set equal to whichever key is being pressed as the program goes through line 4Ø. As you can see in line 5Ø, you can use the greater than (>) and less than (<) symbols, which we discussed in chapter eight, in connection with strings. These look at all elements of a string, and compare them in terms of alphabetical order (so "ZEBRA" is *less than* "AARDVARK", and "BEAST" is *greater than* "BEAUTY"). You can compare strings using equals (=) and not equals (<>) as is shown in the next few lines of the program. Line 6Ø compares the key you've pressed (B$) with the first element of A$, and if they are the same, continues through the multi-statement line to play the sounds produced by the G loop. Line 65 then prints a green square on the spot where the next letter will appear. Line 7Ø compares B$ with LEFT$(A$,1), and if finds they are *not equal* sends the program to lines 11Ø and 12Ø where you are told "SORRY, THAT'S WRONG" and your score is given. Line 8Ø strips the string A$ of its first character, by setting A$ equal to MID$(A$,2). Line 85 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 14Ø to print out the "YOU'RE THE CHAMP!!!" message. If

not, line 90 adds one to variable S, your score, then cycles back to line 30 to print the next letter for you.

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 RESTORE
20 DIM A(5)
30 FOR B=1 TO 5
40 READ A(B)
50 PRINT A(B)
60 NEXT B
70 DATA 88,8392,23,32,444
```

Using line 40, the program READs through the DATA statement in line 70 in order, printing up each item of DATA (with line 50).

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 55, so it reads as follows:

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

```
30 FOR B=1 TO 5
40 READ A(B)
50 PRINT A(B)
55 IF B=3 THEN RESTORE
60 NEXT B
70 DATA 88,8392,23,32,444
```

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:

```
1 DATA 45
10 REM READ,DATA RESTORE
20 DIM A(5)
22 DATA 88
30 FOR B=1 TO 5
40 READ A(B)
50 PRINT A(B)
55 DATA 432
60 NEXT B
70 DATA 933,254
```

READ and DATA work just as well with string information:

```
10 REM READ/DATA WITH STRINGS
20 FOR B=1 TO 5
30 READ A$
40 PRINT A$
50 NEXT B
60 DATA THIS,SHOWS,DATA,IN,ACTION
```

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 MIXED DATA
20 FOR B=1 TO 5
30 READ A$:READ A
40 PRINT A$,A
50 NEXT B
60 DATA THIS,342,SHOWS,34,DATA,3333
70 DATA IN,23,ACTION,742
```

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 9000 and 9010. The relative value of these strange objects is stored in the two 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 hard-nosed 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 are 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 0, then RETURN).

Line 100 sets the variable PA (which will control the background) to zero (black), and in line 110, variable IN (which will control print colour) is set to 8 (yellow). The subroutine from line 7000 (to which you are sent by line 120) sets the BORDER and background to PA and the ink to IN, followed by clearing the screen. On returning from this subroutine, a function is defined in line 130. This will be called later on in the program to generate random integers. Line 150 prints the program's name near the bottom of the screen, and line 900 seeds the random number generator.

Arrays are dimensioned and variables initialised in the lines 910 to 1160. Line 1015 makes sure the VIC is in capitals mode, make sure the shift lock key is off.

The C$ array is filled with your initial supply of goods (the coal dust and the like) with the loop from 1040 to 1080, and a diagonal row of stars is printed (line 1075) to keep you amused while this is going on. Line 1150 ensures that around 15 per cent of the 'O' array will contain numbers generated at random between one and 10000 (using the function S which was, you'll recall, defined in line 130). More stars are printed by line 1155.

Once the loop has been traversed, 1165 sends the program to the subroutine starting at 8500 which makes a few noises, and flashes the border at you, ending up (line 8560) with a black border again. Line 1170 sets the variable W (the year) to 2100, and 1180 sets up your starting money (variable M) to a value somewhere between 10000 and 20000 (using function S again). Line 1190 sends the computer to the subroutine from line 5000, which works out the value of your cargo, adding this value (J) to your money (M), on returning from the subroutine, sets J1 equal to the value of your cargo (J). Line 2000 assigns values to the background, border and print colours, then sends action to the subroutine from line 7000 to put them in order.

In lines 2040 and 2050 the year (W) and your money (M) is printed out. Another blank line is printed (by line 2060) then

the loop from 2070 to 2090 prints out your holdings. The string variable B$ is set equal to "BUY" in line 2100, and then in the subroutine from 3000 (where the program is directed by line 2110) the entire buying and selling occurs. Before 3000 is called again, by line 2140, B$ is set equal to "SELL". The word assigned to B$ determines the word which is assigned to S$ (either "BUY" in line 3030 or "SELL" in the following line). Both B$ and S$ are used in the print lines from 3150, where you are told that "TRADER 1 WISHES TO BUY..." or "TRADER 1 WISHES TO SELL..."

The year (W) is incremented in line 2150, and the next line checks to see if 20 years have elapsed since the game began. If they have (and W is not less than 2120, as checked by 2160) the program 'falls through' to the next line, where the final report is generated: "YOU HAVE ARRIVED IN PORT..." and so on, until you are told how much profit you made (line 2300). The 'best profit so far' is calculated (the best profit is assigned to variable I). You are then asked the question "DO YOU WANT TO PLAY AGAIN?" (asked in line 2400) can be either a "Y" or an "N". If you answer "Y", the program goes to line 1000, which ensures that I, the 'best profit so far' variable, is not reset in line 910. The function do not need to be redefined either, so they are missed out (the DEF FN line is before 1000).

```
10 POKE36878,15
20 Y$="■■■■■■■■■■■■■■■■■■■■■■"
25 X$="■■■■■■■■■■■■■■■■■■■■■■"
30 CO$="■■■■■"
100 LETPA=0
110 LETIN=8
120 GOSUB7000
130 DEF FNS(X)=INT(RND(1)*X)+1
150 PRINTLEFT$(Y$,22)"   GALACTIC  GR
OCER"
900 I=RND(-TI)
910 I=0
```

```
1000 DIMC$(21)
1005 DIMO%(21)
1015 PRINTCHR$(142);
1020 DIMM%(21)
1025 RESTORE
1030 DIMP(21)
1040 FORA=1TO21
1045 POKE36876,150+A
1050 READA$
1070 C$(A)=A$
1075 PRINTLEFT$(Y$,A);LEFT$(X$,A);"*"
1080 NEXTA:POKE36876,0
1090 FORA=1TO21
1100 IFA<6 THENREADM%(A):NEXT
1110 M%(A)=1
1120 NEXTA
1130 FORA=1TO21
1135 POKE36876,200-A
1140 READP(A)
1150 IFRND(1)>.85THENO%(A)=FNS(10000)
1155 PRINTLEFT$(Y$,A);LEFT$(X$,22-A);
"*"
1160 NEXTA:POKE36876,0
1165 GOSUB8500
1170 W=2100
1180 M=10000+FNS(10000)
1190 GOSUB5000
1195 J1=J
2000 PA=0:IN=8:GOSUB7000
2010 PRINT"■■■GALACTIC  GROCER■■■
"
2040 PRINT"YEAR";TAB(8);W
2050 PRINT"MONEY";TAB(7);"£";M
2060 PRINT
2070 FORA=1TO21
```

```
2080 IFO%(A)<>0THENPRINTO%(A);TAB(8);
C$(A)
2090 NEXTA
2100 B$="BUY"
2110 GOSUB3000
2130 B$="SELL"
2140 GOSUB3000
2150 W=W+1
2160 IF W<2120 THEN2000
2170 GOSUB5000
2180 J2=J
2200 PA=6:IN=2:GOSUB7000
2210 PRINT"YOU HAVE ARRIVED IN   PORT
."
2225 Z=J1:GOSUB4500
2230 PRINT"YOU STARTED OUT WITH"
2235 PRINT"£";Z$:PRINT"WORTH OF GOODS
"
2245 Z=J2:GOSUB4500
2260 PRINT"■YOU NOW HAVE £";Z$
2270 PRINT"WORTH OF GOODS■"
2290 V=INT((J2-J1)/J1*100)
2300 PRINT"YOU MADE";V:PRINT"% PROFIT
"
2320 IF I<V THEN I=V
2340 PRINT"■BEST SO FAR IS";I;"%"
2400 PRINTLEFT$(Y$,21);"DO YOU WANT T
O PLAY   AGAIN?"
2405 GETQ$:IFQ$<>"Y"ANDQ$<>"N"THEN240
5
2410 IFQ$="Y"THEN1000
2420 STOP
3000 PRINT
3010 N=FNS(5)
3030 S$="BUY"
```

```
3040 IF B$="BUY" THENS$="SELL"
3045 PRINT"THERE ARE";N;"TRADERS"
3050 PRINT"WISHING TO ";S$
3055 PRINT"SOME GOODS▯▯";LEFT$(Y$,22)
;LEFT$(X$,8)"PRESS ANY KEY"
3060 GETA$:IFA$<>""THEN3060
3062 GETA$:IFA$=""THEN3062
3065 GOSUB8000
3070 FORA=1TON
3075 F=FNS(3)
3080 X=FNS(21)
3085 IFF=1THENGOSUB4000:IFX=0THEN3270
3090 Y=FNS(10000)
3095 P=FNS(Y)
3100 Z=FNS(100)
3140 Z=INT(Z*P(X))/100
3145 GOSUB4500
3150 PRINT"TRADER";A;"WISHES TO"
3160 PRINTS$;P;"TO";Y
3170 PRINTC$(X);" AT £";Z$
3180 PRINT"EACH"
3190 PRINT"(YOU HAVE £";M;")"
3195 PRINT"(YOU HAVE";O%(X);C$(X);")"
3200 GOSUB7500
3205 PRINT"HOW MANY TO ";B$:U$="":INP
UTU$
3210 IFU$=""THEN3270
3215 U=VAL(U$)
3220 IFU<PORU>YTHEN3200
3225 IFB$="SELL"THENU=-U
3230 IFU<-O%(X)THEN3200
3240 IFM-U*Z<=0THEN3190
3250 M=M-U*Z
3260 O%(X)=O%(X)+U
3270 NEXTA
```

```
3275 IFN=0THENFORH=1TO300:NEXTH
3280 RETURN
4000 PRINT"TRADER";A;"SAYS:-"
4010 PRINT"WHAT DO YOU WANT TO":PRINT
B$
4020 Z$="":INPUTZ$
4025 IFZ$=""THENX=0:RETURN
4030 FORB=1TO21
4040 IFZ$=MID$(C$(B),M%(B),LEN(Z$))TH
ENX=B:GOTO4100
4050 NEXTB
4060 PRINT
4080 GOTO4020
4100 IFRND(1)<.8THENX=B:RETURN
4110 PRINT"SORRY, TRADER";A;"DOES"
4115 PRINT"NOT WANT ANY"
4120 PRINTMID$(C$(B),M%(B)):X=0
4130 RETURN
4500 Z$=STR$(Z)
4510 FORR=1TOLEN(Z$)
4520 IFMID$(Z$,R,1)="."ANDR=LEN(Z$)TH
ENRETURN
4530 IFMID$(Z$,R,1)="."ANDR=LEN(Z$)-1
THENZ$=Z$+"0"
4540 NEXTR
4550 RETURN
5000 J=0
5010 FORA=1TO21
5020 J=J+O%(A)*P(A)*2
5030 NEXTA
5040 J=J+M
5050 RETURN
7000 POKE36879,PA*16+PA+8
7010 PRINTMID$(CO$,IN,1);"■";
7040 RETURN
```

```
7500 NO=FNS(20)-FNS(20)
7510 POKE36876,200+NO
7520 POKE36876,209+NO
7530 POKE36876,0:RETURN
8000 IFS$="SELL"THENPA=3:IN=1:GOSUB70
00:RETURN
8010 PA=2:IN=2:GOSUB7000
8020 RETURN
8500 POKE36879,(PEEK(36879)AND248)OR2
8510 POKE36876,130
8520 POKE36879,(PEEK(36879)AND248)OR7
8530 POKE36876,150
8540 POKE36879,(PEEK(36879)AND248)OR5
8550 POKE36876,140
8560 POKE36879,(PEEK(36879)AND248)
8570 POKE36876,0:RETURN
9000 DATATONS COAL DUST,CMT ORANGE PE
EL,KG GRAIN,GALLONS SILICON INK
9005 DATABARS REJECT SOAP,SMARTIES,EX
-DICTATORS,TEST-TUBES,TV SETS,FISH
9010 DATAMISSILES,ASPIDISTRAS,MATCH H
EADS,SLIPPERS,CHIPPED MUGS,ZINC HUB-C
APS
9015 DATATOE NAILS,PEN TOPS,PAPER BAG
S,BADGES,NOISES
9020 DATA6,5,4,9,6
9030 DATA10,10,3,30,0.3,0.1,15,1,250,
3,1000,4,.1,5,1.5,10,1,0.2,0.1,3,23
```

CHAPTER TWELVE
WONDERS OF THE VIC CHIP

Your computer has a chip inside it known as the VIDEO INTERFACE CHIP. The VIC takes its name from this chip, which handles the screen display, the sound, and joysticks etc. Remember in chapter seven we introduced the idea of registers when discussing the sound, we shall now introduce more registers. but first, it might be an idea to introduce something known as binary arithmetic.

If you feel this next bit is going to give you a headache, just move to the later stages of the chapter and come back to the theory when you feel more confident.

Computers, and your VIC is no exception, think in a number system known as binary. In binary, each digit can be either a 1 or a Ø. Each digit is referred to as a bit, and it is common to use 8 or 16 bits. Normally, the right most digit is the units, the next is the tens, the next the hundreds etc. Notice how this is going up in powers of ten. In binary the right most digit is one, the next is two, the next is four, the next is eight etc. Notice how this is going up in powers of two. This is why normal numbers are said to be in base ten (denary), and binary numbers are said to be in base two.

Imagine an eight bit number, where each bit is a column. We will place a 1 in a column if we want to include the corresponding power of two, otherwise we will use a Ø. For an eight bit number,the powers of two range from Ø to 7, and are:

1, 2, 4, 8, 16, 32, 64, 128. As you can see if a 1 was placed in each column and all the powers of two added, the highest number stored in an eight bit system would be 255.

The first ten numbers in binary are:

NUMBER	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1
2	Ø	Ø	Ø	Ø	Ø	Ø	1	Ø
3	Ø	Ø	Ø	Ø	Ø	Ø	1	1
4	Ø	Ø	Ø	Ø	Ø	1	Ø	Ø
5	Ø	Ø	Ø	Ø	Ø	1	Ø	1
6	Ø	Ø	Ø	Ø	Ø	1	1	Ø
7	Ø	Ø	Ø	Ø	Ø	1	1	1
8	Ø	Ø	Ø	Ø	1	Ø	Ø	Ø
9	Ø	Ø	Ø	Ø	1	Ø	Ø	1
.								
.								
254	1	1	1	1	1	1	1	Ø
255	1	1	1	1	1	1	1	1

Do you understand it now? No, then read it again, but slowly! Your VIC has a large number of memory locations numbered between Ø and 65535. Some of these you can store numbers in (known as RAM), others you cannot alter because the computers instructions are stored there (known as ROM), and some are just empty spaces which may be filled when you plug cartridges into the back of your VIC. The ones we are interested in are in the RAM (random access memory, because you can access what you like). All of these memory locations are known as bytes and are composed of eight bits (that's what all that was about). Sometimes the VIC combines two bytes so it can handle sixteen bit numbers (up to 65535 — the number of memory locations), but that's not important right now. You can look at any of these memory locations with the function PEEK(N). This returns the memory stored in memory location (called an address from now on) whose number is N. This has a command

related to it : POKE N,P. This stores the number P (which must be between Ø and 255) in address number N.POKE will only affect RAM, whereas PEEK will look at any address. Some areas of RAM the computer uses for itself, others are used for storing your program and variables in coded form, and others are used for storing the screen and colour displays. For this reason you should be careful where you POKE (!). If you upset the computer too much, you may have to switch off and on again. You can never damage your computer with POKE, but you can easily destroy any program you have in. Don't let this worry you though, experiment around and see what wonderful effects you can achieve with this command.

And now, back to binary. You've already met the statements AND and OR which were used to combine conditions in an IF... THEN statement, they also have a different use in binary.

In binary, if we AND two numbers together, the corresponding bits of each number are compared and if both the first AND the second are one, then the result is one — otherwise the result is zero for that bit.

Example 98 AND 174

	Ø1100010	98
AND	1Ø1Ø1110	174
GIVES	ØØ1ØØØ1Ø	34

If we OR two numbers together, the corresponding bits of each number are compared and if either the first OR the second is one, then the result is one — otherwise it is zero.

Example 98 OR 174

	Ø1100010	98
OR	1Ø1Ø1110	174
GIVES	11Ø11110	238

There is another one, although not used as often as the others we may as well mention it here. It is called NOT.

If we NOT a number, all its bits are reversed, i.e. all ones turn to zeros and all zeros turn to ones.

Example NOT 98

NOT	01100010	98
GIVES	10011101	157

The VIC chip (we hadn't forgotten about it) has a number of control registers which you can alter. These are rather like RAM locations, except that storing things in them creates special effects.

THE VIC REGISTERS

There are sixteen registers in all, but some have more than one function by splitting into two parts. The address of each register will be given, together with a description of its functions.

REGISTER	*ADDRESS*	*DESCRIPTION*
0	36864	The high bit of this register determines whether the VIC will be in interlace mode or not. When in interlace mode, the VIC picture can be mixed with a picture from another source provided you are using the right equipment. To turn it on: POKE 36864, PEEK(36864) AND 127. This leaves bits 0–6 alone but replaces bit 7 with a zero. To turn it on: POKE 36864, PEEK(36864) OR 128. This leaves bits 0–6 alone and replaces bit 7 with a one. Bits 0–6 determine the horizontal position of the main picture on the TV screen. The normal value is 12. The bigger the value, the further to the right the image appears. Remember that the value can be between 0 and 127 with 6 bits. If N is the position of the screen type:

POKE 36864,(PEEK(36864)AND 128) OR N.
This resets bits Ø–6 to zero but leaves bit 7 alone. Bits Ø–6 are then replaced with N.

1 36865 All eight bits are used for this one (Ø–7), which determines the vertical position of the picture on the screen. The value is normally 38. The bigger the number, the further down the image appears. You can move the image by two dots (often called pixels) at a time. Therefore a value of 24 would move the display up by two dots. If N (between Ø and 255) is the position of the screen, type: POKE 36865, N.

2 36866 The high bit of this register is used with register 5, and will be described there.

Bits Ø–6 determine the number of columns on the screen (normally 22). Numbers over 27 will give a 27 column screen. The cursor controls will not work normally if you alter this, as they are based around a screen of 22 columns. If N is the number of columns (between Ø and 127), type: POKE 36866, (PEEK(36866) AND 128) OR N.

3 36867 Bit 7 is used with register 4, and will be described there.

Bits 1–6 determine the number of rows on the screen (normally 23). If this is set greater than 23, strange things will appear at the bottom of the screen (this is known as garbage — a rather general term used to mean anything you don't want). If N is the number of rows, type:

POKE 36867,(PEEK(36867) AND 129) OR (N*2). This leaves bits 7 and Ø alone, and replaces bits 1–6 with N. Notice how N has to be doubled in order to change it from bits Ø–5 to bits 1–6.

Bit Ø is used to determine the size of the characters on the screen. Normally they are made of 8 by 8 pixels, but when this bit is set to a 1, each character is composed of 8 dots across by 16 down. The use of this will be discussed in chapter fourteen. To set normal height, type: POKE 36867, PEEK(36867) AND 254. To set double height, type: POKE 36867, PEEK(36867) OR 1.

4 36868 This register forms bits Ø–7 of a 9 bit number, the high bit out of register 3 is used for bit 8. The resulting number is known as the raster value, and is used to synchronize the light pen with the TV picture.

5 36869 This is a hard one. Bits 4–7 determine which part of memory the VIC stores the screen image. Bit 7 MUST be a 1. Bit 7 of register 2 is bit 9 of the screen address, and bits 4–6 of register 5 form bits 10–12 of the address of the screen. To work out where the screen starts in memory, use the formula:

LET S = 4*(PEEK(36866) AND 128) + 64*(PEEK(36869) AND 112).

Bit 7 of register 2 also has another purpose. If it is a Ø, the colour starts at address 37888, otherwise it starts at

address 38400 (this is what it normally is for the unexpanded VIC). A formula for calculating the start of colour memory is:

LET C = 37888 + 4*(PEEK(36866) AND 128).

Bits 0–3 determine where the computer gets its information about the shapes of the characters. Normally this points to an area known as the character generator ROM, where both the VICs character sets are stored. To design your own characters you must make this point into some spare RAM where you have stored your characters. This will be discussed in chapter fourteen.

6	36870	This contains the horizontal co-ordinate of the light pen, from the left of the screen.
7	36871	This contains the vertical co-ordinate of the light pen from the top of the screen.
8	36872	This contains a value between 0 and 255 for game paddle X.
9	36873	Same as above, but for paddle Y.
10	36874	The high bit (bit 7) switches the bass voice on (voice 1). This is why the four voice registers must be POKEd with a value greater than 127 to produce a note, the high bit of each register must be a 1 to enable the voice.

		Bits Ø–6 are proportional to the frequency of the note, i.e. the higher the number, the higher the note.
11	36875	The high bit switches the alto voice on (voice 2). Bits Ø–6 are again proportional to the frequency, but one octave higher than voice 1.
12	36876	The high bit switches the soprano voice on (voice 3). Bits Ø–6 are proportional to the frequency, but one octave higher than voice 2 (i.e. two octaves higher than voice 1).
13	36877	The high bit switches the noise on (voice 4). Bits Ø–6 are proportional to frequency.
14	36878	Bits 4–7 determine the auxiliary colour, which is used with multi colour graphics. It can have a value between Ø and 15 (that's sixteen colours). If N is the value then type: POKE 36878,PEEK(36878) AND 15 OR (16*N). Bits Ø–3 correspond to the volume of the sounds. Ø is silence, and 15 is loudest. If N is the volume, type: POKE(36878) AND 24Ø OR N.
15	35879	Bits 4–7 determine the background colour of the screen (one of sixteen

colours numbered Ø–15). If N is the value, type:
POKE 36879, PEEK(36879) AND 24Ø OR N.

Bit 3, when set to a 1 causes normal display, and when set to a Ø causes reverse mode. In this mode, all characters on the screen will appear reversed, they will still have the same codes as in normal mode, because it is entirely independent of the reverse mode caused by a control code. To turn on reverse mode, type: POKE 36879, PEEK(36879) AND 247, to turn it off, type: POKE 36879, PEEK(36879) OR 8.

Bits Ø–2 set the colour of the screen border (one of eight colours numbered Ø–7). If N is the value, type: POKE 36879, PEEK(36879) AND 248 OR N.

That concludes the overview of the VIC chip, now for what you've been waiting for:

LIVING MORE COLOURFULLY or WHERE YOU SHOULD COME TO IF YOU MISSED OUT THE HARD STUFF

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 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. You'll see the names of the colours above keys 1 to 8. There is another way to use these colours without having to use control codes. This involves the POKE command once again. Imagine that whenever you use the POKE command for colour, the colours are numbered Ø to

7 (sometimes 0–15) instead of 1 to 8 (i.e. subtract 1 from the number on the colour key). The first method is to type POKE 646, N where N is the number of the colour.

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

```
10 REM FAST FOOD WITH COLOUR
15 POKE36879,8:PRINT"□";
20 PRINT"ENTER RANDOMIZE":INPUT"SEED"
;A:IFA>0THENA=-A
25 A=RND(A)
30 A=INT(RND(1)*4+1)
35 B=INT(RND(1)*5+2)
37 PRINT"A IS";A:PRINT"B IS";B
40 POKE 646,B:PRINT"YOU'VE ORDERED"
50 IFA=1 THEN POKE646,B/2:PRINT"A HAM
BURGER WITH THE  LOT"
60 IFA=2 THEN POKE646,B+1:PRINT"A LAR
GE FRENCH FRIES"
70 IFA=3 THEN POKE646,B+1:PRINT"A SER
VE OF RIBS"
80 IFA=4 THEN POKE646,B:PRINT"TWO HOT
 DOGS WITH     KETCHUP"
90 FORI=1TO400:NEXT
100 GOTO 20
```

The background colour can be any one of sixteen, and is stored in VIC register 15. The sixteen colours are as follows:

0 BLACK	8 ORANGE
1 WHITE	9 LIGHT ORANGE
2 RED	10 PINK
3 CYAN	11 LIGHT CYAN
4 PURPLE	12 LIGHT PURPLE
5 GREEN	13 LIGHT GREEN
6 BLUE	14 LIGHT BLUE
7 YELLOW	15 LIGHT YELLOW

Incidentally, the first eight colours are the ones normally used for the colour of the text your VIC prints in. You can see all this in action with the following program:

```
10 REM TEXT AND BACKGROUND
20 INPUT"BACKGROUND COLOUR";B
30 INPUT"TEXT COLOUR";T
40 POKE646,T:PRINT"■■■■■";
45 POKE36879,PEEK(36879)AND15 OR B*16
50 PRINT"I AM PRINTING IN COLOUR";T
60 PRINT"■■■ON BACKGROUND";B
70 FORA=1TO900:NEXTA
80 RUN
```

To set the background to colour N (between Ø and 15), type:

POKE 36879, PEEK(36879) AND 15 OR N*16.

The border colour can only be one of the first eight, and is held in the same register as the background colour. To change to border colour N (between Ø and 7), type:

POKE 36879, PEEK(36879) AND 248 OR N.

This variation on the previous program demonstrates:

```
10 REM TEXT, BACKGROUND AND BORDER
20 INPUT"BACKGROUND COLOUR";B
30 INPUT"TEXT COLOUR";T
35 INPUT"BORDER COLOUR";BC
40 POKE646,T:PRINT"■■■■■";
45 POKE36879,PEEK(36879)AND15 OR B*16
47 POKE36879,PEEK(36879)AND248 OR BC
50 PRINT"I AM PRINTING IN COLOUR";T
60 PRINT"■■■ON BACKGROUND";B
65 PRINT"■■WITH BORDER";BC
70 FORA=1TO900:NEXTA
80 RUN
```

Enter and run the following demonstration to see the colour range of your computer:

```
10 REM RANDOM PAINTING
15 POKE36878,15
20 LET A=INT(RND(1)*16)
30 LET B=INT(RND(1)*8)
40 IF A=B THEN 30
50 LET C=INT(RND(1)*8)
60 POKE646,B
65 POKE36879,PEEK(36879)AND15 OR A*16
67 POKE36879,PEEK(36879)AND248 OR C
70 FORD=32 TO 52
80 PRINTTAB(RND(1)*22);CHR$(D)
90 POKE36876,130+D:NEXTD
100 POKE36876,0:FORE=1TO300:NEXTE
110 GOTO 20
```

Remember how, not so long back, we mentioned there was another way to change the colour of the text and graphics on the screen. Your VIC stores in its memory, the position of everything on the screen. Because there are normally 23 lines of 22 columns each, there are 506 positions on the screen (23*22 = 506). Each of these positions has a corresponding 'byte' or location in memory to decide what character is stored there. Each location can hold a number between 0 and 255, each number is a code for a character (be careful — this is not the same as the ASCII code mentioned earlier). This code is listed in your users guide. On the unexpanded VIC, the top left hand location is numbered 7680 and as you move across the right of the screen, the locations increase by one. When you get to the far right, you go back to the left but a line further down. Because there are 22 columns, to stay in the same column but move up or down one line, you must subtract or add 22. To move left or right, you must subtract or add 1. If X is the horizontal coordinate from the left (0–21) and Y is vertical coordinate from the top of the screen (0–22), the position to POKE can be found by:

P = 7680 + Y*22 + X

There is an area of memory very similar to the one above, which refers to the colour on the screen. On the unexpanded VIC, the top left hand location is number 38400. Using the above coordinates, the position to POKE can be found by:

P = 38400 + Y*22 + X

This is demonstrated by the program below:

```
10 REM POKING THE SCREEN
20 PRINT"□";
30 REM RANDOM COORDINATES
35 REM Y (0-22)
40 LET Y=INT(RND(1)*23)
45 REM X (0-21)
50 LET X=INT(RND(1)*22)
55 REM COLOUR (RED)
60 POKE 38400+Y*22+X,2
65 REM CHARACTER (BALL-CODE 81)
70 POKE 7680+Y*22+X,81
80 GOTO 40
```

Now to explain the use of things like colour more fully, we are going to take a simple program, and gradually elaborate it, showing how adding such things as border 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 "M"

key, and you move yourself left using the "Z", and right using the "C" 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 30) the program starts with a limit of 15 shots. In the last, most complex, version you will have 50 shots. The number of shots is deliberately kept low in the first version so you will not be able to get a high score just by leaving your finger on the "M" 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 press RUN then RETURN, to get it underway.

```
10 REM DUCK SHOOT
11 Y$="[illegible]"
12 X$="[illegible]"
20 LET SCO=0:PRINT"[illegible]";:POKE650,255
30 LET SHOTS=15
40 LET A$="ZAB  DK SL DF GFD FGG "
50 LET ACROSS=10
60 LET DOWN=15
70 PRINTLEFT$(Y$,8);A$
80 PRINTLEFT$(Y$,DOWN);LEFT$(X$,ACROS
S-1);" X ":GETK$
90 IFK$<>"M"THEN100
95 SHOTS=SHOTS-1:IFMID$(A$,ACROSS,1)=
" "THEN100
97 SCO=SCO+57:A$=LEFT$(A$,ACROSS-1)+"
 "+MID$(A$,ACROSS+1)
100 PRINT"[illegible]SCORE:";SCO;TAB(12)"SHOTS:
";SHOTS;" "
105 PRINTTAB(12)"LEFT"
110 IFSHOTS<1THENPRINTLEFT$(Y$,11);"T
HAT'S THE END OF THE GAME":STOP
```

key, and you move yourself left using the "Z", and right using the "C" 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 30) the program starts with a limit of 15 shots. In the last, most complex, version you will have 50 shots. The number of shots is deliberately kept low in the first version so you will not be able to get a high score just by leaving your finger on the "M" 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 press RUN then RETURN, to get it underway.

```
10 REM DUCK SHOOT
11 Y$="SQQQQQQQQQQQQQQQQQQQQQQQQ"
12 X$="]]]]]]]]]]]]]]]]]]]]]]]]]"
20 LET SCO=0:PRINT"♥";:POKE650,255
30 LET SHOTS=15
40 LET A$="ZAB  DK SL DF GFD FGG "
50 LET ACROSS=10
60 LET DOWN=15
70 PRINTLEFT$(Y$,8);A$
80 PRINTLEFT$(Y$,DOWN);LEFT$(X$,ACROS
S-1);" X ":GETK$
90 IFK$<>"M"THEN100
95 SHOTS=SHOTS-1:IFMID$(A$,ACROSS,1)=
" "THEN100
97 SCO=SCO+57:A$=LEFT$(A$,ACROSS-1)+"
 "+MID$(A$,ACROSS+1)
100 PRINT"SSCORE:";SCO;TAB(12)"SHOTS:
";SHOTS;"  "
105 PRINTTAB(12)"LEFT"
110 IFSHOTS<1THENPRINTLEFT$(Y$,11);"T
HAT'S THE END OF THE GAME":STOP
```

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

Line 40 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 22 characters long. You can check this by running the program briefly, stopping it with BREAK (RUN/STOP), then typing in as a direct command, PRINT LEN(A$). If your string is the correct length, PRINT LEN(A$), followed by RETURN will give you the answer 22.

The appearance of movement given to the ducks is created by use of Commodore 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 moment is line 130, which resets A$ equal to all of the string without its first character, that is, MID$(A$,2), 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 70) in the same position, at 8,0 (8 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.

The string handling also makes it very simple to cause the shot duck to disappear from the sky. As the string is 22 characters long, each character "shot" can be referred to by MID$(A$,N,1). That is MID$(A$,1,1) is the first element of the string, MID$(A$,2,1) is the second one and so on, until MID$(A$,22,1) is the very last spot within the string.

Look at line 9Ø. 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 9Ø, that K$ does not equal "M" (that is, you are not pressing the "M" key) then it jumps to line 1ØØ, otherwise it continues straight to line 95. Here it decrements the variable SHOTS by one. Then it hits another IF/THEN condition, which makes use of the Commodore 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 8Ø, a three-element string, with a space either side of the "X" is printed at ACROSS *minus one*, 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 95 discovers that MID$(A$,ACROSS,1) is a space, you have missed, so the program jumps to line 1ØØ. Otherwise the variable SCO is incremented by 57 and, finally in line 97, that element of A$ is set to a blank, so the "duck" disappears.

Now all this takes some time to explain, but you will find the computer does it apparently instantaneously. You press "M" 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 11Ø) that the program continues until you run out of shots, when the game terminates. 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 a complete new line 15, and modifying certain other lines.

```
10 REM DUCK SHOOT
11 Y$="[illegible]"
12 X$="[illegible]I"
15 POKE36879,25
20 LET SCO=0:PRINT"[illegible]";:POKE650,255
30 LET SHOTS=15
40 LET A$="ZAB  DK SL DF GFD FGG "
50 LET ACROSS=10
60 LET DOWN=15
70 POKE646,INT(RND(1)*6+2):PRINTLEFT$
(Y$,8);A$
80 POKE646,2:PRINTLEFT$(Y$,DOWN);LEFT
$(X$,ACROSS-1);" X ":GETK$
90 IFK$<>"M"THEN100
95 SHOTS=SHOTS-1:IFMID$(A$,ACROSS,1)=
" "THEN100
97 SCO=SCO+57:A$=LEFT$(A$,ACROSS-1)+"
 "+MID$(A$,ACROSS+1)
100 PRINT"[illegible]SCORE:";SCO;TAB(12)"SHOTS
:";SHOTS;" "
105 PRINTTAB(12)"[illegible]LEFT"
110 IFSHOTS<1THENPRINTLEFT$(Y$,11);"T
HAT'S THE END OF THE GAME":STOP
120 ACROSS=ACROSS+(K$="Z")-(K$="C")
130 A$=MID$(A$,2)+LEFT$(A$,1)
140 GOTO 70
```

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. Line 15 sets the background colour to white, as well as the border, clears the screen and sets the foreground colour to red.

Even if we had done nothing else there would be a significant improvement in the program, the colours are far more interesting than a two tone display.

However, we want to add two more commands to the program which will alter the display for the better. These are in line 70 and 100. The RND function is used to choose a colour at random for each time the ducks and your score are reprinted. As you'll see when the program is running, the change occurs so rapidly the ducks appear to shimmer through the spectrum as they fly across the screen, and even though it takes the computer an appreciable time to generate a random number, the effect on the speed of the program appears to be nil.

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, by adding some sound, and getting the BORDER (the area around the picture) 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 on the program.

```
10 REM DUCK SHOOT
11 Y$="
12 X$="
15 POKE36879,25
20 LET SCO=0:PRINT"
22 V1=15:V2=15:S1=0:S2=0:S3=0:S4=0:DU
=1
25 FORG=1TO20:S2=G+200:GOSUB9000:NEXT
G
30 LET SHOTS=15
35 S2=0:FOR G=50TO20STEP-2:S3=200+G:G
OSUB9000:NEXTG:S3=0
40 LET A$="ZAB  DK SL DF GFD FGG "
50 LET ACROSS=10
60 LET DOWN=15
70 POKE646,INT(RND(1)*6+2):PRINTLEFT$
(Y$,8);A$:S2=128+ASC(A$):GOSUB9000:S2
=0
80 POKE646,2:PRINTLEFT$(Y$,DOWN);LEFT
$(X$,ACROSS-1);" X ":GETK$:S3=180+ACR
OSS:GOSUB9000
82 S3=0
90 IFK$<>"M"THEN100
95 S3=150+SHOTS*3:GOSUB9000:S3=0:SHOT
S=SHOTS-1:IFMID$(A$,ACROSS,1)=" "THEN
100
97 SCO=SCO+57:A$=LEFT$(A$,ACROSS-1)+"
 "+MID$(A$,ACROSS+1):S3=200-SHOTS:GOS
UB9000:S3=0
98 POKE36879,PEEK(36879)AND248ORRND(1
)*8:POKE36879,PEEK(36879)AND248ORRND(
1)*8
99 POKE36879,PEEK(36879)AND248ORRND(1
)*8:POKE36879,PEEK(36879)AND248OR1
100 PRINT"
```

```
:";SHOTS;" "
105 PRINTTAB(12)"■LEFT"
110 IFSHOTS<1THENPRINTLEFT$(Y$,11);"T
HAT'S THE END OF THE GAME":STOP
120 ACROSS=ACROSS+(K$="Z")-(K$="C")
130 A$=MID$(A$,2)+LEFT$(A$,1)
140 GOTO 70
8999 REM SOUND ROUTINE
9000 POKE36878,PEEK(36878)AND240
9005 POKE36874,S1:POKE36875,S2:POKE36
876,S3:POKE36877,S4
9010 DV=(V2-V1)/DU
9015 FORI=2TODU
9020 V1=V1+DV:POKE36878,PEEK(36878)AN
D240 OR V1
9030 NEXTI:POKE36878,PEEK(36878)AND24
0
9040 POKE36874,0:POKE36875,0:POKE3687
6,0:POKE36877,0
9050 RETURN
```

The new lines are 25 and 35 which use the sound routine to create two "loops" of sound before the program gets underway. Refer back to the section on sound if you wish to refresh your memory at this point.

Line 25 is a loop, using G as the control variable. The loop runs from one to 20, and each value of G is used in the second part of line 25 to create a tone, which — because G is increasing — rises rapidly. The duration parameter is set at 1 which is close to the shortest sound which we have found which can be heard clearly. Line 35 produces another loop, this time counting downward. You'll discover that different STEP sizes produce quite different types of loop effects, and you may well wish to change the STEP in both this line and in line 25.

These two loops, however, are little more than "window dressing" designed to produce a good starting effect. The other beeps used by contrast, are related to other things happening within the program when it is running. You'll see that a sound call has been added at the end of line 7Ø. This takes the ASC (the number which the computer uses to refer to the character being printed, so PRINT CHR$(42) produces character 42, which you can look up on the ASC/CHR$ section earlier in this book) of the *first element* of the string A$, that is the element which is further to the left, and creates a tone from this. The effect of this is to produce a short "beep" just before a duck flies off the screen to the left.

You'll see another sound call at the end of line 8Ø. This one is sounded every time the program cycles and, because as the variable ACROSS gets bigger, the pitch of the note gets higher, you'll find that moving your "X" across the screen to the right will produce a constant higher tone, while moving it to the left will lower the tone.

Perhaps the most interesting sounds are in line 95. Firstly, you get a beep, related to the number of shots you have left, every time you touch "M", whether you hit anything or not. Run the program, and hold your finger on "M", and you'll hear the tone steadily decrease till the "THAT'S THE END OF THE GAME", message appears. Line 95, as you know, checks to see if your shot has hit anything (that is, it checks to see if that particular element of the string A$ is equal to a space), and if it finds that it is not a space (that is, that a "duck" is there, and has just that instant been shot) the computer — as well as increasing your score by 57 — beeps again, with a tone which although different to the first one in the line, is related to the number of shots left. This means that if you have a successful shot, you'll hear first the tone (which falls with each shot fired) from the first part of line 95, followed by a tone (which rises as the SHOTS variable is decremented) which signals that a duck has been shot. So

you hear two tones, in rapid succession. And if you're not quick in taking your finger off the "trigger" you'll hear a third one, or even more.

Lines 98 and 99, which change the border colour four times, are — of course — only triggered if you've downed a duck. It has the effect of creating border flashes very quickly in random colours, before reverting to white. The delay caused by this flash is very short, and gives good visual feedback, to back up the feedback from the sound to tell you that you've bagged another duck.

The next version of DUCK SHOOT we'll look at is, as you can see, considerabaly different from the ones we've been examining to date. At the very least, line 40 now looks extremely strange. This line is where a user-defined graphic (which, believe it or not, *does* look like a duck when the program is running) takes the place of the randomly-chosen letters. No matter how hard you look, you will not find anything that looks like that little duck on the keyboard. So where has it come from?

```
10 REM DUCK SHOOT
11 Y$="[illegible]":POKE5
5,255:POKE56,27
12 X$="[illegible]":GOSUB1
50
15 POKE36879,25:POKE36869,255
20 LET SCO=0:PRINT"[illegible]";:POKE650,255
22 V1=15:V2=15:S1=0:S2=0:S3=0:S4=0:DU
=1
25 FORG=1TO20:S2=G+200:GOSUB9000:NEXT
G
30 LET SHOTS=15
35 S2=0:FOR G=50TO20STEP-2:S3=200+G:G
OSUB9000:NEXTG:S3=0
40 LET A$="@@@  @@ @@ @@ @@@ @@@ "
50 LET ACROSS=10
```

```
60 LET DOWN=15
70 POKE646,INT(RND(1)*6+2):PRINTLEFT$
(Y$,8);A$:S2=250-SHOTS:GOSUB9000:S2=0
80 POKE646,2:PRINTLEFT$(Y$,DOWN);LEFT
$(X$,ACROSS-1);"■ X ":GETK$:S3=180+AC
ROSS:GOSUB9000
82 S3=0
90 IFK$<>"M"THEN100
95 S3=150+SHOTS*3:GOSUB9000:S3=0:SHOT
S=SHOTS-1:IFMID$(A$,ACROSS,1)=" "THEN
100
97 SCO=SCO+57:A$=LEFT$(A$,ACROSS-1)+"
 "+MID$(A$,ACROSS+1):S3=200-SHOTS:GOS
UB9000:S3=0
98 POKE36879,PEEK(36879)AND248ORRND(1
)*8:POKE36879,PEEK(36879)AND248ORRND(
1)*8
99 POKE36879,PEEK(36879)AND248ORRND(1
)*8:POKE36879,PEEK(36879)AND248OR1
100 PRINT"■■SCORE:";SCO;TAB(12)"SHOTS
:";SHOTS;" "
105 PRINTTAB(12)"■LEFT"
110 IFSHOTS<1THENPRINTLEFT$(Y$,11);"T
HAT'S THE END OF THE GAME":POKE36869,
240:STOP
120 ACROSS=ACROSS+(K$="Z")-(K$="C")
130 A$=MID$(A$,2)+LEFT$(A$,1)
140 GOTO 70
150 FORA=0TO7
160 READB
170 POKE7168+A,B
180 NEXTA:FORA=7424TO7431:POKEA,0:NEX
TA
190 RETURN
200 DATA0,4,73,222,62,8,0,0
```

```
8999 REM SOUND ROUTINE
9000 POKE36878,PEEK(36878)AND240
9005 POKE36874,S1:POKE36875,S2:POKE36
876,S3:POKE36877,S4
9010 DV=(V2-V1)/DU
9015 FORI=2TODU
9020 V1=V1+DV:POKE36878,PEEK(36878)AN
D240 OR V1
9030 NEXTI:POKE36878,PEEK(36878)AND24
0
9040 POKE36874,0:POKE36875,0:POKE3687
6,0:POKE36877,0
9050 RETURN
```

The duck has been 'user-defined'. User-defined graphics are one of the really great features of your computer.

The routine from line 150 onwards reads in the data for the UDG (user-defined graphic) for the duck. This replaces the "@" sign. Details of this will be explained in chapter fourteen.

Note by the way, that there is a change to the end of line 70 in this version of the program with the user-defined duck. Try working out a duck-shape of your own, on a grid you have drawn up yourself, and enter the required numbers in line 200.

The final version of this program we will discuss has three rows of flying ducks. It is best to aim at the middle row of ducks (which fly more quickly than the bottom row) because they are worth 517 points, as opposed to the 57 that each of the ducks on the bottom row is worth. The top row of ducks is just there to confuse you. They disappear automatically as the ducks in the middle row are shot, but cannot be shot directly, and do not contribute to your score.

The middle row of ducks is held in the string B$ which is set equal to A$ in line 45, as you can see in the listing:

```
10 REM DUCK SHOOT
11 Y$="QQQQQQQQQQQQQQQQQQQQQQQQ":POKE5
5,255:POKE56,27
12 X$="]]]]]]]]]]]]]]]]]]]]]]]]":GOSUB1
50
15 POKE36879,25:POKE36869,255
20 LET SCO=0:PRINT"♥♥";:POKE650,255
22 V1=15:V2=15:S1=0:S2=0:S3=0:S4=0:DU
=1
25 FORG=1TO20:S2=G+200:GOSUB9000:NEXT
G
30 LET SHOTS=15
35 S2=0:FOR G=50TO20STEP-2:S3=200+G:G
OSUB9000:NEXTG:S3=0
40 LET A$="@@@  @@ @@ @@ @@@ @@@ "
45 LET B$=A$
50 LET ACROSS=10
60 LET DOWN=15
70 POKE646,INT(RND(1)*6+2):PRINTLEFT$
(Y$,8);A$:S2=250-SHOTS:GOSUB9000:S2=0
75 PRINTLEFT$(Y$,6);B$;LEFT$(Y$,5);MI
D$(B$,3);LEFT$(B$,2)
80 POKE646,2:PRINTLEFT$(Y$,DOWN);LEFT
$(X$,ACROSS-1);"▓ X ":GETK$:S3=180+AC
ROSS:GOSUB9000
82 S3=0
90 IFK$<>"M"THEN100
92 S3=150+SHOTS*3:GOSUB9000:S3=0:SHOT
S=SHOTS-1:IFMID$(A$,ACROSS,1)=" "THEN
100
94 SCO=SCO+57:A$=LEFT$(A$,ACROSS-1)+"
 "+MID$(A$,ACROSS+1):GOTO97
```

```
95 IFMID$(B$,ACROSS,1)=" "THEN100
96 SCO=SCO+517:B$=LEFT$(B$,ACROSS-1)+
" "+MID$(B$,ACROSS+1)
97 S3=200-SHOTS:GOSUB9000:S3=0
98 POKE36879,PEEK(36879)AND248ORRND(1
)*8:POKE36879,PEEK(36879)AND248ORRND(
1)*8
99 POKE36879,PEEK(36879)AND248ORRND(1
)*8:POKE36879,PEEK(36879)AND248OR1
100 PRINT"■■SCORE:";SCO;TAB(12)"SHOTS
:";SHOTS;" "
105 PRINTTAB(12)"■LEFT"
110 IFSHOTS<1THENPRINTLEFT$(Y$,11);"T
HAT'S THE END OF THE GAME":POKE36869,
240:STOP
120 ACROSS=ACROSS+(K$="Z"AND ACROSS>1
)-(K$="C"AND ACROSS<21)
130 A$=MID$(A$,2)+LEFT$(A$,1)
135 B$=MID$(B$,3)+LEFT$(A$,2)
140 GOTO 70
150 FORA=0TO7
160 READB
170 POKE7168+A,B
180 NEXTA:FORA=7424TO7431:POKEA,0:NEX
TA
190 RETURN
200 DATA0,4,73,222,62,8,0,0
8999 REM SOUND ROUTINE
9000 POKE36878,PEEK(36878)AND240
9005 POKE36874,S1:POKE36875,S2:POKE36
876,S3:POKE36877,S4
9010 DV=(V2-V1)/DU
9015 FORI=2TODU
9020 V1=V1+DV:POKE36878,PEEK(36878)AN
D240 OR V1
```

```
9030 NEXTI:POKE36878,PEEK(36878)AND24
0
9040 POKE36874,0:POKE36875,0:POKE3687
6,0:POKE36877,0
9050 RETURN
```

The middle row of ducks is shot at in line 96. If you shoot a duck in the bottom row, your shot ends there. After all, you cannot expect it to continue on to get a duck from the middle row as well. The GOTO 100 at the end of line 94 ensures that this does not happen.

Line 135 moves the middle and top rows of ducks, changing the elements in the string by an extra element compared to the changes occuring in line 130. Line 70 prints all three rows of ducks, "inventing" the top (dummy) row by printing B$ out of register, so the ducks there appear ahead of, although flying in a synchronisation with, those in the middle row. this will be clear when you run the program.

That brings us to the end of this series of DUCK SHOOT games. There are four things you can do to further develop the program:

- Cut the number of shots available down to make it more challenging.
- User-define the man, so it is not just an "X".
- Add a "high score" feature, so the game will restart, preserving a high score you can try and better.
- Allow the computer to detect when all the A$ and B$ ducks have been shot (that will happen when A$ and B$ are just 32 spaces long, and contain nothing but spaces) and add a bonus to the score if this occurs before all the shots have been fired.

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).

The next program *dimensions* (this, as we said, simply means sets up, or creates) an array called A, with room for fifteen elements. The B loop from lines 3Ø to 5Ø fills the array with random digits between zero and nine, and then prints them back for you with the loop from lines 7Ø to 9Ø.

```
10 REM ARRAYS
20 DIM A(15)
30 FOR B=1 TO 15
40 A(B)=INT(RND(1)*9)
```

```
50 NEXT B
60 FOR Z=1 TO 15
70 PRINT"A(";Z;") IS";A(Z)
80 FOR I=1 TO 100:NEXT I
90 NEXT Z
```

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 MULTI-D ARRAYS
20 DIM A(4,4)
30 FOR B=1 TO 4
40 FOR C=1 TO 4
50 A(B,C)=INT(RND(1)*9)
60 NEXT C
70 NEXT B
80 PRINT"□□    1  2  3  4"
90 FORB=1 TO 4
95 PRINT"□";B;"■■ ■";
100 FOR C=1 TO 4
110 PRINT A(B,C);
120 NEXT C
130 PRINT
140 NEXT B
```

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

```
■■■■ 1  2  3  4
■■ 8  8  8  0
■■ 8  7  8  7
■■ 6  3  2  2
■■ 3  5  8  7
```

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 Ø at the end of the line is A(1,4) and the 8 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 STRING ARRAYS
20 DIM A$(5)
30 FOR B=1 TO 5
40 LET A$(B)=CHR$(INT(RND(1)*26+65))
50 NEXT B
60 FOR B=1 TO 5
70 PRINT "A$(";B;") IS ";A$(B)
80 NEXT B
```

If you refer to an array that has not been dimensioned beforehand, the computer automatically dimensions one with eleven elements (Ø to 1Ø). You will get an error message if you refer to an element outside the arrays range. Notice also that you cannot redimension an array with the same name. To do this you must clear all the variables first with the command CLR. Beware though, this command will make your VIC forget all the variables in its memory (incidentally, so does RUN).

```
10 REM STRING ARRAYS II
20 DIM A$(5)
30 FOR B=1 TO 5
40 READ A$(B)
50 NEXT B
60 PRINT A$(INT(RND(1)*5+1));
70 GOTO 60
100 DATAHI THERE, I'M CALLED VIC
110 DATAHERE'S LOOKING AT YOU KID
120 DATAHELP, FAR OUT MAN!
```

```
FAR OUT MAN!
HELP
HELP
HELP
HERE'S LOOKING AT YOU KID
HELP
HI THERE
HERE'S LOOKING AT YOU KID
HI THERE
HERE'S LOOKING AT YOU KID
FAR OUT MAN!
HI THERE
I'M CALLED VIC
FAR OUT MAN!
FAR OUT MAN!
FAR OUT MAN!
I'M CALLED VIC
```

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 with the water into which it has been placed.

Here is the program listing:

```
10 REM DRAGON'S LAIR
15 POKE56,27:POKE55,255
20 DIMA(10,10):M=0
30 GOSUB500
40 X=INT(RND(1)*2)
50 IFX=0THENP=P+1
60 IFX=1THENP=P-1
70 X=INT(RND(1)*2)
80 IFX=0THENQ=Q+1
90 IFX=1THENQ=Q-1
100 IFQ<1THENQ=Q+1
110 IFP<1THENP=P+1
120 M=M+1
130 PRINT"■■■■■■■■■■■■RTTEMPT #";M
420 FORX=1TO10
430 FORY=1TO10
432 GR=1:CO=0
435 IFX=QANDY=PTHENGR=0:CO=2
440 POKE38470+Y*22+X,CO
445 POKE7750+Y*22+X,GR
450 NEXT Y,X
480 A(P,Q)=0
490 IFQ>8ORP>8THEN600
495 GOTO 40
500 Q=INT(RND(1)*3)+4
505 P=INT(RND(1)*3)+4
510 POKE36869,255
520 FORI=7168TO7183:READX
530 POKEI,X:NEXTI
560 POKE36879,26:PRINT"■"
580 RETURN
590 DATA35,70,76,95,255,223,18,51
595 DATA0,8,8,20,38,20,0,0
600 POKE36869,240
610 PRINT"■■■■■■■WHEW..HOME AT LAST"
```

DAVY JONES' LOCKER

You can relax now 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 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:

```
5 POKE36878,10
10 REM FULL FATHOM FIFTY
15 POKE 36879,25:PRINT""
20 DIM B(2):DIM D(2):RO=1
25 A$="                         "
30 FOR A=1 TO 2
40 PRINT"FULL FATHOM FIFTY":P
RINT"ROUND NU
MBER";RO;"  "
50 PRINT"VIC";B(1
)
52 PRINT"HUMAN";B(2)
55 POKE 36879,25+INT(RND(1)*7)
60 IF B(1)>49 OR B(2)>49 OR B(2)>49 T
HEN GOTO 230
70 IF A=1 THEN PRINT"I
'LL NOW ROLL THE DIE":GOTO 90
```

```
80 PRINT"  PRESS 'R' T
O ROLL    "
85 GET Z$:IF Z$="R" THEN 90
86 POKE 36874,200:POKE 36875,200:POKE
 36876,200:FOR I=1 TO 40
87 NEXT I:POKE 36874,0:POKE 36875,0:P
OKE 36876,0
88 FOR I=1 TO 50:NEXT I:GOTO 85
90 FORJ=200 TO 250:POKE 36875,J:POKE
36876,J:NEXT J:POKE 36875,0:POKE 3687
6,0
100 PRINT"";A$
110 FOR C=1 TO 2
120 PRINT"    ROLLING
DIE #";C
130 FOR M=240 TO 150 STEP-.5:POKE 368
76,M:NEXT M:POKE 36876,0
140 LET D(C)=INT(RND(1)*6)+1
150 PRINT"    IT CAME
UP";D(C);"    "
160 FOR G=200 TO 250 STEP.4:POKE 3687
4,G:NEXT G:POKE 36874,0
165 PRINT"";A$
170 NEXT C
180 IF D(1)=D(2) THEN 190
185 PRINT" THEY DO NOT
 COUNT":GOTO 200
190 PRINT"THE ROLL WA
S";D(1);"AND";D(2):LET B(A)=B(A)+3*D(
1)
200 FOR G=150 TO 200:POKE 36875,G:NEX
T G:POKE 36875,0
205 PRINT"";A$
210 NEXT A
220 RO=RO+1:FOR G=1 TO 50:POKE 36879,
```

```
24+INT(RND(1)*8):NEXT G:POKE 36879,25
:GOTO 30
230 PRINT"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒AND THE
 WINNER IS THE"
240 IF B(1)<B(2) THEN PRINT"▒       ▒▒
 HUMAN "
250 IF B(1)>B(2) THEN PRINT"▒       ▒▒
  VIC  "
```

CHAPTER FOURTEEN
LET THE GOOD GRAPHICS ROLL

As well as the graphics available to you from the keyboard, the VIC allows you to define your own characters.

If you missed out the section on binary in chapter twelve, now might be a good time to go back and read it. All the characters on your VIC are made up of 64 dots, arranged in 8 rows of 8 dots. Can you see the relation between this and the binary system? Because each row is made up of 8 dots which can be either on or off, a row can be represented by an 8 bit number where each bit can be either 1 or Ø. Because an 8 bit number is a byte, and there are 8 rows of dots in a character, 8 bytes are required to define a character. If we imagine an 8 by 8 grid, with each row corresponding to a byte. Every space in the grid is represented by a Ø in the binary number, and every filled in block is represented by a 1. We can then change the resulting 8 binary numbers into decimal as was shown in chapter twelve. Example, the letter "A":

CHARACTER	BINARY	DECIMAL
_ _ _ * * _ _ _	ØØØ11ØØØ	24
_ _ * _ _ * _ _	ØØ1ØØ1ØØ	36
_ * _ _ _ _ * _	Ø1ØØØØ1Ø	66
_ * * * * * * _	Ø1111111Ø	126
_ * _ _ _ _ * _	Ø1ØØØØ1Ø	66
_ * _ _ _ _ * _	Ø1ØØØØ1Ø	66
_ * _ _ _ _ * _	Ø1ØØØØ1Ø	66
_ _ _ _ _ _ _ _	ØØØØØØØØ	Ø

Can you see the correspondence? Your VIC has all its characters stored in memory in this way, 8 bytes per character. They are stored in ROM (so you can't alter them) starting at location 32768. The data for the letter "A" starts at location 32776. Use the PEEK function to see if the numbers are what they should be.

If the characters are in ROM, how then are we going to change them or add our own? As was mentioned in chapter twelve, one of the registers of the VIC chip allows us to tell the computer where to get the data for the characters. By POKEing this register we can make it think the character data is in an area of RAM (which we can alter) where we have designed our own characters.

When we do this, we must store our character data at the top of RAM (with a large address) which is above our program. We must be careful not to POKE into the RAM that holds our program or variables. To this end we make the VIC think that it has less memory than it does. By lowering the top of user memory (as the RAM where your program is stored is called) we can put our graphic data above this, and it will not be touched by the program.

Register 5 of the VIC chip (location 36869) normally holds 240, which makes the VIC get its character data from the ROM. When we design our own characters, they are normally stored at location 7168 onwards. Therefore, we have to make the VIC think that the top of memory is location 7167. To do this, we carry out the following POKEs:

```
POKE 55,255:POKE 56,27
```

We usually store the data for our characters in DATA statements, which are then READ and POKEd into memory.

It is usually a good idea to design a space so that the screen does not become a mess. For a space, all you have to do is POKE locations 7424–7431 with zeros. This is demonstrated by the next program, which designs and animates a space invader.

```
9 REM FRED THE INVADER
10 POKE 55,255:POKE 56,27
15 Y$="
17 X$="
20 GOSUB 1000
29 REM SET UP VIC CHIP FOR UDG
30 POKE 36869,255:PRINT"
39 REM BLACK BACKGROUND AND BORDER
40 POKE 36879,8
49 REM GREEN INVADER
50 X=11:Y=11:DX=INT(RND(1)*3-1):DY=IN
T(RND(1)*3-1)
59 REM PRINT
60 PRINT LEFT$(Y$,Y);LEFT$(X$,X);"
69 REM PAUSE
70 FOR I=1 TO 50:NEXT I
79 REM RUBOUT
80 PRINT LEFT$(Y$,Y);LEFT$(X$,X);" "
90 IF RND(1)>0.8 THEN DX=INT(RND(1)*3
-1):DY=INT(RND(1)*3-1)
100 X=X+DX:Y=Y+DY
110 IF X<0 OR X>21 THEN DX=-DX:X=X+DX
120 IF Y<1 OR Y>22 THEN DY=-DY:Y=Y+DY
130 GOTO 60
999 REM USER DEFINED CHARACTERS
1000 FOR I=7424 TO 7431:POKE I,0:NEXT
 I
1010 FOR I=7168 TO 7175:READ A:POKE I
,A
1020 NEXT I:RETURN
```

```
1030 DATA 56,124,214,214,124,56,84,14
6
```

If we start our characters at location 7168, when we use reverse mode, instead of user defined graphics, you get the normal graphics — but not in reverse mode, in normal mode. In this way you can mix normal and user defined graphics. This is shown in the next program, which prints up a score while the space invader moves around.

```
9 REM FRED THE INVADER
10 POKE 55,255:POKE 56,27
15 Y$="
17 X$="
20 GOSUB 1000
29 REM SET UP VIC CHIP FOR UDG
30 POKE 36869,255:PRINT "   SCORE: 00
0000"
35 PRINT "        HI THERE!":PR
INT "  I'M FRED THE INVADER"
39 REM BLACK BACKGROUND AND BORDER
40 POKE 36879,8
49 REM GREEN INVADER
50 X=11:Y=11:DX=INT(RND(1)*3-1):DY=IN
T(RND(1)*3-1)
59 REM PRINT
60 PRINT LEFT$(Y$,Y);LEFT$(X$,X);"  "
69 REM PAUSE
70 FOR I=1 TO 50:NEXT I
79 REM RUBOUT
80 PRINT LEFT$(Y$,Y);LEFT$(X$,X);"  "
90 IF RND(1)>0.8 THEN DX=INT(RND(1)*3
-1):DY=INT(RND(1)*3-1)
100 X=X+DX:Y=Y+DY
110 IF X<0 OR X>21 THEN DX=-DX:X=X+DX
120 IF Y<1 OR Y>22 THEN DY=-DY:Y=Y+DY
130 GOTO 60
```

```
999 REM USER DEFINED CHARACTERS
1000 FOR I=7424 TO 7431:POKE I,0:NEXT
 I
1010 FOR I=7168 TO 7175:READ A:POKE I
,A
1020 NEXT I:RETURN
1030 DATA 56,124,214,214,124,56,84,14
6
```

Another way of doing this would be to PEEK the characters you wanted from the ROM and POKE them into RAM together with your own. The next program prints messages in reverse text (which you can't get with the above method) together with the space invader. The above method should be used if possible, because this method wastes valuable memory which could be used for more of your own graphics.

```
9 REM FRED THE INVADER
10 POKE 55,255:POKE 56,27
15 Y$="
17 X$="
20 GOSUB 1000
29 REM SET UP VIC CHIP FOR UDG
30 POKE 36869,255:PRINT "
0000"
35 PRINT "                HI THERE":PRIN
T " I AM FRED THE INVADER"
39 REM BLACK BACKGROUND AND BORDER
40 POKE 36879,8
49 REM GREEN INVADER
50 X=11:Y=11:DX=INT(RND(1)*3-1):DY=IN
T(RND(1)*3-1)
59 REM PRINT
60 PRINT LEFT$(Y$,Y);LEFT$(X$,X);"
69 REM PAUSE
70 FOR I=1 TO 50:NEXT I
```

```
79 REM RUBOUT
80 PRINT LEFT$(Y$,Y);LEFT$(X$,X);" "
90 IF RND(1)>0.8 THEN DX=INT(RND(1)*3
-1):DY=INT(RND(1)*3-1)
100 X=X+DX:Y=Y+DY
110 IF X<0 OR X>21 THEN DX=-DX:X=X+DX
120 IF Y<1 OR Y>22 THEN DY=-DY:Y=Y+DY
130 GOTO 60
999 REM USER DEFINED CHARACTERS
1000 FOR I=7424 TO 7431:POKE I,0 NEXT
 I
1010 FOR I=7168 TO 7175:READ A:POKE I
,A
1020 NEXT I
1030 DATA 56,124,214,214,124,56,84,14
6
1039 REM READ DOWN REVERSE ALPHABET
1040 FOR I=0 TO 207:REM 26 LETTERS TI
MES 8 BYTES
1050 POKE 7176+I,PEEK(33800+I):NEXT I
1060 RETURN
```

MULTI COLOUR MODE

Wouldn't it be nice if you could get more than one colour in a character square? Well you can (that's why we've got the heading), in fact you can have three colours not counting the background. These colours are as follows:

Background
Border
Auxiliary
Foreground

The background colour is the same as normal.
The border colour is the same as normal.

The auxiliary colour can be one of sixteen (numbered 0–15)

and is set by:

POKE 36878, PEEK(36878) AND 15 OR (N*16) where n is the number of the colour.

The foreground colour can be one of eight as normal, but 8 is added to the number when it is POKED to colour memory or location 646 (the current colour indicator). Adding 8 lets the computer know that we want the character to be in multi colour mode.

Because there are four different colour combinations, we can represent this with two bits (counting between Ø and 3). If we now have two bits per dot in a row, we can only have four dots in a row. Therefore multi coloured characters are composed of four dots across by eight down. The eight bits of each byte are split into four pairs, and each pair is interpreted as follows:

ØØ — the dot is printed in the background colour.
Ø1 — the dot is printed in the border colour.
1Ø — the dot is printed in the foreground colour.
11 — the dot is printed in the auxiliary colour.

Multi colour and normal modes can be mixed as this next program shows:

```
10 REM MULTI-COLOUR
15 POKE56,27:POKE55,255
20 GOSUB1000
25 PRINT"[redacted]THERE!"
50 FOR C=8TO15
55 POKE646,C
60 PRINT"[redacted]ABC[redacted]DEFG"
65 FORN=ØTO15
70 POKE36878,PEEK(36878)AND15OR(N*16)
75 FORI=1TO50:NEXTI
80 NEXTN
85 NEXTC:GOTO50
1000 FORI=7424TO7431:POKEI,Ø:NEXTI
```

```
1005 FORI=7168TO7231:READA
1010 POKEI,A:NEXT
1015 POKE36869,255:POKE36879,26
1020 POKE36878,PEEK(36878)AND15OR(7*16)
1025 RETURN
9000 DATA0,160,164,165,165,165,170,170
9005 DATA0,10,10,10,10,10,170,170
9010 DATA0,0,64,80,80,80,80,80
9015 DATA0,160,164,244,252,0,160,164
9020 DATA175,167,165,165,165,245,253,255
9025 DATA250,250,250,10,10,15,15,15
9030 DATA80,80,80,80,80,80,208,240
9035 DATA165,165,165,165,165,245,253,255
```

Remember the space invader? If we want to make him appear a bit more lively, we could make his legs move as he went about the screen. There are two ways of doing this, the first is to define two graphics corresponding to the two positions of the invader's legs, and print them alternately:

```
10 POKE56,27:POKE55,255
20 GOSUB 1000
30 PRINT"J"
40 X=0:Y=1:GR=0:CO=5:D=1
50 POKE 7680+Y*22+X,32
60 X=X+D
70 IFX=0 ORX=21THEND=-D:Y=Y+1
80 IF Y>20 THEN30
90 POKE 38400+Y*22+X,CO
100 POKE 7680+Y*22+X,GR
110 FORI=1TO200:NEXTI
120 GR=1-GR
130 GOTO50
1000 FORI=7424TO7431:POKEI,0:NEXTI
1010 FORI=7168TO7183:READA
1020 POKEI,A:NEXTI
```

```
1030 POKE36879,8:POKE36869,255
1040 RETURN
9000 DATA34,68,124,84,124,56,68,130
9010 DATA136,68,124,84,124,56,68,40
```

The second way would be to redefine the actual character itself, without having to reprint it:

```
10 POKE56,27:POKE55,255
20 GOSUB 1000
30 PRINT"J"
40 X=0:Y=1:GR=0:CO=5:D=1
50 POKE 7680+Y*22+X,32
60 X=X+D
70 IFX=0 ORX=21THEND=-D:Y=Y+1
80 IF Y>20 THEN30
90 POKE 38400+Y*22+X,CO
100 POKE 7680+Y*22+X,GR
110 FORI=1TO200:NEXTI
120 IFPEEK(7168)=34THEN140
130 POKE7168,34:POKE7175,130:GOTO150
140 POKE7168,136:POKE7175,40
150 GOTO50
1000 FORI=7424TO7431:POKEI,0:NEXTI
1010 FORI=7168TO7175:READA
1020 POKEI,A:NEXTI
1030 POKE36879,8:POKE36869,255
1040 RETURN
9000 DATA34,68,124,84,124,56,68,130
```

HIGH RESOLUTION GRAPHICS

High resolution graphics is the term used to describe the ability to switch on or off any dot on the screen. This takes a lot of memory, and since the unexpanded VIC doesn't have all that much to start with, we can only use a small screen. What we are going to do is termed 'bit-mapping' because each dot on the screen corresponds to a bit in memory.

What we must do is fill the entire screen with user defined graphics, each one different from the last. These are all initially defined as spaces (filled with zeros) so that the screen appears empty. When we want to plot a point (or pixel) on the screen, we work out which user defined graphic has to be altered and then set the required bit in the graphic to a 1. This instantly lights up the dot on the screen. The formula for calculating the user defined graphic to be altered is:

GR = INT(X/8) + INT(Y/8)*8

Where X is the horizontal coordinate (Ø at the left to 63 at the right) and Y is the vertical coordinate (Ø at the top to 63 at the bottom). The formula for calculating the correct row in the user defined graphic data is:

RO = 8*(Y/8 – INT(Y/8))

The location in memory which has to be POKEd is then:

LO = 7168 + GR*8 + RO

The decimal value which will be ORed with the correct byte is calculated by:

DV = 2↑(7 – (X – INT(X/8)*8))

Thus a program to plot a parabola might look like this:

```
10 REM PARABOLA
20 REM INITIALISE DISPLAY
30 POKE56,27:POKE55,255
40 PRINT"J":POKE36879,8
50 FORI=ØTO63
60 POKE7680+I,I:POKE38400+I,1
70 NEXTI
78 REM SHRINK SCREEN
79 REM HORIZONTALLY
80 POKE 36866,PEEK(36866)AND128OR8
90 POKE 36864,PEEK(36864)AND128OR25
99 REM VERTICALLY
```

```
100 POKE 36867,PEEK(36867)AND129OR16
110 POKE 36865,65
119 REM CLEAR UDGS
120 FORI=7168TO7679:POKEI,0:NEXTI
129 REM UDG MODE
130 POKE36869,255
199 REM PLOT PARABOLA
200 FORX1=-31TO31
210 Y1=X1↑2
219 REM SCALE VALUES
220 X=X1+31
230 Y=Y1/16
240 GOSUB 10000
250 NEXTX1
259 REM LOOP
260 GOTO 260
9999 REM HIRES PLOT
10000 IFX<0ORX>63ORY<0ORY>63THENRETURN
10005 Y=INT(63-Y+.5):X=INT(X+.5)
10010 GR=INT(X/8)+INT(Y/8)*8
10020 RO=8*(Y/8-INT(Y/8))
10030 LO=7168+GR*8+RO
10040 DV=2↑(7-(X-INT(X/8)*8))
10050 POKELO,PEEK(LO)OR DV
10060 RETURN
```

Notice how in lines 220 and 230, the X and Y coordinates are scaled before the subroutine is called. The coordinates are rounded in line 10005. Also notice the last line, which makes the computer go round and round forever, known as an "infinite loop". You can, of course, use the plotting routine from line 9999 onwards in your own programs (you must remember to include the initialisation from lines 20 to 130). The next program demonstrates the use of sines and cosines to plot ovals and circles.

```
10 REM OVALS
20 REM INITIALISE DISPLAY
30 POKE56,27:POKE55,255
40 PRINT"■":POKE36879,8
50 FORI=0TO63
60 POKE7680+I,I:POKE38400+I,1
70 NEXTI
78 REM SHRINK SCREEN
79 REM HORIZONTALLY
80 POKE 36866,PEEK(36866)AND128OR8
90 POKE 36864,PEEK(36864)AND128OR25
99 REM VERTICALLY
100 POKE 36867,PEEK(36867)AND129OR16
110 POKE 36865,65
119 REM CLEAR UDGS
120 FORI=7168TO7679:POKEI,0:NEXTI
129 REM UDG MODE
130 POKE36869,255
199 REM PLOT OVALS
200 FORA=1TO105STEP.3
210 B=π*A/50
220 C=31*COS(B)+31
230 D=31*SIN(B)+31
240 X=C:Y=D:GOSUB10000
250 X=C/2:Y=D:GOSUB10000
260 X=C:Y=D/2:GOSUB10000
270 X=C/2:Y=D/2:GOSUB10000
280 NEXTA
290 GOTO290
9999 REM HIRES PLOT
10000 IFX<0ORX>63ORY<0ORY>63THENRETURN
10005 Y=INT(63-Y+.5):X=INT(X+.5)
10010 GR=INT(X/8)+INT(Y/8)*8
```

```
10020 RO=8*(Y/8-INT(Y/8))
10030 LO=7168+GR*8+RO
10040 DV=2↑(7-(X-INT(X/8)*8))
10050 POKELO,PEEK(LO)OR DV
10060 RETURN
```

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 any one of those. 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 DEF FN command, which is used in lines 160, 180 and 200. Each time the program reaches the function FN SC(X) the computer calculates the formula in the DEF FN statement, replacing any occurrences of the letter in brackets in the definition with the letter in brackets in the call. Thus if we define a function named SC to find out what is on the screen:

```
DEF FN SC(A) = PEEK(7680 + Y*22 + A)
```

and we then call it with the statement:

```
PRINT FN SC(X)
```

This has the same effect as if we had typed:

```
PRINT PEEK(7680 + Y*22 + X)
```

The advantage of defining the function is that it saves typing and memory if you will be using that formula a few times. Notice that you cannot use words such as LET and PRINT etc., in the definition, only things which return a value. Functions must be named with one or two characters, the first must be a letter and the second can be a letter or number.

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 screen code (different from the ASC code), minus 48. You'll see why, if you look up the code of the "/" which is used for the walls, and the code of "1". Good rolling.

And this is the listing:

```
10 REM BAGATELLE
15 DEF FN SC(A)=PEEK(7680+Y*22+A):POK
E36878,15
17 Y$="
18 X$="
20 GOSUB 1000
```

```
30 GOSUB 500
35 FORY1=1 TO 5
40 FORX=1 TO 2:PRINTLEFT$(Y$,2);LEFT$
(X$,19);Y1
50 PRINTLEFT$(Y$,23);
55 IFX=2 THENPRINTY1;"ME ";
56 IFX=1 THENPRINTY1;"YOU";
57 PRINTLEFT$(Y$,21);
60 IFX=2THEN70
65 PRINT"WHICH NUMBER TO START":INPUT
"WITH";K:IFK<1 OR K>9THEN65
67 GOTO 80
70 K=INT(RND(1)*9+1)
75 PRINTLEFT$(Y$,21);"■I WILL START W
ITH■■■
77 FORE=1TO100:POKE36876,200-E/2::NEX
TE:POKE36876,0
80 PRINTLEFT$(Y$,1);LEFT$(X$,3+K);"■"
;K:FORE=200TO250
82 POKE36876,E:NEXTE:POKE36876,0:PRIN
TLEFT$(Y$,1);LEFT$(X$,3+K);"■■";K
90 BA=K+3:EAB=BA
100 BD=1:EDB=1
110 IA=1
120 ID=1
130 PRINTLEFT$(Y$,EDB+1);LEFT$(X$,EAB
);" ":EAB=BA
140 EDB=BD
150 PRINTLEFT$(Y$,BD+1);LEFT$(X$,BA);
"●"
160 FLAG=0:Y=BD+1:A=FN SC(BA)
170 IFA<>32 THENFLAG=1:GOTO220
180 Y=BD:A=FN SC(BA+1)
190 IFA<>32 THENFLAG=1:GOTO220
200 A=FN SC(BA-1)
```

```
210 IFA<>32 THENFLAG=1
220 IFFLAG=0THEN225
221 POKE36876,130:IF RND(1)>.3 THEN22
3
222 IA=-IA:ID=-ID:POKE36876,150+2*IA+
3*ID
223 POKE36876,0:IFA<27THENA=A+64
224 C(X)=C(X)+A-48
225 PRINTLEFT$(Y$,6);LEFT$(X$,18);C(1
);LEFT$(Y$,8);LEFT$(X$,18);C(2)
230 BD=BD+(ID*RND(1))+0.75
235 IF BA<3 THENIA=-IA:BA=5
236 IF BD<1 THENID=-ID:BD=3
237 IF BA>15THENIA=-IA:BA=13
240 BA=BA+(IA*RND(1))
250 IF BD<19 THEN130
260 FORG=128TO228:POKE36876,G:NEXTG:P
OKE36876,0
270 GOSUB500
280 NEXTX
290 NEXTY1
300 PRINTLEFT$(Y$,15);"THE WINNER IS"
;LEFT$(Y$,17);LEFT$(X$,5);
310 IFC(1)<C(2) THEN PRINT"THE MACHIN
E"
320 IFC(2)<C(1) THEN PRINT"THE HUMAN"
490 STOP
500 PRINT"  123456789    ":PRINT
LEFT$(Y$,5);LEFT$(X$,18);"HUME";LEFT$
(Y$,6);
502 PRINTLEFT$(X$,18);C(1);LEFT$(Y$,7
);LEFT$(X$,18);"VIC";LEFT$(Y$,8);
505 PRINTLEFT$(X$,18);C(2);LEFT$(Y$,1
);LEFT$(X$,18);"BALL"
507 PRINTLEFT$(X$,18);"
```

```
510 FORQ=1 TO 19
520 POKE646,2+RND(1)*3:PRINTLEFT$(Y$,
Q+1);"■■/";LEFT$(Y$,Q+1);LEFT$(X$,17)
;"/"
530 NEXTQ
535 PRINTLEFT$(Y$,19);"■■/////////////
/////"
540 PRINT"■■/////////////////////"
550 PRINTLEFT$(Y$,3);LEFT$(X$,5);"1
1  1";LEFT$(Y$,5);LEFT$(X$,3);"Z X  Z
  X  Z"
560 POKE646,2+RND(1)*5:PRINTLEFT$(Y$,
7);LEFT$(X$,4);"1 11 11 1"
570 POKE646,2+RND(1)*5:PRINTLEFT$(Y$,
10);LEFT$(X$,3);"7   7   7  7"
580 POKE646,2+RND(1)*5:PRINTLEFT$(Y$,
12);LEFT$(X$,3);"5   5   5  5"
590 POKE646,2+RND(1)*5:PRINTLEFT$(Y$,
15);LEFT$(X$,6);"9   9   9"
595 PRINTLEFT$(Y$,17);LEFT$(X$,4);"8
  8   8"
600 RETURN
1000 DIM C(2)
1060 POKE36879,30:PRINT"■■";
1070 RETURN
```

REPTILE

In this program, by Paul Toland, you are in control of a snake, and you must try to grow your snake as long as possible, by directing it to the pound signs which it eats. However, the £s remain on the screen for only a short time, after which they turn into poisonous dollar signs. The snake will also die if it hits the surrounding border, or itself. Once the game is over, you'll be told how long you became.

Notice how we use the function to look at the screen in this program as well.

```
0 POKE56,27:POKE55,255
1 POKE36878,15:DEF FN SC(A)=PEEK(7680
+Y*22+A)
2 GOSUB400
5 GOTO280
7 L=2
10 X1$=CHR$(10)+CHR$(11)
20 Y1$=CHR$(10)+CHR$(10)
22 Y$="
25 X$="
30 POKE36879,120:PRINT"
35 FORI=128TO250
40 POKE36876,I
50 NEXTI:POKE36876,0
60 MX=INT(RND(1)*22)
70 MY=INT(RND(1)*22)
80 Y=MY:IFFN SC(MX)<>32THEN60
90 PRINT LEFT$(Y$,MY+1);LEFT$(X$,MX);
"
100 POKE36876,150:POKE36876,0
110 FORI=1TO40
120 OX=ASC(X1$)
125 GETK$
130 OX=OX+(K$="A")-(K$="D")
140 OY=ASC(Y1$)
150 OY=OY+(K$="W")-(K$="X")
160 IFOX=ASC(X1$)ANDOY=ASC(Y1$)THENOX
=OX+OX-ASC(MID$(X1$,2)):OY=OY+OY-ASC(
MID$(Y1$,2))
165 IFOX<00ROX>21ROY<0ROY>21THEN240
170 Y=OY:IF FN SC(OX)=156 THENL=L+1
180 IF FN SC(OX)=164 OR FN SC(OX)=0 T
```

```
HEN260
182 PRINT LEFT$(Y$,OY+1);LEFT$(X$,OX)
;"■■"
185 PRINT LEFT$(Y$,ASC(MID$(Y1$,L))+1
);LEFT$(X$,ASC(MID$(X1$,L)));" "
190 X1$=CHR$(OX)+LEFT$(X1$,L)
200 Y1$=CHR$(OY)+LEFT$(Y1$,L)
210 NEXTI
220 Y=MY:IF FN SC(MX)=156 THENPRINTLE
FT$(Y$,MY+1);LEFT$(X$,MX);"■■$"
230 GOTO60
240 PRINTLEFT$(Y$,1);"■■YOU CRASHED I
NTO THE  SURROUNDING WALL------";
250 GOTO270
260 PRINTLEFT$(Y$,1);"■■YOU HIT SOMET
HING YOU SHOULD NOT HAVE~"
270 PRINT"■AFTER GROWING TO";L
275 POKE36876,128:FORI=1TO500:NEXTI
277 POKE36876,0:FORI=1TO2000:NEXTI
280 PRINT"■■@@@@@@@@@@■SNAKE■@@@@@@@@"
290 PRINT"■YOU ARE DIRECTING A  MONE
Y SNAKE AROUND THESCREEN USING THE KE
YS"
292 PRINT"■A,D,W&X. YOUR AIM IS  TO M
AKE THE SNAKE GROWBY GUIDING IT TO TH
E"
295 PRINT"■E SIGNS, ON WHICH IT  FEED
S."
300 PRINT"■■EACH £ REMAINS ON THE SC
REEN FOR A SHORT  PERIOD BEFORE"
302 PRINT"■CONVERTING INTO A $  WHIC
H KILLS IF EATEN. THE SNAKE ALSO DIES
 IF";
305 PRINT"■IT BITES ITSELF."
307 PRINT"■■PRESS 'Y' TO START THEGAM
```

```
E OR 'N' TO STOP"
310 GETA$
320 IFA$="Y"THEN7
330 IFA$<>"N"THEN310
335 POKE36869,240:PRINT"[illegible]"
340 STOP
400 REM CHARACTER DATA
410 FORI=7424TO7431:POKEI,0:NEXTI
420 FORI=7168TO7175
430 READN:POKEI,N
440 NEXTI:POKE36869,255:RETURN
450 DATA231,165,255,36,36,255,165,231
```

STARS AND STRIPES

Despite what you first thought when you saw the title, this program does not draw an American Flag. Instead, it plays an elaborate, and most colourful, version of Noughts and Crosses. Once you've played a few rounds against the program as it is, you can modify it to play against itself, which is fun to watch. To make it an 'Auto-Stars and Stripes', change line 80 so it reads as follows:

```
80 B$=RIGHT$(STR$(INT(RND(1)*9+1)),1)
```

Here is the main listing, in which you play against the computer:

```
5 POKE36878,15
10 REM STARS AND STRIPES
12 POKE36879,30:PRINT"[illegible]";
15 PRINT"[illegible]              "
17 PRINT"[illegible]
"
20 DIMA$(9),C(9):C(2)=1
30 RESTORE:G=0:FORB=1TO9:POKE36876,13
0+B*2:READA$(B):IFC(B)=0THENG=G+1
```

```
32 POKE36876,0:NEXTB
33 IFG=0THENE=2:GOTO115
35 GOTO 76
40 PRINT"[■■■■■■■■■■■■■■]";:FORD=1TO9
45 IF C(D)=0THENPRINT"[■]";D;
50 IF C(D)=1THENPRINT"[■] X ";
55 IF C(D)=2THENPRINT"[■] O ";
60 IF D=3 OR D=6 THENPRINT:PRINTTAB(7
);"          ":PRINTTAB(7);
70 NEXTD:PRINT
75 RETURN
76 GOSUB40
80 GETB$:IFB$<"1"ORB$>"9"THEN80
90 E=VAL(B$):IFC(E)<>0THEN80
100 C(E)=2
110 IFLEN(A$(E))=0THEN117
115 F=ASC(A$(E))-80:GOTO120
117 PRINT"[■■■]";TAB(7);"IT'S A DRAW":F
ORG=150TO200:POKE36876,G:NEXT:POKE368
76,0:RUN
120 IFC(F)=0THEN140
130 A$(E)=MID$(A$(E),2):GOTO110
140 C(F)=1
145 GOSUB40
150 IFC(1)=C(2)ANDC(2)=C(3)ANDC(1)<>0
THEN ON C(1)GOTO510,520
160 IFC(4)=C(5)ANDC(5)=C(6)ANDC(4)<>0
THEN ON C(4)GOTO510,520
170 IFC(7)=C(8)ANDC(8)=C(9)ANDC(7)<>0
THEN ON C(7)GOTO510,520
180 IFC(1)=C(5)ANDC(5)=C(9)ANDC(1)<>0
THEN ON C(1)GOTO510,520
190 IFC(3)=C(5)ANDC(5)=C(7)ANDC(3)<>0
THEN ON C(3)GOTO510,520
200 IFC(1)=C(4)ANDC(4)=C(7)ANDC(1)<>0
```

```
THEN ON C(1)GOTO510,520
210 IFC(2)=C(5)ANDC(5)=C(8)ANDC(2)<>0
THEN ON C(2)GOTO510,520
220 IFC(3)=C(6)ANDC(6)=C(9)ANDC(3)<>0
THEN ON C(3)GOTO510,520
230 GOTO 30
510 PRINT"■■■■";TAB(5);"I WIN!!":FORG
=128TO208:POKE36876,G:NEXTG:POKE36876
,0:RUN
520 PRINT"■■■■YOU WIN!!":FORG=128TO20
8:POKE36876,G:NEXTG:POKE36876,0:RUN
530 DATAUXWSVTY,P,UWYQTVX,QSUVYX,QSWT
VYX,SQUWX,QSUXY,SQWUYVT,SQUXW
```

CHECKERS

Challenge your computer to this traditional board game of skill. In this program, written by Graham Charlton, you enter your moves as the letter across the top followed by the letter down the side of the square you want to move from, then after pressing RETURN the letters of the square you're moving to. the entry must be in the form of "AB" or "FE".

```
1 GOSUB9000
50 MO=0
60 GOSUB7000
1000 PRINT"■■■■■■■■■■■■■■■■"
1005 PRINT"THIS MOVE":MO=0:INPUTFH$
1010 PRINT"■TO":INPUTTH$
1050 PRINT"■STAND BY"
1060 X$=LEFT$(FH$,1):Y$=RIGHT$(FH$,1)
:FX=ASC(X$):FY=ASC(Y$)
1065 X$=LEFT$(TH$,1):Y$=RIGHT$(TH$,1)
:TX=ASC(X$):TY=ASC(Y$)
1070 FX=FX-64:FY=FY-64:TX=TX-64:TY=TY
-64
```

```
1420 MO=0
1430 A%(TY,TX)=A%(FY,FX):A%(FY,FX)=B
1450 IFABS(TY-FY)>1THENMO=1:A%(FY+((T
Y-FY)/2),FX+((TX-FX)/2))=B:T=T+1
1470 GOSUB7000
2000 Y=8
2006 X=8
2020 IF A%(Y,X)<>C AND A%(Y,X)<>KTHEN
2100
2030 IFA%(Y,X)=C AND Y=8THENA%(Y,X)=K
2040 FORD=0TO3
2041 Q=-1
2045 IFX+2*X(D)<1 OR X+2*X(D)>8THEN20
90
2046 IFY+2*Y(D)<1 OR Y+2*Y(D)>8THEN20
90
2050 IF(A%(Y+Y(D),X+X(D))=H OR A%(Y+Y
(D),X+X(D))=W)ANDA%(Y+2*Y(D),X+2*X(D)
)=BTHENQ=D
2070 IFA%(Y,X)<>K ANDD>1THEN2100
2080 IFQ>-1ANDQ<4THEN2125
2090 NEXTD
2100 IFA%(Y,X)=HANDY=1THENA%(Y,X)=W
2110 X=X-1:IFX>0THEN2020
2115 Y=Y-1:IFY>0THEN2006
2120 IFQ=-1THEN2350
2125 PX=X+2*X(Q):PY=Y+2*Y(Q)
2130 S=S+1
2135 A%(Y+Y(Q),X+X(Q))=B
2140 A%(PY,PX)=A%(Y,X)
2145 A%(Y,X)=B
2150 GOSUB7000
2155 M=-1
2160 FORD=0TO3
2165 IFPX+2*X(D)<1 OR PX+2*X(D)>8THEN
```

```
2200
2166 IFPY+2*Y(D)<1 OR PY+2*Y(D)>8THEN
2200
2170 IFA%(PY+Y(D),PX+X(D))<>HAND A%(P
Y+Y(D),PX+X(D))<>WTHEN2180
2175 IFA%(PY+Y(D)*2,PX+2*X(D))=BTHENM
=D
2180 IFA%(PY,PX)<>KANDD>1THEN2210
2190 IFM>-1THEN2210
2200 NEXTD
2210 IFM=-1THEN50
2220 A%(PY+Y(M),PX+X(M))=B
2222 A%(PY+2*Y(M),PX+2*X(M))=A%(PY,PX
)
2224 A%(PY,PX)=B
2226 S=S+1
2230 GOTO50
2350 Y=0
2360 PY=INT(RND(1)*8+1):PX=INT(RND(1)
*8+1)
2370 Y=Y+1
2371 Q=-1
2375 IFY>400THEN2440
2380 IFA%(PY,PX)<>CANDA%(PY,PX)<>KTHE
N2360
2390 FORD=0TO3
2395 Q=-1
2396 IFPY+Y(D)<1ORPY+Y(D)>8THEN2425
2397 IFPX+X(D)<1ORPX+X(D)>8THEN2425
2400 IFA%(PY,PX)=CANDD>1THEN2425
2410 IFA%(PY+Y(D),PX+X(D))=BTHENQ=D
2420 IFQ>-1ANDQ<4THEN2460
2425 NEXTD
2430 IFY<401THEN2360
2440 F$="L"
```

```
2450 GOTO7000
2460 A%(PY+Y(Q),PX+X(Q))=A%(PY,PX)
2470 A%(PY,PX)=B
2490 GOTO50
7000 PRINT"■■ ABCDEFGH"
7005 FORY=1TO8:POKEC0+Y*22,1:POKESC+Y
*22,Y
7010 FORX=1TO8:CL=0:IFA%(Y,X)=160THEN
CL=6
7015 IFA%(Y,X)=C OR A%(Y,X)=KTHENCL=5
7020 IFA%(Y,X)=H OR A%(Y,X)=WTHENCL=1
7025 POKEC0+Y*22+X,CL:POKESC+Y*22+X,A
%(Y,X)
7030 NEXTX,Y
7035 PRINT"■■■■■■■■■■■■■■COMPUTER"S"■■
          "
7037 PRINT"■■HUMAN"T
7040 PRINT"

  "
7045 PRINT"
                    "
7050 PRINT"

  "
7210 IFF$="L"THENPRINT:PRINT"■■I CONCE
DE THE GAME":STOP
7230 IFS=12THENPRINT:PRINT"■■■■■■■I W
IN":STOP
7240 IFT=12THENPRINT:PRINT"■■■■■■■YOU
 WIN":STOP
7260 U$="":IFMO<>1THEN7265
7262 PRINT"■■■■■■■■■■■■■■CAN YOU JUMP AG
AIN              ":INPUTU$
7265 MO=0
```

```
7310 IFLEFT$(U$,1)="Y"THENFH$=CHR$(TX
+64)+CHR$(TY+64):GOTO1010
7320 RETURN
7900 STOP
9000 DIMA%(8,8)
9005 Y(0)=1:X(0)=-1:Y(1)=1:X(1)=1
9010 Y(2)=-1:X(2)=-2:Y(3)=-1:X(3)=1
9050 H=209:C=215:W=139:K=151
9060 B=160:Q=-1:F$="":S=0:T=0
9061 CO=38400:SC=7680
9065 FORY=1TO8:IF2*INT(Y/2)=YTHENFORX
=1TO7STEP2:GOTO9075
9070 FORX=2TO8STEP2
9075 A%(Y,X)=160
9080 IFINT(X/2)*2=XTHENA%(Y,X-1)=32
9085 IFINT(X/2)*2<>XTHENA%(Y,X+1)=32
9090 NEXTX,Y
9115 FORY=1TO3
9120 IF2*INT(Y/2)=YTHENFORX=1TO7STEP2
:GOTO9130
9125 FORX=2TO8STEP2
9130 A%(Y,X)=C
9135 NEXTX,Y
9140 FORY=6TO8
9145 IF2*INT(Y/2)=YTHENFORX=1TO7STEP2
:GOTO9155
9150 FORX=2TO8STEP2
9155 A%(Y,X)=H
9160 NEXTX,Y
9165 FH$="":TH$=""
9300 POKE36879,8:PRINT"■■DRAUGHTS":PR
INT:PRINT"■DO YOU WANT FIRST MOVE"
9310 INPUTY$:PRINT"■":IFLEFT$(Y$,1)="
Y"THENRETURN
9340 GOSUB7000
```

```
9350 A=INT(RND(1)*3+1)*2:Q=INT(RND(1)
*2)
9370 A%(3+Y(Q),A+X(Q))=C:A%(3,A)=160
9390 RETURN
```

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.

You copy the computer's sequence by using the keys numbered 1 to 4. There is a high score feature.

```
10 H=0
20 POKE36879,8
30 POKE36878,10
50 PRINT"■■"
60 DATA2,5,3,7
70 J=RND(-TI)
80 RESTORE
90 PRINT"■"
100 A$=""
110 S=0
120 D=100
130 M$="###"
200 PRINTM$;TAB(13);"■SIMON"
210 PRINT"■■■■SCORE";S:PRINT"■■■■■■
HI SCORE";H
220 PRINT"■■----------"
```

```
240 FORI=1TOD*3:NEXT
250 A$=A$+CHR$(INT(RND(1)*4)+1)
255 FORT=1TOLEN(A$)
260 FORA=1TOASC(MID$(A$,T))
270 READB
280 NEXTA
285 RESTORE
286 POKE646,B
290 PRINT"
      "
300 PRINT"";TAB((ASC(
MID$(A$,T))-1)*5);
305 PRINTASC(MID$(A$,T))
310 POKE36876,183+(ASC(MID$(A$,T))-1)
*12:POKE36875,201:FORI=1TOD*8:NEXT
320 POKE36876,0:POKE36875,0
330 PRINT"
      "
335 PRINT"";TAB((ASC(
MID$(A$,T))-1)*5);"    "
340 NEXTT
350 RESTORE
360 FORA=1TO4
370 PRINT"";TAB((A-1
)*5);A
380 READB
386 POKE646,B
390 PRINT"";TAB((A-1
)*5);"     "
400 NEXTA
410 RESTORE
415 FORT=1TOLEN(A$)
420 FORA=1TOD*3
430 GETI$
440 IFI$>"0"ANDI$<"5"THEN550
```

```
450 NEXTA
460 M$=MID$(M$,2)
465 FORX=1TO2:FORI=1TO12
470 POKE36879,8+I*16+INT(RND(1)*8)+1:
POKE36878,10+I:POKE36877,I*INT(RND(1)
*8)+1
480 POKE36876,200-I*10:POKE36875,150+
INT(NRD(1)*8)+1*5:POKE36874,130+INT(R
ND(1)*100)
490 NEXTI,X
500 POKE36874,0:POKE36875,0:POKE36876
,0:POKE36877,0:POKE36878,10
510 FORI=245TO150STEP-.5:POKE36877,I:
NEXT:FORI=120TO200:POKE36877,I:NEXT
515 POKE36877,0
520 POKE36879,8
530 PRINT"J"
535 IFM$<>""THEN255
540 GOTO640
550 IFVAL(I$)<>ASC(MID$(A$,T))THEN460
560 POKE36876,183+(VAL(I$)-1)*12:POKE
36875,201
562 FORI=1TO500:NEXT:POKE36875,0:POKE
36876,0
565 GETI$:IFI$<>""THEN565
570 NEXTT
580 PRINT"J"
590 S=S+1
600 D=D-S/5
610 IFD<0THEND=0
620 IFH<STHENH=S
630 GOTO200
640 FORA=1TO20
650 POKE36875,120+A
660 POKE36876,240-A*5
```

```
670 FORI=1TO100:NEXT:POKE36875,0:POKE
36876,0
675 NEXTA
680 PRINT"■■■■■■■SCORE";S
690 PRINT"■■■HI SCORE";H
700 PRINT"■■■■■■■■■■PRESS ANY KEY TO P
LAY    AGAIN."
710 GETA$:IFA$=""THEN710
720 GOTO80
```

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.

```
5 CO$="■■■■■■■■":POKE36878,15
10 REM ECOLOGICAL DISASTER
15 RN=RND(1):BW=0:POKE36879,24:PRINTC
HR$(147);
20 PRINTTAB(2);"■YOU ARE THE
 PRESIDENT'S"
30 PRINT" ADVISOR ON NATURAL    RESOU
RCES,AND PART OF YOUR JOB IS TO STOCK
"
40 PRINT" A NEW LAKE IN         MICHI
GAN WITH FISH    AND EELS...."
50 PRINT" THE ONLY CATCH IS     THAT
THE FISH AND     EELS ARE ENEMIES....
"
60 PRINT"  YOU MUST SELECT      START
ING NUMBERS OF   FISH AND EELS WHICH"
70 PRINT" WILL ENSURE THE       LONGE
```

```
ST POSSIBLE      SURVIVAL OF THE LAKE
."
150 PRINT"■■  PRESS A KEY WHEN
 YOU'RE READY      "
160 GETA$:IFA$=""THEN160
165 PRINT"■"
170 PRINT"ENTER THE STARTING    NUMBE
R OF EELS        (LESS THAN 100)."
175 INPUTEL
180 IFEL>100OREL<1THEN175
190 PRINT"ENTER THE STARTING    NUMBE
R OF FISH        (LESS THAN 100)."
195 INPUTFI
200 IFFI>100ORFI<1THEN195
205 FI=FI/3:WE=0
210 FORG=130TO230
220 PRINTMID$(CO$,RND(1)*6+1,1);
225 PRINT"■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■THE L
AKE IS EVOLVING"
230 POKE36876,G:NEXTG:POKE36876,0
235 WE=WE+1:PRINT"■";
240 POKE36879,40:PRINT"■REPORT TO PRE
SIDENT ATTHE END OF WEEK";WE
250 EL=EL+((8*EL-EL*FI/3)*RN)
260 FI=FI+((4*FI-FI*EL)*.01)
270 PRINT"■■■■■■";TAB(5);MID$(CO$,RND(
1)*6+1,1);-INT(FI)*(FI>0);
275 PRINT"■■ FISH"
280 PRINT"■■■■■■";TAB(5);MID$(CO$,RND(
1)*6+1,1);-INT(EL)*(EL>0);
285 PRINT"■■ EELS"
290 IFEL<2 OR FI<2THENPRINT"■";:GOTO3
10
300 GOTO210
310 IFWE>BWTHENBW=WE
```

```
315 FORG=1TO40
320 PRINTMID$(CO$,RND(1)*6+1,1);"ECOL
OGICAL    DISASTER";
330 POKE36879,40+RND(1)*8:POKE36876,2
00-G
340 NEXTG:POKE36876,0
360 POKE36879,25:PRINT"■■";
380 PRINT"■■■■■■  IT'S ALL OVER NOW,"
390 PRINT" NATURAL RESOURCES     ADVI
SOR. SIR!"
400 PRINT"■■■  YOUR LAKE STAYED"
405 PRINT" ALIVE FOR";WE;"WEEKS"
410 PRINT" SO YOUR BEST TO DATE"
415 PRINT" IS";BW;"WEEKS..."
420 FORG=130TO230STEP.1
425 POKE36876,G
430 NEXTG:POKE36876,0
440 PRINT"■";:GOTO20
```

CASUALTY

In this program written by Paul Toland, you are on a minefield which is full of casualties. You have to push a wheelchair around the minefield, avoiding the mines and electrified fence, to collect each casualty and bring him or her to the hospital (shown as a +). The wheelchair can only carry one casualty at a time.

```
5 POKE56,27:POKE55,255:POKE36878,15
10 Y$="■■■■■■■■■■■■■■■■■■■■■■■■■"
15 X$="■■■■■■■■■■■■■■■■■■■■■■■■!"
17 DEF FN SC(A)=PEEK(7680+Y*22+A)
20 GOSUB400
22 GOTO610
30 POKE36879,127:PRINT"■■DDDDDDDDDDD
```

```
DDDDDDDDDD"
35 FORI=7702TO8142STEP22
40 POKEI+30720,2:POKEI+30741,2
45 POKEI,5:POKEI+21,5:NEXTI
50 FORI=8164TO8185:POKEI+30720,2
55 POKEI,4:NEXTI
60 PRINT"■";
90 FORI=1 TO 20
100 PRINT LEFT$(Y$,RND(1)*17+3);LEFT$
(X$,RND(1)*18+2);"C"
110 NEXTI
115 PRINT"■";
120 FORI=1TO20
130 X=INT(RND(1)*18)+2
140 Y=INT(RND(1)*17)+2
150 IF FN SC(X)<>32THEN130
160 PRINTLEFT$(Y$,Y+1);LEFT$(X$,X);"B
"
170 NEXTI
190 PRINT"■";
200 CAR=0
210 RET=0
220 X=11:Y=1
222 A=0:D=0
230 PRINTLEFT$(Y$,Y+1);LEFT$(X$,X);"
"
235 PRINTLEFT$(Y$,2);LEFT$(X$,11);"■■
+"
237 GETI$
240 IFI$="A" THENA=-1:D=0
245 IFI$="D" THENA=1:D=0
250 IFI$="X" THENA=0:D=1
255 IFI$="W" THENA=0:D=-1
257 X=X+A:Y=Y+D
260 CH=FN SC(X)
```

```
265 PRINTLEFT$(Y$,Y+1);LEFT$(X$,X);CH
R$(64+CAR)
270 IFCH=3 THEN500
280 IFCH=4 ORCH=5 THEN520
290 IFCH=2 ANDCAR THEN540
300 IFCH=2 THENCAR=1
310 IFCH=171ANDCAR THENCAR=0:POKE3687
6,200:RE=RET+1:POKE36876,0:IFRET=20TH
EN560
320 FORI=1TO DELAY:NEXTI
330 GOTO230
400 FORI=7424TO7431:POKEI,0:NEXTI
410 FORI=7168TO7215
420 READN:POKEI,N
430 NEXTI
440 POKE36869,255:RETURN
450 DATA192,192,144,240,144,144,158,2
04
460 DATA192,192,156,252,152,159,159,2
05
470 DATA0,0,0,0,0,209,209,255
480 DATA0,0,0,60,82,255,60,0
490 DATA65,65,162,162,20,20,8,8
495 DATA12,3,12,48,192,48,12,3
500 PRINTLEFT$(Y$,Y+1);LEFT$(X$,X);"
C":POKE36876,128
510 PRINTLEFT$(Y$,2);"YOU HIT A MIN
E":POKE36876,0:GOTO600
520 PRINTLEFT$(Y$,2);"ZZZZZZZZAAA
AAPPP"
525 FORI=128TO170STEP3
530 POKE36876,I:POKE36875,I+10
535 NEXTI:POKE36876,0:POKE36875,0:GOT
O600
540 PRINTLEFT$(Y$,2);"THE WHEEL C
```

```
HAIR IS"
545 PRINT"■■■■■■OVER-LOADED":GOTO600
560 POKE36876,200:POKE36875,250
570 PRINT:PRINT"■■YOU DID IT!!!!"
580 POKE36876,0:POKE36875,0
600 PRINT"■■YOU RESCUED";RET
605 PRINT"■ OUT OF THE 20"
607 FORI=1TO4000:NEXTI
610 PRINT"■■DDDDDD■MINE-FIELD■DDDDDD"
620 PRINT"■■ THERE ARE 20          CAS
UALTIES LYING IN   THE MINEFIELD.IT I
S"
622 PRINT"■YOUR JOB TO BRING THEMTO T
HE HOSPITAL IN A  WHEELCHAIR ONE AT A
"
625 PRINT"■TIME. THE FENCE AROUNDTHE
FIELD IS          ELECTRIFIED SO AVOI
D"
627 PRINT"■CONTACT WITH IT."
630 PRINT"■■■CHOOSE A SKILL LEVEL  0
TO 5 (5 EASY)"
635 INPUT"■";DELAY
640 IF DELAY<0 ORDELAY>5 THEN635
650 DELAY=DELAY*15+4
660 GOTO30
```

SLALOM

In this game you control a skier who is skiing down a slope. Turn your man left with the Z key, right with the M key. Decrease his speed with the C key and increase it with the B key. Watch out for the trees and ice, and try to navigate between the posts.

```
1 PRINT"[illegible]":POKE36879,30:FORI=7680TO81
85:POKEI,160:NEXT:POKE36878,8
2 POKE55,96:POKE56,29:FORI=7520TO7679
:READR:POKEI,R:NEXT:POKE36869,255
3 P=11:SC=7680:CS=38400:CO=30720:S=0:
L=200+(INT(RND(1)*50)):D=2:SP=2
10 IFL=0THEN92
11 IFRND(1)<.25THEN100
12 IFRND(1)<.05THEN70
13 IFRND(1)<.35THEN30
15 FORI=1TOINT(RND(1)*4)+1
20 TR=8164+INT(RND(1)*22):POKETR+CO,5
:POKETR,49:NEXT
25 GOTO100
30 CR=8164+INT(RND(1)*17)
40 FORI=1TOINT(RND(1)*4)+1
50 POKECR+CO+I,3:POKECR+I,48
60 NEXT
65 GOTO100
70 PO=8164+INT(RND(1)*14)
75 POKEPO+CO,4:POKEPO,50:FORI=1TOINT(
RND(1)*4)+2
80 POKEPO+I,224:NEXT
90 POKEPO+I+CO,4:POKEPO+I,50
91 GOTO100
92 POKE8172+CO,0:POKE8176+CO,0:POKE81
51+CO,6:POKE8152+CO,6:POKE8153+CO,6
95 POKE8172,231:POKE8176,229:POKE8151
,58:POKE8152,59:POKE8153,60
100 POKE7702+P,160:POKE7724+P,160
105 PRINT"[illegible]":S=
S+1:L=L-1
110 PRINT"[illegible]345[illegible]";S
120 Z=PEEK(197)
```

```
130 U=P
135 IFZ=33THEND=D-1:IFD<1THEND=1
136 IFZ=36THEND=D+1:IFD>3THEND=3
137 IFZ=33ORZ=36THENPOKE36877,240
140 U=U-(D=1)*(U>0)+(D=3)*(U<21)
150 SP=SP-(Z=34)*(SP>1)+(Z=35)*(SP<5)
151 IFZ=34ORZ=35THENPOKE36874,200
160 IFPEEK(7724+U)=49ORPEEK(7724+U)=5
0THENP=U:GOTO1000
170 IFPEEK(7724+U)=224THENS=(S*2)+(10
*SP):FORX=200TO245:POKE36876,X:NEXT
180 IFPEEK(7724+U)=48THENSP=INT(RND(1
)*5)+1:S=INT(S/2):POKE36875,235
182 P=U
185 IFPEEK(7768+P)>57ANDPEEK(7768+P)<
61THEN2000
190 POKE7702+P+CO,2:POKE7724+P+CO,2:P
OKE7702+P,44:POKE7724+P,44+D
195 IFL=-40THEN3000
200 FORI=1TO100-(SP*20):NEXT:POKE3687
4,0:POKE36875,0:POKE36876,0:POKE36877
,0:GOTO10
251 ,31,251,136,48,0
255 -6
1000 POKE36874,0:POKE36875,0:POKE3687
6,0:POKE36877,0
1001 POKE7702+P,160:POKE7722+P,160:IF
P>15THEN1110
1010 FORI=7746+PTO7746+P+INT(RND(1)*5
)+1
1020 D=D+1:IFD>4THEND=1
1030 POKEI+CO,2:POKEI,53+D
1035 POKE36877,230+(I-7746)
1040 FORJ=1TO180:NEXT
1050 POKEI,160:NEXT:P=P+5
```

```
1110 POKE36877,0:FORI=7746+PT08164+PS
TEP22
1120 D=D+1:IFD>4THEND=1
1130 POKEI+CO,2:POKEI,53+D
1132 READNO
1135 POKE36875,NO
1140 READPA:FORJ=1TOPA:NEXT:POKE36875
,0:FORJ=1TO22:NEXT
1150 POKEI,160:NEXT
1155 POKE36877,140:FORI=1TO500:NEXT
1160 POKE36877,0:FORI=1TO3000:NEXT:PO
KE36869,240:PRINT"□":POKE36879,8
1200 PRINT"■BAD LUCK;YOU CRASHED"
1300 PRINT"■**********************"
1301 PRINT"▙SCORE:"S
1305 PRINT"▙DO YOU"
1310 PRINT"WANT ANOTHER GO■?":POKE198
,0
1320 GETA$
1330 IFA$="Y"THENRUN
1340 IFA$<>"N"THEN1320
1350 END
2000 POKE36874,0:POKE36875,0:POKE3687
6,0:POKE36877,0
2005 POKE36878,1:POKE36877,241:FORI=1
TO10
2010 FORJ=0TO2
2015 POKE7724+P-(J*22)+CO,2
2020 POKE7724+P-(J*22),61+J:FORX=1TO3
00:NEXT:POKE7724+P-(J*22),160
2030 NEXT
2040 FORJ=0TO2
2045 POKE7680+P+(J*22)+CO,2
2050 POKE7680+P+(J*22),63-J:FORX=1TO1
00:NEXT:POKE7680+P+(J*22),160
```

```
2060 NEXTJ,I
2070 POKE36877,0
2200 POKE36869,240:PRINT"□":POKE36879
,110
2210 PRINT"■WELL DONE!"
2240 GOTO1300
3000 POKE36869,240:PRINT"□":POKE36879
,216
3010 PRINT"■■YOU MISSED THE FINISH PO
ST!!"
3020 GOTO1300
7762 -7746
9000 DATA56,56,56,16,124,186,186,186,
186,40,40,42,44,72,144,32
9010 DATA186,40,40,40,40,68,0,0,186,4
0,40,168,104,36,18,8
9020 DATA0,0,0,4,18,173,102,193,8,28,
28,62,62,127,127,8
9030 DATA16,24,28,30,16,16,16,16
9040 DATA0,0,238,136,232,40,238,0,0,0
,238,170,174,172,234,0
9050 DATA0,0,244,128,224,128,244,0
9060 DATA56,56,16,124,186,170,40,108,
48,136,251,31,251,136,48,0
9070 DATA108,40,170,186,124,16,56,56,
0,12,17,223,248,223,17,12
9080 DATA255,194,222,222,194,222,222,
255,255,138,170,170,170,171,170,255
9090 DATA255,43,235,235,35,171,43,255
9100 DATA56,56,16,124,186,170,40,108,
56,56,16,60,219,24,36,102
9110 DATA0,186,186,146,124,56,40,198
9500 DATA195,600,0,60,195,450,0,60,19
5,150,195,600,0,60,203,450,201,150
9510 DATA0,60,201,450,195,150,0,60,19
```

```
5,450,0,60,195,150,0,60,195,750,0,250
,0,250
```

OPERATION DESPERATION

Your planes engine has stopped as you are flying over a large city. Your altitude is decreasing rapidly, and your only hope of landing intact is to flatten the city with your bombs. Due to a fault in your missile tracking system, only one bomb may be in the air at any one time.

At the start of the program, line 2Ø sends action to the subroutine at line 35Ø, where the user-defined graphics are constructed "@" is the unit of 'wall of window' used to build up the city. "C" is the bomb, "A" is the roof of the buildings, "B" is the ground, and "D" is the plane.

Once the program is working satisfactorily, the graphics can be created. Working in this way (instead of defining the characters before you start on the mechanics of the program proper) ensure you do not get so engrossed in making pretty graphics that you forget to make the program worthwhile in its own right.

On returning from that subroutine, a routine (from 28Ø) is called by line 3Ø. This prints up the instructions:

> THE ENGINE OF YOUR PLANE HAS STALLED WHILST YOU ARE OVER A LARGE CITY. YOUR ONLY HOPE OF LANDING SAFELY IS TO FLATTEN THE CITY WITH YOUR BOMBS. PRESS 'F' TO RELEASE A BOMB; ONLY ONE BOMB MAY BE FALLING AT ANY ONE TIME.
>
> PRESS 'Y' TO START, 'N' TO STOP.

After the instructions you'll see one way of using GET to wait until a specific key is pressed (other ways of doing these are demonstrated in other programs in this book). The string

variable A$ is set equal to GET. Line 31Ø checks to see if A$ equals "Y" and if it does, the program goes to line 4Ø, to begin in earnest. If A$ does not equal "Y", the programm 'falls through' to the next line, where a check is made to see if A$ equals "N". If it does, the program terminates in line 32Ø. If it does not, line 33Ø is reached, which sends the computer back to 3ØØ to read the keyboard again (that is, to get another value of GET). This cycle continues until either "Y" or "N" is detected.

Back at line 4Ø, the background and border colours are set to white (colour one), and the screen is cleared (end of line 4Ø). Our city is constructed by the double loop (I and J) from lines 5Ø to 1ØØ. Note that a random colour (black, blue or red) is selected by line 6Ø. As this is outside the J loop, everything printed within a particular run of the J loop will be in one of the three colours, although buildings on either side (each complete J loop constructs one building) may well be in other colours. As you'll see when you run this, the effect is most impressive.

The variables X and Y are both set equal to zero. The variable BY is set to −1 in line 115. BY is the 'bomb drop' flag. (A flag is a variable within a program which is used to signal the presence or otherwise of a certain condition). In this case, the flag BY equals −1 when no bomb has been dropped, and changes to +1 when a bomb has been dropped. BX is the X coordinate of the bomb when it is falling, and BX is set equal to zero in line 115.

Line 12Ø prints a white blank where the plane has just been. By printing over the old plan position with a blank, and very shortly afterwards (line 16Ø) printing a new plane, the illusion of a moving plane is created. In essence, as you saw earlier in the book, this is how most moving graphics programs work. An object is printed in one position, held there for a moment, erased, then reprinted in a different position, creating the impression that a single object has moved from one point to the next.

Line 130 adds one to the value of X, which determines how far across the screen the plane will be printed. If X equals 22, then the plane has reached the side, so one is added to the value of Y, to move it one line down the screen, and X is reset to one to make the plane re-appear on the left hand side of the screen. If Y equals 22 (the end of line 140 checks for this) then the computer knows the plane has managed to reach the ground safely, so attention goes to line 240, where the "well done, you've made it" message is printed.

The contents of the square into which the plane will be printed is checked. If it is 32 then all is well. There is just empty sky ahead. Any other value (that is, not a space — code 32) then the computer knows the plane is about to crash, and the program goes to line 260 where the "you blew it" message is printed. The plane is printed, by line 160, in its new position.

Line 170 looks to the keyboard, to see if "F" for fire is being pressed. If it is, and the flag BY has the value −1 (remember, BY changed to +1 when a bomb is in the air) then the starting position of the bomb is set equal to the current position of the plane, to ensure that the bomb will be seen to fall from the plane. Line 180 checks the value of BY again, and if it is −1, then goes back to line 120 to repeat the 'print the blank, check for a crash, print the plane' routine.

If the bomb has been dropped, then the test on BY in line 180 proves negative, so the GO TO 120 at the end of line 180 is ignored. Line 190 'unprints' the bomb, 200 increments its downward position, and line 205 checks to see if the bomb has hit the ground (BY equals 22) and if it has, makes a noise, and returns to 120 to move the plane. PEEK is used again; this time to check if the bomb has hit the top of a building. If it has, the top of the building is erased, the bomb flag is set back to −1, there is a tone, and the program returns to 120 to move the plane along. If the bomb has not hit something (so the PEEK finds a 32), the bomb is reprinted in its new position. Line 225

sends action back to 120 to move the plane. Notice that so long as BY equals +1, the 'print the bomb' cycle is traversed, until the bomb hits the ground or the top of a building.

Congratulations are held by line 240: "CONGRATULATIONS — YOU MADE IT". After a tone, the program goes to line 280 where the instructions and option to play is displayed again. Line 260 holds the grim news of failure: "YOU CRASHED AND ARE NOW PART OF THE CITY SKYLINE". A mournful tone is heard and the instruction/game option appears.

You may wish to add a delay after lines 240 and 260 so that there will be a short pause between the end of one game, and the possibility of a new game.

```
5 POKE56,27:POKE55,255
10 LETSC=7680:LETCS=38400
20 GOSUB350
30 PRINT"□":GOTO280
40 POKE36879,25:POKE36869,255:PRINT"□
"
45 FORI=484TO505:POKECS+I,5:POKESC+I,
2:NEXT
50 FORI=1TO20
60 IFRND(1)<.5THENCO=0:GOTO70
64 IFRND(1)<.5THENCO=2:GOTO70
66 IFRND(1)<.5THENCO=6:GOTO70
70 FORJ=21TO12+RND(1)*10STEP-1
80 POKESC+J*22+I,0:POKECS+J*22+I,CO
90 NEXTJ:POKESC+J*22+I,1:POKECS+J*22+
I,CO
100 NEXTI
110 X=0:Y=0
115 BY=-1:BX=0
120 FORI=1TO100:NEXTI:POKECS+Y*22+X,1
```

```
130 X=X+1
140 IFX=22THENY=Y+1:X=0:IFY=23THEN240
150 IFPEEK(SC+Y*22+X)<>32THEN260
160 POKESC+Y*22+X,4:POKECS+Y*22+X,0
170 GETA$:IFA$="F"ANDBY=-1THENBY=Y:BX
=X-1
180 IFBY=-1THEN120
190 POKECS+BY*22+BX,1:POKESC+BY*22+BX
,32
200 BY=BY+1
205 IFBY=22THENBY=-1:GOTO120
210 IFPEEK(SC+BY*22+BX)<>32THENGOTO23
0
220 POKESC+BY*22+BX,3:POKECS+BY*22+BX
,2
225 GOTO120
230 POKECS+BY*22+BX,2:POKESC+BY*22+BX
,32
235 FORI=1TO150:NEXT:POKECS+BY*22+BX,
1
236 BY=-1:GOTO120
240 POKESC-1+Y*22+X,4:POKECS-1+Y*22+X
,5
245 PRINT"■■■ CONGRATULATIONS,      Y
OU MADE IT!"
250 GOTO280
260 POKESC+Y*22+X,6:POKECS+Y*22+X,2
265 FORI=1TO1300:NEXT
268 PRINT"■YOU CRASHED AND ARE   NOW
PART OF THE CITY  SKYLINE."
280 POKE36869,240:PRINT"■■THE ENGINE
OF YOUR    PLANE HAS STALLED     WHIL
ST YOU ARE"
282 PRINT"OVER A ";
285 PRINT"LARGE CITY.YOURONLY  HOPE O
```

```
F LANDING SAFELY IS TO FLATTEN  THE C
ITY WITH YOUR"
286 PRINT"BOMBS.PRESS 'F' TO    RELEA
SE A BOMB;ONLY   ONE MAY BE FALLING A
T ANY ONE ";
290 PRINT"TIME.":PRINT"■■■ PRESS THE
'Y' KEY TO   START,'N' TO STOP.  "
300 GETA$
310 IFA$="Y"THENGOTO40
320 IFA$="N"THENSTOP
330 GOTO300
350 FORI=7168TO7207:READN:POKEI,N:NEX
T
360 RETURN
410 DATA127,127,127,73,73,73,127,127
420 DATA0,0,0,28,62,127,73,73
430 DATA255,255,255,255,255,255,255,2
55
440 DATA36,60,24,24,60,60,60,24
450 DATA0,144,136,254,63,12,8,16
```

APPENDIX
USING MATHS

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

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

The mathematical functions are:

Computer word:	Meaning:
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
EXP	(see valentino)

π	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 COMMODORE/VIC 20 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 COMMODORE/VIC 20 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 BBC MICRO
GETTING STARTED ON YOUR TRS 80/DRAGON
GETTING STARTED ON YOUR ATARI
GETTING STARTED ON YOUR ORIC

Futura Publications
Non-fiction
0 7088 24455

GB £ NET +002.95
ISBN 0-7088-2445-5
00295
9 780708 824450