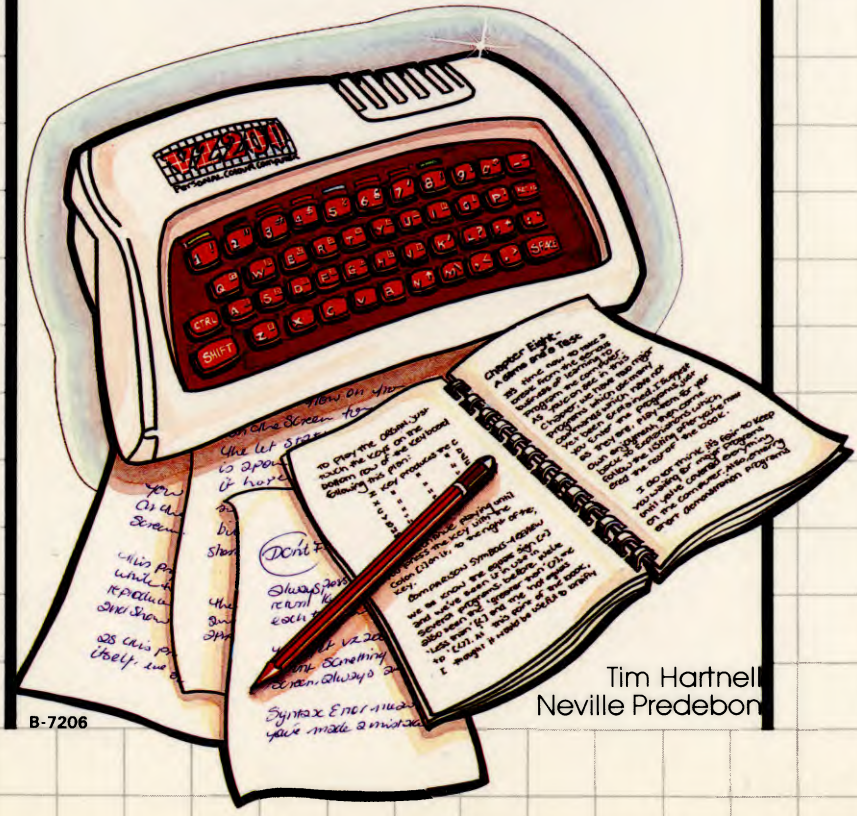
DICK SMITH

VZ200

Personal Colour Computer

Getting Started



Tim Hartnell
Neville Predebon

B-7206

Getting Started

with your

Dick Smith VZ200

Personal Colour Computer

Tim Hartnell
Neville Predebon

GETTING STARTED

WITH YOUR

DICK SMITH VZ200

By

Tim Hartnell

and

Neville Predebon

First published in Australia by:

Dick Smith Management Pty. Ltd.,
PO Box 2113,
North Ryde, NSW, 2113

First printing: September, 1983

ISBN 0 949772 21 6

Dick Smith Catalogue No. B 7206

Typesetting and artwork by Interface Publications, Level 25, Nauru House, 80 Collins Street, Melbourne, 3000, Victoria, Australia

Contents

Chapter Twelve - Getting Listed

Chapter Thirteen - Graphic Modes

Appendix - Glossary of Computer Words

Foreword

If you've never programmed a computer before, and you'd like to be able to program your computer in just a few hours, then this book was written for you.

You've probably realised how good it would be to know at least a little about programming, so you can impress your friends, instruct your kids, and create your own utility and games programs. This book will show you how to do this in just a matter of hours.

For a start, you'll learn just how few commands are needed to work your new computer so it will be up and running -- under your control -- in less than a minute after you start reading the first page of chapter one.

You'll find out the most important words for programming, and -- once you've worked through the book -- you'll have a library of interesting programs to keep you, and your computer, occupied for weeks to come.

Tim Hartnell,
Neville Predebon,
Melbourne, 1983

Chapter One - Getting Started

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 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, is not designed simply to be read. The 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

The first thing we have to do in learning to use the computer is to able to use the keyboard. While not quite the same as a typewriter keyboard, it is not too far removed from it, and should present few problems. The keyboard looks a little daunting when first seen, but you'll discover that you will feel quite confident about using it after only a few hours.

We'll be looking at the major features of the keyboard step by step.

THE CONTROL KEY

The most important key on the computer is the control key. It is located above the shift key, and marked CTRL. This key gives you access to the 'keywords' and 'functions'(above and below each key) and also controls the computer's editing features.

Holding down the control key, and pressing practically any other key, will give you the keyword written above the key.

You'll see there is another control function above the RETURN key. This allows you to access the functions below each key. These go hand in hand with the keywords to make up a program or program line. Note that you can type out a word (such as PRINT) in full if you like, letter by letter, instead of getting it in one press from a key.

When you press any key, **without using the control key**, they you'll get the main character on that key, the larger one in the lower left corner of each square. On most of the brown squares you'll see a letter; on others a number or punctuation mark.

KEYWORDS

Keywords are important parts of programs. They are the sections of the program which actually tell the computer what to do. The keywords are divided into two groups, commands and functions. We'll look at the commands first.

Many of the commands the computer obeys are like ordinary English words. PRINT, for example, means just what it says, and tells the computer to print something on the TV screen. Program lines are numbered, as you can see if you look at some of the program listings in this book, and the command GOTO (followed by a number) tells the computer - naturally enough - to GO TO that line number. You'll be pleased to discover you already know a large number of the words used in the programming language BASIC which your computer speaks.

Other words are not so obvious, such as GOSUB and DIM. Despite this, you'll have little trouble in understanding what they mean when you get to them.

RUBOUT AND INSERT

These two keys - located towards the right of the third row of keys - are part of the computer's editing system. INSERT makes room in a program line for a new character or keyword, and RUBOUT removes a part of a program line which is no longer wanted.

INSERT works by placing a space between any two characters in a line, so that you can later type in new letters in that space. To use it, you just move the cursor (that white rectangle you can see on the screen) over the character which is where you want to place your new material, and hit the INSERT key over and over again until you have as many spaces as you need.

To use the RUBOUT key, you must move the cursor to the beginning of the character or keyword you want to delete. Press CTRL and RUBOUT, and the line will disappear leftward into the cursor.

SHIFTING

There is - as we're sure you've noticed - more than one character on most of the keys. The symbols in the top right corner of each of the keys are punctuation marks, and graphics and arithmetic symbols. You get these characters by holding down the SHIFT key and the relevant character key at the same time. You get the graphics symbols (those funny partially filled-in rectangles) in the same way.

FUNCTIONS

Functions are the second group of keywords (counting the commands, which we looked at earlier, as the first group). The functions are, for the most part, written below the keys. Functions are the 'thinking keywords', that is they often are called upon to make decisions while a program is running.

Here's a simple example of a function in use:

```
10 PRINT RND(100);: GOTO 10
```

If you typed this into your computer, pressed the RETURN key afterwards, then entered the word RUN and pressed the RETURN key again, you'd see numbers between one and 100 all over the screen. Hit the BREAK key when you want the random numbers to stop.

You get functions by pressing CTRL and holding it down, press RETURN still keeping CTRL down, and then press the relevant key. This sounds complicated, but you'll find you soon get the hang of it.

To get the one line random number program above into

your computer (using keywords direct from the keyboard, rather than typing the words out in full), you'd proceed as follows (and it is actually a lot simpler than it looks):

- Type in 10 (the line number)
- Press CTRL and hit the P key to get PRINT
- Still holding down CTRL, press RETURN
- Still keeping CTRL down, press the F key, to get RND(
- Let go of the CTRL key and type 100
- Press the SHIFT key and hit the 9 key to get the closing bracket
- Let go of the SHIFT key and hit RETURN
- Type RUN and hit RETURN again

As we said, this may sound long-winded, but it actually takes less than three seconds to do. (If you've had some experience with other computers, you may prefer to type out the words in full for a while, then later on take advantage of the single touch entry system.)

THE CURSOR

That flashing white rectangle which follows you as you type on the screen is called the cursor, and it has a number of uses.

Because it can move to any part of the screen, it's very useful in editing as we discussed earlier. For example, if you wanted to change the 100 in the line given earlier into a 50, you would first move the cursor to the beginning of the the number 100. You do this with the versatile CTRL key, using the arrow above the full stop key until the cursor is at the

beginning of line 10. Then, still keeping CTRL down, press the comma key until you are over the 1 in 100.

Keep it pressed. If you go past the 1, don't worry. Just use the M key until you are over it again. Still keeping the CTRL key down, press RUBOUT once. The number should now be 00. Type in 5, which will cover the first 0. Then let go of the CTRL key and press RETURN over and over again until the cursor is below the program line, and you get the message:

```
?OUT OF DATA ERROR
```

Now type in RUN, and a number between one and 50 should appear on the screen.

THE GRAPHICS MODES

We'll only touch on this subject briefly here and will go into more detail in chapter thirteen. The computer can operate in two graphics modes. The first one (mode 0) is the standard display mode you've been using to date. The second (mode 1) is a graphics mode in which you can use colored dots (called 'pixels' in computer jargon) to create pictures.

To see this in action, enter and run the next program:

```
5 REM - MODES -
10 MODE(1): COLOR RND(8),1
15 X=0: Y=0
20 FOR N=0 TO 55: SET(X,Y)
25 X=X+1: Y=Y+1: NEXT N
30 GOTO 30
```

To get back to mode 0, press CTRL and BREAK.

INVERSE

The INVERSE command (located above the colon key) is used to provide inverse video for information held within quote marks. The computer will not accept keywords written in inverse. Try this:

```
10 PRINT "THIS IS THE COMPUTER"
```

Put the computer into inverse after the first quote mark by typing CTRL INVERSE (the rightmost key on the second row from the bottom), then follow the same procedure before the second quote mark to turn the video back to normal.

Keep practicing on the computer until you're used to the keyboard. You'll discover that the more you use your computer, the more you'll learn about its abilities and capabilities. Don't be afraid to experiment. You can't damage the computer just by typing in commands.

Now that you're familiar with the keyboard, it's time to go to chapter two, and start learning to use the computer itself.

Chapter Two - Putting Things on the Screen

We start learning to program using the most commonly-used command in BASIC, the word PRINT.

Type the following on your computer:

```
PRINT 2 (Note that you can
         either spell PRINT
         out in full or use
         the CTRL key)
```

Until you press the RETURN key, the computer will do nothing. Specifically, at this point, it will ignore the command PRINT 2. Press RETURN now, and you should see the number 2 appear underneath the words PRINT 2. This is the way PRINT works.It takes the information which follows the command PRINT, with a few exceptions which we'll learn about shortly, and PRINTs this on the screen which is, after all, exactly what you'd expect it to do.

But your computer is not completely stupid. That is, it can do more than just blindly print what you tell it to. If the word PRINT is followed by a sum, it will work it out before printing, and give you the result. Try it now. Enter the following line, then press RETURN:

```
PRINT 5 + 3 (You get + by typing
             shift ;)
```

You should see the figure 8 appear. The computer added 5 and 3 together, as instructed by the plus (+) sign, then printed the result on the screen. It can do subtraction as well (clever inventions, these

computers). Type in this, and press RETURN to see subtraction (and PRINT) at work:

```
PRINT 7 - 2
```

Now the computer can - of course - carry out 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 X symbol you probably used at school. Instead, it uses an asterisk (*) and for division the computer uses a slash (/).

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 the word PRINT, follow it with the information you want the computer to print, and that's all there is to it. But, it is not quite 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 zero. The computer thought we wanted a 'variable', rather than the word testing. We won't try to explain the meaning of the word variable at this point (it's not on the curriculum

for this chapter], but it means simply that the computer thought you wanted to print a number which had the name of 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 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 the RETURN key, the word TESTING will appear below the command statement. 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 to print. Information held in this way between quote marks is called a most peculiar name in computer circles. The jargon for the information enclosed in quote marks is 'string'. So, in our example above, the word testing, when enclosed in quote marks, is a string. [You can, in fact, get away with just the first quote mark so the line reads PRINT "TESTING but this is not good practice.]

OUR FIRST PROGRAM

Type the following into your computer. Notice that each line starts with a number. Type this into the computer, and follow this with the other material.

```
10 PRINT "JACK AND JILL"
```

Now press RETURN. You have just entered the first line of your first program.

Type in the next line, the one starting with 20, and press RETURN once you have it in place. Do the same with the rest of the lines:

```
20   PRINT "WENT UP THE HILL"
30   PRINT "TO FETCH A PAIL"
40   PRINT "OF WATER"
```

When you run this you'll see the following (if all is well):

```
JACK AND JILL
WENT UP THE HILL
TO FETCH A PAIL
OF WATER
```

ADDING NEW LINES

The computer, clever beast that it is, allows you to enter your lines in any order you choose. It will then sort them into order for you. Although our first program, and most of the other ones in this book, are numbered in 10's, starting at 10, there is no particular reason why you should follow this convention if you do not want to. However, there is

a reason for leaving 'gaps' in the counting. Although our first program could easily be numbered in 1's, it would leave no room to add later lines, if we decided there was a need to do so.

To see the computer sorting lines into order, adding the following:

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

Now type in LIST, to get the computer to list out the current program it is holding, and you'll see line 25 neatly in its proper numerical place. Now, run the program again. You should find that line 25 made no difference at all to it.

MAKING REMARKS

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 only there for your convenience, for the convenience of the programmer, or of someone else reading the program.

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

```
5    REM JACK AND JILL POEM
```

You may wonder why this would be necessary. After all, it is pretty obvious that the computer is

holding a '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 this 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. They serve to remind programmers what each section of the program 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 two -- 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'll 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 them by adding REM statements 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 entering NEW and then pressing RETURN. Type the following program into your computer and run it:

```
10   PRINT 1,2
15   PRINT
20   PRINT 1;2;3
25   PRINT
30   PRINT "COMPUTER"
35   PRINT
40   PRINT "23 + 34 = ";23 + 34
45   PRINT
50   PRINT 2*3
55   PRINT
60   PRINT 3^5
65   PRINT
70   PRINT "THE ANSWER IS ";23 + 5 - 7/6
```

Note that you can use the question mark (?) in place of the word PRINT. It becomes PRINT when you LIST the program. Now, let's look at the program we've just run. There is a lot we can learn from it.

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 between the 1 and the 2. This has the effect of making the computer print the numbers with a wide space between them. The comma can be used in this way to space out numbers neatly for a table of results or a similar purpose. (Try

PRINT 1,,2 and see what effect this has.) When you use a comma in this way, to divide the things which follow a PRINT statement (but not when the comma is inside a string, that is, between quote marks) you'll find it divides the screen up into neat little rows. Try PRINT 1,2,3 and see the result of the commas. Then you can try the effect of PRINT 1,,,2,,,3,,,4,,,5,,,6,,,7,,,8,,,9,,,0 to make it perfectly clear what is going on.

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

Line 20 has three numbers (1, 2 and 3) separated not by commas (as in line 10) 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 a single space on either side of them. When printed, numbers are always followed by a space. Positive numbers are also preceded by a space. You use the semicolon when you want printed material to follow other printed material without a break.

Line 30 is a word, 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 40 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 quotes in a PRINT statement as a number. This is why it got so upset earlier when we told it to print the word TESTING without putting the word in quote marks. It looked for a number which was called testing and because it could not find one (as we had not told the computer to let testing equal some numerical value), it refused to co-operate.

So line 40 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 50 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 60 we come across a new, and strange sign, ↑. This means 'raise to the power' so line 60 means print the result of 3 raised to the fifth power. In ordinary arithmetic, we indicate this by putting the 5 up in the air beside the three. However, it is pretty difficult for a computer to print a number halfway up the mast of another number, so we use the ↑ symbol to remind us (by pointing upward) that it really means 'print the second number up in the air'.

The final line of this program combines a string ('THE ANSWER IS') with numerical information (23 + 5 - 7/8). You can see that, as expected, the computer works out the sum before printing the answer, and prints the string exactly as it is. Look closely at the end of the string. After the closing quote there is a semicolon which, as we learned in line 20, joins various elements of a PRINT statement together. This semicolon means that the result of the calculation is printed up next to the end of the string.

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

Chapter Three - Ringing the Changes

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

CLEAR THAT SCREEN

Enter the following program into your computer and run it.

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

When you run the program, you'll see the word 'TESTING' appear under RUN, more or less as you'd expect. However, below it you'll see a question mark. Where did that come from? The question mark is known as an input prompt. An input prompt, which appears in a program when the computer comes to the word INPUT, means the computer is waiting for you to enter something else into the machine, or just to press RETURN. You'll recall that we spoke earlier about strings, and about how they were anything which was enclosed within quote marks. In line 20 above, the computer is waiting for a string input (because the A which follows the word INPUT is, in turn, followed by a dollar sign). You can either enter a word, a number, any combination of words, numbers and/or symbols in response to a string input. (But you can only type in a number in

response to a numerical input. If you just press the RETURN key when the computer wants a number, the computer will assume you want zero.)

Anyway, when you respond to the input prompt by pressing RETURN, you'll see the screen clears and the word TESTING disappears. Where did it go? We pointed out that the computer works through a program in line order. Firstly the program printed TESTING on the screen with line 10 and then progressed to line 20, where it waited for an input (or for you to press RETURN). Once you've done this in line 20, the computer moved along to line 30 where it found CLS and obeyed that instruction. The instruction was clear the screen, so the computer did just that and the screen cleared.

Run the program a few 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 the RETURN key, you can write a program which clears the screen automatically, as our next example demonstrates. Enter this next program into your computer, type in RUN and then RETURN, and then sit back for the Amazing Flashing Word demonstration. Note that you should have a space on either side of words like FOR and TO.

```
 5 REM - FORTO -
7 CLS
10 PRINT@234,"AUTOTESTING"
15 FOR D=0 TO 599
20 NEXT D
```

```
25 CLS
30 FOR D=0 TO 599
35 NEXT D
40 GOTO 10
```

Run this program, and you'll see the word AUTOTESTING alternatively flashing off and on in the middle 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 the word AUTOTESTING. 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 15 and 20 (the FOR is in line 15, the NEXT in line 20) tell the computer to count from zero to 599 before moving on. As you can see, it does this counting pretty quickly.

So, it waits for a moment while counting from zero to 599. Then it comes to line 25, which is the command CLS, which tells the computer to clear the screen. The computer then encounters, in lines 30 and 35, another FOR/NEXT loop, so waits a while as it counts from zero to 599 again. Continuing on in sequence, it comes to 40 where it finds the instruction GOTO 10. This, as is immediately obvious, tells the computer to go to line number 10. When the computer gets to line 40, it obeys the GOTO instruction, and starts over from line 10, going through the autotesting printing, counting to 599, clearing the screen, counting to 599 again, and then coming to GOTO 10 so that it starts all over again.

CHANGING PROGRAM LINES EASILY

It is relatively easy to change lines within a typed program. All you have to do is use the M and CTRL keys to the right of the fourth line on the keyboard, to move the cursor to the required position, then type in the desired changes.

If you had a line 10 which read...

```
10 REM AN EDIT TEST
```

...and you wanted it to read...

```
10 REM AN EXCITING EDIT TEST
```

...all you would need to do would be to move the cursor to the line, then use the INSERT key to make space for the word EXCITING (nine spaces) before typing it in.

After doing this, you just press the RETURN key again, and type in LIST (or use CTRL 5) and press RETURN again. This time, when the program is listed, you'll see the new version of line 10 is included within the program.

If you have only a single letter wrong within a line, you simply move the cursor to the error and then type in the correct letter or letters. These will automatically replace the incorrect material. If you want to wipe out an entire line, just enter the line number, and press RETURN.

Now, these instructions may seem a little complex. However, they do not need to be mastered before you

can continue your learning. If you're not sure how a particular line should be edited, and you can't be bothered looking it up in here or in your manual, just type the whole line again. When you press RETURN, the new line will automatically take the place of the old one within the listing.

GETTING THE PROGRAM BACK

If you want to see a complete listing after it has vanished once a program has been run, all you need to do (as we mentioned briefly befcre) is type in the word LIST then press the RETURN key.

There is no reason, when using LIST, why you must list from the top of the program. When you have longer programs, you may well want to list only part of them. You do this by use of the hyphen (-), as follows:

LIST - 100	This lists up to, and including line 100
LIST 50 - 90	This lists lines 50 to 90
LIST 150 -	This lists the program from line 150 to the end
LIST 270	This lists just line 270

USING THE PRINTER

Full instructions on printer use come, of course, with the printer, but if you prefer not to bother with them at the moment, and you just want your printer to work, these are the commands you'll need:

LLIST - to list the current program

COPY - to dump the full contents of the screen

LPRINT - this is used within a program when you want to print something on the printer, rather than on the screen

LLIST is easy to remember, as it is very similar to LIST and has a similar function, except that it lists to the printer rather than listing to the screen. In the same LPRINT is easy to remember, as it does more or less what PRINT does, except on the paper rather than the screen. Note that LLIST can be used in the same way as LIST to get just parts of a listing (so LLIST 40 - 70 is valid).

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 words which have not been explained out of the programs. However, this is not a major problem, and you'll probably be able to work out what many of them mean, just from seeing them in the context of a program line.

We are working methodically through the commands available on the computer, and in due course, all of the important ones will be covered. When you come across a word in a program which seems unfamilair, just type it in. You'll find that you'll soon start picking up the meaning of words which have not been explained, just by seeing how they are used within the program. So if you find a new word, don't worry. The program will work perfectly without you knowing what the word is, and investigating the listing after you've seen the program running is likely to lead you to work out what it means.

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

Your computer's ability to generate random numbers is very useful in order to get the computer to imitate the random events of the real world. The BASIC word RND lies at the heart of using this means of generating random numbers.

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:

```
5 REM - RND -
10 PRINT RND(0);
15 GOTO 10
```

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

```
0198217  .900392  .943668  .5964
19  .116922  .626239  .705592  .
323249  .791395  .561799  .51580
5  .0985055  .610808  .690898  .
785137  .087168  .184719  .73075
```

As you can see, RND(0) 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:

```
5 REM - RND2 -
10 PRINT RND(6);
15 GOTO 10
```

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

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

Even though we could 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'd find that the

sequence of numbers was starting to look very familiar. The random numbers, as you'd discover if you ran the program a number of times, are not really random at all.

This is because the computer does not really generate random numbers, but only looks as if it is doing so. Inside its electronic head, your computer holds a long, long list of numbers, which it prints in order when asked for random numbers. The list is so long, that it is impossible to see a pattern in it, once it is running.

FAST FOOD CRAZINESS

We'll look now at a 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 the random number generator pick your food for you:

```
5 REM - FASTFOOD -
10 CLS
15 A=RND(4)
20 PRINT "YOU'VE ORDERED ";
25 IF A=1 THEN PRINT "A HAMBURGER WITH THE LOT"
30 IF A=2 THEN PRINT "A LARGE SERVE OF FRIES"
35 IF A=3 THEN PRINT "A SERVE OF RIBS"
40 IF A=4 THEN PRINT "TWO HOT DOGS WITHTOMATO SAUCE"
45 FOR D=0 TO 599: NEXT D
50 PRINT
55 GOTO 15
```

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

```
YOU'VE ORDERED A HAMBURGER WITH THE LOT

YOU'VE ORDERED A SERVE OF RIBS

YOU'VE ORDERED A LARGE SERVE OF FRIES

YOU'VE ORDERED A HAMBURGER WITH THE LOT

YOU'VE ORDERED TWO HOT DOGS WITH
TOMATO SAUCE
```

When you look back at the listing, you'll see how the program sets the letter A to the value of the random number in line 15. In this case, the letter A is standing for a number. It is called a variable, or (because in this case it stands for a number), 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 lines 25,30,35 and 40, the value assigned to A determines which food order you place. Read this over again if it seems incomprehensible the first time.

Chapter Five - Round and Round We Go

In this chapter, we'll be introducing a very useful part of your programming vocabulary - FOR/NEXT loops. You'll recall that we mentioned FOR/NEXT loops when demonstrating the use of CLS to clear the screen. A FOR/NEXT loop was also used in our FAST FOOD program (line 45) to add a delay.

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

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

With the second like this:

```
20   NEXT A
```

The control variable, the letter after FOR and NEXT, must be the same. (You can, in fact, leave the second A out altogether, as the computer will know what you mean. However, leaving the control variable out makes programs harder to read and alter, so this practice is not recommended in your early programming days.)

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

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

When you run it, you'll see the numbers one to 20 appear on the screen, much as you may have expected.

Now try this version:

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

This is the result of running it:

```
 765  766  767  768  769  770  7
71  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 always count in this way. 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  60  70  80
90  100
```

STEPPING DOWN

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

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

This is what the program output looks like:

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

MAKING A NEST

It is possible to place one or more FOR/NEXT loops within each other. This is called nesting loops. In the next example, the B loop is nested within 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 program produces 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 need to be very careful to ensure that the first loop started is the last loop which is finished. That is, if FOR A...was the first loop you mentioned in the program, the last NEXT must be NEXT A.

Try swapping line 10 with line 20 in the program, and see what happens when you get your FORs and NEXTs mixed up.

You may recall I mentioned that you do not, in fact, have to mention the control variable with the NEXT if you do not want to. I also said that it was not good programming practice to leave it out as it made programs somewhat difficult to unravel. However, as I imagine you've realised by now, leaving off the control variables at least gets around the problem of wrongly specifying the NEXT in nested loops.

You can replace lines 40 and 50 of the program with either of the following (removing the old line 50 completely):

```
40   NEXT A:NEXT B
        or
40   NEXT:NEXT
        or
40   NEXT A,B
```

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
15 FOR B=1 TO 12
20 PRINT A;"TIMES";B;"IS";A*B
25 NEXT B: 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 program demonstrates:

```
10 FOR A=1 TO 12
15 FOR B=12 TO 1 STEP -1
20 PRINT A;"TIMES";B;"IS";A*B
25 NEXT B: NEXT A
```

Here's part of the output of that program:

```
1 TIMES 1 IS 1
2 TIMES 12 IS 24
2 TIMES 11 IS 22
2 TIMES 10 IS 20
2 TIMES 9 IS 18
2 TIMES 8 IS 16
2 TIMES 7 IS 14
```

CRACKING THE CODE

It's time now for our first real program. In this game 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 CODEBREAKER, based on a program 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 position and a 'black' is a correct digit in the right position within the four digits of the code. As you can see from this, you are aiming to get four blacks. Digits may be repated within the four-number code.

Enter the program and play a few rounds against the computer. Then, return to the book for a discussion on it, which will highlight the role played by the FOR/NEXT loops.

```
5 REM - CODEBREAKER -
10 CLS
15 PRINT@64, "********************************"
20 PRINT "WHEN YOU ARE TOLD TO DO SO,","ENTER
          A 4-DIGIT NUMBER";
25 PRINT " AND THEN HIT 'RETURN'."
30 PRINT: PRINT "DIGITS CAN BE REPEATED."
35 PRINT: PRINT "YOU WILL HAVE 8 CHANCES TO
          CRACKTHE CODE."
40 PRINT: PRINT "********************************"
45 FOR D=0 TO 4999: NEXT D: CLS
50 DIM B(4): DIM D(4)
```

```
55 H=0
60 FOR A=1 TO 4
65 B(A)=RND(9)
70 NEXT A
75 FOR R=1 TO 8
80 PRINT: PRINT "THIS IS ROUND NUMBER";R
85 PRINT: INPUT "WHAT IS YOUR GUESS";G
90 IF G<1000 OR G>9999 THEN 85
95 P=INT(G/1000)
100 Q=INT((G-1000*P)/100)
105 T=INT((G-1000*P-100*Q)/10)
110 S=INT(G-1000*P-100*Q-10*T)
115 D(1)=P: D(2)=Q: D(3)=T: D(4)=S
120 FOR E=1 TO 4
125 IF D(E)<>B(E) THEN 150
130 PRINT " BLACK";
135 B(E)=B(E)+10
140 D(E)=D(E)+20
145 H=H+1
150 NEXT E
155 IF H=4 THEN 255
160 FOR F=1 TO 4
165 D=D(F)
170 FOR X=1 TO 4
175 IF D<>B(X) THEN 195
180 PRINT " WHITE";
185 B(X)=B(X)+10
190 GOTO 200
195 NEXT X
200 NEXT F
205 FOR X=1 TO 4
210 IF B(X)<10 THEN 220
215 B(X)=B(X)-10
220 NEXT X
225 H=0
230 FOR D=0 TO 1999: NEXT D: CLS
235 NEXT R
```

```
240 PRINT: PRINT "YOU DIDN'T CRACK IT..."
245 PRINT: PRINT "THE CODE WAS ";B(1);B(2);
                B(3);B(4)
250 GOTO 265
255 PRINT@128, "WELL DONE, CODEBREAKER!"
260 PRINT: PRINT "YOU CRACKED IT IN ONLY";R;
                "ROUNDS."
265 PRINT: PRINT: INPUT "WOULD YOU LIKE
                ANOTHER GAME";A$
270 IF A$="YES" THEN RUN 50
275 PRINT "VERY WELL."
280 END
```

Here's what the screen looks like at the beginning of the run:

```
**********************************

WHEN YOU ARE TOLD TO DO SO,
ENTER A 4-DIGIT NUMBER AND THEN
HIT 'RETURN'.

DIGITS CAN BE REPEATED.

YOU WILL HAVE 8 CHANCES TO CRACK
THE CODE.

**********************************
```

And here is part of one round played against it:

```
WHAT IS YOUR GUESS? 1212
 BLACK BLACK

WHAT IS YOUR GUESS? 1234
 WHITE BLACK
```

```
WHAT IS YOUR GUESS? 5462
 WHITE BLACK BLACK
```

We'll now go through the program, line by line, a practice we'll be following in several of the programs in this book. If you don't want to read the detailed explanation now (and there may well 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 15 and 40 print a number of asterisks to rule off the title and instructions. Line 45 pauses for a few seconds so that you can read the instructions, before the screen is cleared. Arrays are dimensioned in line 50. We discuss 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, with the list called B, in which the first item can be referred to as B(1), the second as B(2) and so on. You do not really need to dimension an array when less than 11 elements will be needed, but it helps to keep your thinking clear to always dimension arrays before using them in programs. In this program the arrays are used for storing the numbers picked by the computer, and for storing the digits which you pick each time you try to break the code.

H is a numeric variable (we've mentioned numeric variables before, you'll recall) which is set equal to zero in line 55. In line 145, one is added to the value of H each time a black is found, so that if H ever gets to equal four, the computer knows all the digits have been guessed, and goes to the routine from line 255 to print up the congratulations.

The lines from 60 to 70 work out the number which you will have to try and guess. Line 55 uses the RND function we've discussed before 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 60 equals one the first time the loop is passed through, two the second time, and so on, so that the A in line 65 changes as well.

Our next FOR/NEXT loop, which uses R, starts in the next line. It counts from one to eight, to give you eight guesses. Line 85 accepts your guess, after the previous line has told you which guess it is you are entering. The numeric variable G is set equal to your guess, and line 90 checks to make sure you have not entered a five-digit number or one which has less than four digits. If you have, the program goes back to line 85 to ask you once again to enter a guess.

The next section of the program, right through to line 225, works out how well you've done, using a number of FOR/NEXT loops (120 to 150, 160 to 200, 170 to 195 and 205 to 220). Line 235 sends the program back to the line after the FOR R =... to go through the loop again. If the R loop has been run through eight times, then the program does not go back to line 75, but 'falls through' line 235 to 240 to tell you that you have not guessed the code in time, and to tell you what it is. Line 245 prints out the code.

If you do manage to guess it, so that H equals four in line 155, then the program jumps to line 255 to print out the congratulatory message.

Lines 265 to 280 provide the finishing touch. Line 265 asks you if you would like to play again. If you do, then the computer RUNs line 50. Note that this is different from GOing TO line 50. By telling the computer to RUN, we are telling it to clear out all the old values assigned to variables, so they don't effect the next game. The variables would not be wiped out if the line read GOTO 50.

Even if you don't crack the code the first time you play, you still get the chance to play again (see line 250). By going back to line 50, you're saved having to read through the instructions again.

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 reaches the final line.

This is not always true. The GOTO command sends action through a program in any order which you determine. Enter the following program, and before you run it, see if you can predict what the result of running it will be:

```
5 REM - GOTO -
10 GOTO 25
15 PRINT "THIS IS 15"
20 GOTO 35
25 PRINT "THIS IS 25"
30 GOTO 15
35 PRINT "THIS IS 35"
40 FOR Z=1 TO 200: NEXT Z
45 GOTO 25
```

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 have seen on your screen:

```
THIS IS 25
THIS IS 15
THIS IS 35
THIS IS 25
THIS IS 15
THIS IS 35
THIS IS 25
THIS IS 15
THIS IS 35
```

The program starts at line 10, and finding GOTO 25 there, moves onto line 25 to print the message "THIS IS 25". It then continues on to line 30 where it finds the instruction GOTO 15. Without question, it zips back to line 15 to print out "THIS IS 15" then goes to line 20 which directs it to line 35. At line 35 it finds the instruction to print out "THIS IS 35" which it obeys. The computer then follows through to line 40 where the Z loop inserts a short pause, before the computer moves on to line 45 to find yet another GOTO instruction, this time to line 25, which is just about where we began...and the whole thing starts over again.

RESTRICTIVE PRACTICES

Using GOTO in this way is called unconditional branching. The command is not qualified in any way, so the computer always obeys it. This brings us neatly to the next computer words we will consider. These are a pair of words IF and THEN, nearly always found (or implied) together, which impose conditions on branching by GOTO commands. This pair of words is easy to understand. IF something is true, THEN do something else. IF you are hungry, THEN order a hamburger. IF you want a big car, THEN save for it. IF something THEN 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:

```
5 REM - DIE ROLL -
10 GOTO 70
15 PRINT "ONE"
20 GOTO 70
```

```
25 PRINT "TWO"
30 GOTO 70
35 PRINT "THREE"
40 GOTO 70
45 PRINT "FOUR"
50 GOTO 70
55 PRINT "FIVE"
60 GOTO 70
65 PRINT "SIX"
70 A=RND(6)
75 FOR Z=1 TO 200: NEXT Z
80 IF A=1 THEN 15
85 IF A=2 THEN 25
90 IF A=3 THEN 35
95 IF A=4 THEN 45
100 IF A=5 THEN 55
105 IF A=6 THEN 65
110 IF A=4 THEN 45
115 IF A=5 THEN 55
120 IF A=6 THEN 65
```

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

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

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

SUBROUTINES, ANOTHER WAY TO FLY

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

The next program 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 throws the die 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 throw the die, so we do not need two identical 'die-throwing routines' within a single program. 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:

```
5 REM - DIE SUB -
10 GOSUB 90
15 PRINT: PRINT
20 FOR R=1 TO 2
25 GOSUB 70
30 IF R=1 THEN A=D
35 IF R=2 THEN B=D
40 NEXT R
42 PRINT TAB(4);
45 IF A>B THEN PRINT "I WIN"
50 IF A<B THEN PRINT "YOU WIN"
55 IF A=D THEN PRINT "IT'S A DRAW"
60 SOUND 25,1
65 RUN
70 REM DIE ROLL SUBROUTINE
75 D=RND(6)
```

```
77 PRINT TAB(4);
80 IF R=1 THEN PRINT "I ROLLED A";D
85 IF R=2 THEN PRINT "YOU ROLLED A";D
90 REM DELAY MINI-SUBROUTINE
95 FOR P=0 TO 399: NEXT P
100 RETURN
```

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

```
I ROLLED A 1
YOU ROLLED A 1
IT'S A DRAW


I ROLLED A 6
YOU ROLLED A 3
I WIN


I ROLLED A 6
YOU ROLLED A 2
```

The program pauses for a short while on line 95, prints two blank lines, then enters the R FOR/NEXT loop. When it gets to line 25 which it does (of course) once each time through the R loop, the program is sent to the subroutine starting at line 70. The 'die is rolled' in line 75, 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 (line 95) 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, that line (the one which is after the one which sent it to the subroutine) is 30.

There, the IF/THENs in lines 30 and 35 determine whether the value of the roll (D) should be assigned to the variable A or to B.

Line 40 ends the FOR/NEXT loop, and then lines 45 to 55 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 45) or whether the human has won (which will happen if A is less than B, a condition tested in line 50 by the 'less than' symbol, <). From here the program goes back to line 10 where it starts again. (By the way, you get the computer to stop running an 'endless' program of this type by holding down the CTRL key and then pressing BREAK until the program halts.)

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

Chapter Seven - Getting into Music

The SOUND command is a great way to add life to your programs. It is amazing, as you'll soon discover, what a little sound can do to enhance a program.

SOUND is always followed by two numbers (or by variables representing numbers). The first number is the pitch, or frequency, of the note to be played, and the second determines for how long the note will sound. The pitch is between 0 and 31, and the duration is between 1 and 9. The pitch value of 0 produces no sound; it is a 'rest' in music terms.

Here's a simple program which puts the SOUND statement through a few of its paces:

```
 5 REM - SOUND -
10 FOR S=0 TO 31 STEP 2
15 SOUND S,1
20 NEXT S
```

And you can combine more than one SOUND statement at a time within a loop for an even more effective result:

```
5 REM - SOUND2 -
10 FOR S=0 TO 31 STEP 2
15 N=31-S
20 SOUND S,1: SOUND N,1
25 NEXT S
```

SOUND ADVICE

The control numbers for the SOUND command can be the result of calculations, instead of being integers or assigned variables. In the next program, SOUND ADVICE, you have to guess the number between one and fifty which the computer has thought of. The feedback on each guess -- which will help you home in on the correct number in the shortest number of guesses -- is in the form of output produced by SOUND. The lower the note, the closer you are to the correct answer. Once you've played the game a few times, you'll see how easily you can interpret the output.

Here's what you see on the screen when playing the game:

```
THIS IS ROUND NUMBER 20

WHAT NUMBER AM I THINKING OF? 31


YES! I WAS THINKING OF 31

YOU GOT THE NUMBER IN ONLY 20
TRIES.

WOULD YOU LIKE TO TRY AGAIN? NO

VERY WELL, THEN.
```

Here is the listing (and note that ABS in line 50 stands for 'absolute' and gives the result of the calculation, stripped of its sign; if the result of a calculation is positive, ABS of that is still positive, while the ABS of a negative number is the number without its negative sign, so ABS (-3) is 3):

```
5 REM - SOUND ADVICE -
10 CLS
15 N=RND(50)
20 R=1
25 PRINT: PRINT "THIS IS ROUND NUMBER";R
30 PRINT: INPUT "WHAT NUMBER AM I THINKING OF";I
35 IF I<1 OR I>50 THEN 30
40 IF I=N THEN 60
45 PRINT "NO,";I;"IS NOT RIGHT."
50 S=ABS(N-I): SOUND S,1
55 R=R+1: FOR D=0 TO 199: NEXT: GOTO 25
60 PRINT: PRINT "YES! I WAS THINKING OF";N
65 SOUND 31,1
70 PRINT: PRINT "YOU GOT THE NUMBER IN
                 ONLY";R,"TRIES."
75 PRINT: INPUT "WOULD YOU LIKE TO TRY AGAIN";A$
80 IF A$="YES" THEN RUN
85 PRINT: PRINT "VERY WELL, THEN.": END
```

The first line of the program is, of course, just a REM statement to tell you the name of the program. Line 10 clears the screen. Line 15 sets the variable N equal to a number chosen at random between one and 50. This is the number which you have to try and guess. The variable R is set equal to one in line 20. As you've probably realised, this counts the number of guesses you make.

Line 25 prints up the number of the current guess,

and line 30 asks you to enter a number. The following line checks the size of your guess, rejecting it if it is above 50 or below one. Note that line 35 ends up with THEN 30, rather than THEN GOTO 30, as you may have expected. You are allowed to leave out the THEN before GOTO, as the computer will understand what you mean. If you feel, however, that the program is easier to understand with GOTO in place, by all means replace it (and do in the same in the following line, 40).

Line 40 checks the answer you've given, and if it finds your answer is the same as the number the computer has thought of, it sends action to line 60 where the congratulations message is printed.

If you are not right, the program goes on to line 45 where after printing a blank line, the 'no, you are wrong' message is displayed. Now we come to the interesting bit. The variable S is set equal to the absolute difference between the computer's number and your guess. The sound is produced in line 50, one is added to the value of the variable R, and then the computer returns to line 25 (via line 55) for the next guess.

Just to show how this program works, try the following exercise while running it. After a short run of random guesses, hit CTRL BREAK to stop the run and type PRINT N. This will display the number that the computer is thinking of.

When you've noted it, type CONT and the game will continue as before, with the computer asking you to enter your guess Before doing that, enter a few numbers around the number you saw, and note the gradual change in pitch. Then hit the computer's

number and note the sharp contrast from the lowest pitch to the highest. You'll see that the tone gets lower and lower as you get nearer the number.

The computer is capable of producing some quite exciting effects on its own, as our next program, which demands no action from you execpt for admiration, convincingly demonstrates. Take your computer to Loch Lomond next time you go there, and conjure up some Highland fancies.

```
5 REM -     BAGPIPES -
10 DIM N(8)
15 FOR S=1 TO 8
20 READ N(S)
25 NEXT S
30 LET F=S*RND(4): IF F>31 THEN 30
35 LET D=(RND(2)*RND(4))
40 SOUND F,D: GOTO 30
45 DATA 1,3,5,7
50 DATA 2,4,6,8
```

MAKING YOUR OWN MUSIC

If the 'auto-bagpipes' are too much for you, try the next program, which allows you to use the bottom row of keys as a kind of organ. It is not too musical, but you should have some fun with it, and it will give you an insight into one way of using the SOUND command.

```
5 REM - VZ ORGAN -
7 CLS
10 INPUT "ENTER A NUMBER FROM 1 TO 9";N
15 IF N<1 OR N>9 THEN 10
20 CLS
```

```
 25 PRINT:PRINT "THE BOTTOM ROW OF KEYS
       ARE THE","    ORGAN KEYS"
 30 PRINT: PRINT " HERE IS A TABLE OF
        THEIR USES;": PRINT
35 PRINT TAB(4);"Z = C";TAB(8), "X = D"
40 PRINT TAB(4);"C = E";TAB(8), "V = F"
45 PRINT TAB(4);"B = G";TAB(8), "N = A"
50 PRINT TAB(4);"M = B";TAB(8), ", = C'"
52 PRINT TAB(4);". = D';TAB(8), "  = E'"
55 PRINT TAB(4);":          =    END"
60 PRINT: PRINT "  PRESS THEM TO PRODUCE NOTES."
65 GOSUB 130
70 IF C$="Z" THEN SOUND 4,N: GOSUB 130
75 IF C$="X" THEN SOUND 6,N: GOSUB 130
80 IF C$="C" THEN SOUND 8,N: GOSUB 130
85 IF C$="V" THEN SOUND 9,N: GOSUB 130
90 IF C$="B" THEN SOUND 11,N: GOSUB 130
95 IF C$="N" THEN SOUND 13,N: GOSUB 130
100 IF C$="M" THEN SOUND 15,N: GOSUB 130
105 IF C$="," THEN SOUND 16,N: GOSUB 130
110 IF C$="." THEN SOUND 18,N: GOSUB 130
115 IF C$=" " THEN SOUND 20,N: GOSUB 130
120 IF C$=":" THEN GOTO 135
125 GOTO 65
130 C$=INKEY$:RETURN
135 CLS
140 PRINT: INPUT "WOULD YOU LIKE TO PLAY AGAIN";A$
145 IF A$="YES" THEN RUN
150 END
```

To play the ORGAN, just touch the keys on the bottom row of the keyboard, following this plan:

Z	key	produces	the	note C
X	"	"	"	D
C	"	"	"	E
V	"	"	"	F
B	"	"	"	G
N	"	"	"	A
M	"	"	"	B
,	"	"	"	C'
.	"	"	"	D'

The organ will continue playing until you press the key with the colon (:) on it, to the right of the ; key.

COMPARISON SYMBOLS - A REVIEW

We all know the equals sign (=) and we've seen it in use in several programs before. We've also seen the 'greater than' (>), the 'less than' (<) and the 'not equals to' (<>). At this point of the book, I 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.

Chapter Eight - A Game and a 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 two major programs which use many commands which have not yet been explained. I 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've mastered the rest of the book.

I do not think it's fair to keep you waiting for major programs until you've covered everything on the computer. Also, entering short demonstration programs can get pretty boring if you're longing to see your computer really in action. Therefore, I hope you'll enter the programs '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 nine.

OUT ON THE FAIRWAY

In the first game, you and your versatile computer have to tackle the Microchip Golfcourse, in the program CADDY, prepared for this book by Glen Pringle.

You have nine holes to negotiate, as as you'll see when you play the game, the computer obligingly keeps the score card for you. After each hole, it will tell you how you are doing to date, and will work out your average score per hole. All you have

to do is hit the ball! If you overshoot, the computer will automatically make sure the next shot is back towards the hole.

```
After that stroke your score is 5
The game so far:
Hole 1 took 7 strokes
Hole 2 took 5 strokes

The average so far is 6


Score up to this hole is 12
<<< Hole number 3 >>>

DIFFICULTY FACTOR IS FOUR
                o
############################\ /#########
############################ ##########
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^


Enter stroke strength? 14

You did it!!
############################\ /#########
############################ o #########
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^


After that stroke your score is 9
The game so far:
Hole 1 took 7 strokes
Hole 2 took 5 strokes
Hole 3 took 9 strokes

The average so far is 7

The score for 3 holes is 21
```

You'll find it pretty tricky going, especially on holes with a high difficulty factor.

Here's the listing, golf pro:

```
10 REM CADDY
20 DIM X(9):CO=0:H$=CHR$(216)
30 U=224:L$="                                "
40 FOR Z=1 TO 9
50 SC=0
60 J=RND(12)
70 Q=RND(3)+2
80 IF Q=5 THEN Q$="FIVE"
90 IF Q=4 THEN Q$="FOUR"
100 IF Q=3 THEN Q$="THREE"
110 CLS:PRINT:PRINT
120 IF Z=2 THEN PRINT "SCORE UP TO THIS
                    HOLE IS"X(1)
130 IF Z>2 THEN PRINT "SCORE UP TO THIS
                     HOLE IS"K
140 PRINT "<<< HOLE NUMBER"Z">>>"
150 PRINT:PRINT "DIFFICULTY FACTOR IS "Q$
160 GOSUB 430
170 PRINT:INPUT "ENTER STROKE STRENGTH"
                    ;A:SOUND 31,2
180 PRINT@U,L$:IF J>24 THEN A=-A
190 J=J+INT(A/RND(Q))
200 IF J=24 THEN GOSUB 490
205 IF J>30 THEN J=30:GOTO 205
207 IF J<1 THEN J=1
210 IF J<>24 THEN PRINT@U+J-1,H$
215 IF J<>24 THEN PRINT@352,L$:PRINT L$
220 SC=SC+1
230 PRINT@448,"AFTER THAT STROKE YOUR
                    SCORE IS"SC
```

```
240 FOR P=1 TO 2500:NEXT P
250 IF J<>24 THEN 110
260 C=C+SC
270 X(Z)=SC
280 IF Z=1 THEN 390
290 K=0
300 PRINT "THE GAME SO FAR:"
310 FOR J=1 TO Z
320 K=K+X(J)
330 PRINT "HOLE"J"TOOK JUST"X(J)"STROKES"
340 FOR M=1 TO 300:NEXT M
350 NEXT J
360 IF Z<9 THEN PRINT:PRINT "THE AVERAGE SO
             FAR IS"INT((K+.5)/Z)
370 FOR P=1 TO 1000:NEXT P
380 IF Z>1 THEN PRINT:PRINT "THE SCORE FOR"
             Z"HOLES IS"C
390 IF Z=1 THEN PRINT:PRINT "THE SCORE FOR
             THE FIRST HOLE IS"C
400 FOR M=1 TO 2500:NEXT M
410 NEXT Z
420 GOTO 560
430 IF J>30 THEN J=30
435 PRINT@196,""
440 PRINT TAB(J-1);H$
450 PRINT "########################\ /######"
460 PRINT "######################### #######"
470 PRINT "^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^"
480 RETURN
490 PRINT@416,"YOU DID IT!!"
500 PRINT@311,H$
510 FOR P=1 TO 300:NEXT P
520 SOUND 21,4:SOUND 16,2:SOUND 16,1:
            SOUND 18,4:SOUND 16,4
530 SOUND 0,1:SOUND 20,4:SOUND 21,4
540 FOR P=1 TO 2000:NEXT P
```

```
550 RETURN
560 PRINT:PRINT "END OF THAT ROUND, GOLFER!"
570 PRINT:PRINT "YOU SCORED"C
580 PRINT "AND YOUR AVERAGE WAS"INT((C+.5)/9)
590 PRINT:PRINT
600 PRINT "ENTER 'Y' FOR ANOTHER ROUND, OR
                  'N' TO QUIT"
610 A$=INKEY$
620 IF A$<> "Y" AND A$<> "N" THEN 610
630 IF A$="Y" THEN RUN
640 PRINT:PRINT "OK, THANKS FOR
                  PLAYING, CHAMP"
```

TESTING YOUR SPEED

The second program in this chapter, REACTION TEST, is great fun to play. You enter the program, type in RUN, and the message STAND BY appears. After an agnozing wait, STAND BY will vanish, to be replaced with the words, "OKAY, HIT 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.

```
OKAY - HIT THE 'Z' KEY!

YOUR SCORE IS 27

THE BEST SO FAR IS 27
```

The computer then tell you how quickly you reacted, and compares this with your previous best time. "THE 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 do such a thing!) before the whole thing begins again.

The game continues until you manage to get your reaction time to below 10, which is not an easy task.

Here's the listing of REACTION TEST:

```
5 REM - REACTION TEST -
7 CLS
10 LET HS=1000
15 FOR W=0 TO 999: IF HS<10 THEN 90
20 PRINT@236,"STAND BY"
25 GOSUB 105
30 GOSUB 100
35 IF A$<>"" THEN 25
40 LET C=0
45 PRINT@134,"OKAY - HIT THE 'Z' KEY!"
50 LET C=C+1
55 GOSUB 100: IF C>=200 THEN GOTO 90
60 IF A$<>"Z" THEN 50
65 PRINT: PRINT "YOUR SCORE IS";C
70 IF C<HS THEN LET HS=C: SOUND 30,2
75 PRINT: PRINT "THE BEST SO FAR IS";HS
80 GOSUB 105: GOSUB 100
85 IF A$<>"" THEN 80
90 NEXT W
95 PRINT: PRINT "YOU'RE THE CHAMP!":
             SOUND 31,5: END
100 LET A$=INKEY$: RETURN
105 FOR P=0 TO 499+RND(999): NEXT P:
             CLS: RETURN
```

Line 20 sets the variable HS to 1000. The variable C is set to zero in line 50 and incremented by one every time this line is revisited, which occurs when you have not managed to get to the 'Z' key. Lines 55 and 60 check to see if you have touched the Z

key, and if not, send the program back to 50 where C is incremented.

Once you've managed to get to Z, the program 'falls through' to line 65 where you are told your score. This is compared with the best score (variable name HS) in the following line, and HS is adjusted to C if C is the lower of the two.

The next line (80) puts in a short pause, and then checks to make sure you have taken your hands off the keyboard. It stays cycling through 80 and 85 until you take your hands off the keys. The NEXT W then sends the program back to the line after the FOR (line 15) and the next round of the game begins.

The FOR/NEXT continues only so long as HS stays greater than 10 (as you can see in line 15). Once you get a high score below 11, the program continues through the NEXT to lines 15 where the words "YOU'RE THE CHAMP!" appear on the screen, and SOUND 31,1 is activated.

Chapter Nine - 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 CN and GS, and combinations such as R2 and C3) and to string variables (a letter followed by a dollar sign, such as A$ or G$ is a string variable). In this chapter, we'll be looking at strings, and at things you can do with them.

THE CHARACTER SET

Every letter, number or symbol the computer prints has a code (the code, by the way, is an ASCII code and ASCII is explained in the glossary). Telling the computer to print the character of that code produces the character.

It is easy to understand this. As the code is an ASCII code, as I pointed out above, the computer word for the code is ASC. Note that the ASC value for 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". Note that we put the "A" in quote marks when we're referring to it as a letter.

Try it now. Enter the following into your computer, and see what you get:

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

Note that the letter for which you want the ASC must be within parentheses and also within quote marks, as above. Now when you get the computer to run the above line, it should give the answer 65.

From this we can see that 65 is the ASC (ASCII code) of "A". We can turn a 65 back into an "A" by asking the computer to print the character which corresponds to ASC code 65. We do this with the BASIC word CHR$, as follows:

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

Run this, and the letter "A" will appear. You can print out every ASC code and its character with the next short program. Enter it, and watch closely:

```
5 REM - CHR$ -
10 FOR C=32 TO 255
15 PRINT C; CHR$(C);" ";
20 FOR D=0 TO 99: NEXT D
25 NEXT C
```

This is the start of the printout you'll see:

```
 32    33 !  34 "  35 #  36 $  3
7 %  38 &  39 '  40 (  41 )  42
*  43 +  44 ,  45 -  46 .  47 /
 48 0  49 1  50 2  51 3  52 4  5
3 5  54 6  55 7  56 8  57 9  58
:  59 ;  60 <  61 =  62 >  63 ?
 64 @  65 A  66 B  67 C  68 D  6
9 E  70 F  71 G  72 H  73 I  74
J  75 K  76 L  77 M  78 N  79 O
 80 P  81 Q  82 R  83 S  84 T  8
5 U  86 V  87 W  88 X  89 Y  90
Z  91 [  92 \  93 ]  94 ^  95 _
 96    97 !  98 "  99 #  100 $
```

TESTING YOUR CHARACTER

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

A letter will appear on the screen. As quickly as you can, find that letter on the keyboard and press it. You'll be told how long it took you, and this time will be compared with your best time.

Notice how the letter which is printed on the screen uses CHR$ in line 30, printing the character of the number chosen at random by line 15 and assigned there to variable A. A$ is set equal to INKEY$ (which is explained a little later in the book) in line 35 and compared with the letter the computer has chosen in line 40.

```
5 REM - CHARACTER TEST -
7 CLS
10 LET BEST=200
15 LET A=65+RND(26)
20 LET S=0
25 PRINT@(32*13+7),
30 PRINT CHR$(A)
35 LET A$=INKEY$
40 IF A$=CHR$(A) THEN GOTO 70
45 LET S=S+1
50 PRINT@(32*9+5),
55 PRINT S
60 IF S<200 THEN GOTO 35
65 PRINT "SORRY - YOUR TIME IS UP": GOTO 80
70 SOUND 30,3: PRINT "WELL DONE - YOU SCORED"
```

```
75 PRINT S;"ON THAT ONE"
80 IF S<BEST THEN LET BEST=S
85 PRINT:PRINT
90 PRINT "THE BEST SCORE SO FAR IS";BEST
95 FOR P=0 TO 19*S: NEXT P: CLS: GOTO 15
```

CUTTING THEM UP

One of the very useful aspects of the BASIC on your computer is the way it can be used to manipulate strings. The words used to handle strings are:

LEFT$
MID$
RIGHT$

(By the way, these are usually spoken aloud as 'left-string', 'mid-string' and 'right-string'.)

The next program shows them in action. Enter it and run it on your computer, then return to the book for a discussion to show what can be learned from it.

```
5 REM - STRINGS -
7 CLS
10 LET A$="FIFTH*AVENUE"
15 PRINT "LEFT$(A$,3)=";LEFT$(A$,3): GOSUB 55
20 PRINT "LEFT$(A$,5)=";LEFT$(A$,5): GOSUB 55
25 PRINT "RIGHT$(A$,3)=";RIGHT$(A$,3): GOSUB 55
30 PRINT "RIGHT$(A$,5)=";RIGHT$(A$,5): GOSUB 55
35 PRINT "MID$(A$,3)=";MID$(A$,3): GOSUB 55
40 PRINT "MID$(A$,5)=";MID$(A$,5): GOSUB 55
45 PRINT "MID$(A$,5,4)=";MID$(A$,5,4): GOSUB 55
50 PRINT "MID$(A$,2,7)=";MID$(A$,2,7): END
55 FOR P=0 TO 499: NEXT P: RETURN
```

As you can see, the program first (in line 10) sets A$ equal to "FIFTH*AVENUE". Then it uses LEFT$, RIGHT$ and MID$ to extract of the original string, A$.

Here's what it looks like when you run it:

```
LEFT$(A$,5)=FIFTH

RIGHT$(A$,3)=NUE

RIGHT$(A$,5)=VENUE

MID$(A$,3)=FTH*AVENUE

MID$(A$,5)=H*AVENUE

MID$(A$,5,4)=H*AV

MID$(A$,2,7)=IFTH*AV
```

Look at the first line of the output. LEFT$(A$,3) = FIF. LEFT$ takes the leftmost portion of the string as far as the number which follows the string. That is, when we have LEFT$(A$,3) it takes the three leftmost characters of the string. The next printout, LEFT$(A$,5) takes the five leftmost characters of the string, producing in this case FIFTH (because they are five leftmost characters of the overall string).

It can be used slightly differently. If we said:

```
PRINT LEFT$("FIFTH*AVENUE",3)
```

the computer would print out FIF. The string, then, can either be a string variable (A$) or the string

in full ("FIFTH*AVENUE").

As you've probably worked out by now, RIGHT$ does the same thing as LEFT$, except it starts at the righthand end of the string. Therefore, RIGHT$(A$,3) selects the three rightmost characters of the string, in this case NUE. Again, as above, this is the same as saying:

```
PRINT RIGHT$("FIFTH*AVENUE",3)
```

MID$ is a little more flexible. It selects a portion from the middle of the string, starting from the character number which follows the string. Therefore, MID$(A$,4) prints all the string starting with the fourth character.

If there is only one number (such as the 4 above), then MID$ selects all of the string to the end of it. However, if there is another number, this second number dictates the length of the string which will be extracted.

You can see in the last two printouts from the program that MID$(A$,5,4) prints the extract of the string four characters long, starting from character five. MID$(A$,2,7) produces a string seven characters long, starting from the second character.

Rerun the program now, putting your name in place of FIFTH*AVENUE in line 10.

PUTTING THEM BACK TOGETHER

Strings can be added together on the computer. The process of adding strings is called the frightening-

looking word concatenation. You can concatenate two or more complete strings together, or just add bits of them, as our next program shows:

```
5 REM - STRINGJOIN -
7 CLS
10 LET A$="AMERICA"
15 LET B$="COLUMBUS"
20 LET C$=A$+B$
25 PRINT "A$=";A$: PRINT
30 PRINT "B$=";B$: PRINT
35 PRINT "C$=";C$: PRINT
40 LET D=RND(6): LET E=RND(6)+7
45 PRINT "MID$(C$"D","E")=";MID$(C$,D,E)
50 LET D$=MID$(C$,D,E)
55 LET E$=A$+D$
60 PRINT "E$=";E$
```

When you run this program, which creates C$ in line 35 by concatenating A$ and B$, you'll see results like these:

```
A$=AMERICA

B$=COLUMBUS

C$=AMERICACOLUMBUS

MID$(C$ 1 , 9 )=AMERICACO

E$=AMERICAAMERICACO
```

PLAYING AROUND

You can do a number of things with string manipulation, 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.

This is the listing of NAME PYRAMID:

```
5 REM - NAME PYRAMID -
7 CLS
10 PRINT: INPUT "WHAT IS YOUR FULL NAME";N$: CLS
15 IF LEN(N$)>15 THEN LET N$=LEFT$(N$,15)
20 LET N=LEN(N$)
35 FOR L=1 TO N
40 PRINT TAB(16-L);
45 FOR H=1 TO 2*L
50 PRINT MID$(N$,L,1);
55 NEXT H: PRINT
60 NEXT L
```

And here are two runs of the program, the first using the name of a famous fictional character (created by Ian Fleming) and the second uses one of our names.

```
JJ
AAAA
MMMMMM
EEEEEEEE
SSSSSSSSS

BBBBBBBBBBBBB
OOOOOOOOOOOOOOO
NNNNNNNNNNNNNNNNN
DDDDDDDDDDDDDDDDDDD
```

```
NN
EEEE
VVVVVV

PPPPPPPPPP
RRRRRRRRRRR
EEEEEEEEEEEEEE
DDDDDDDDDDDDDDD
EEEEEEEEEEEEEEEEE
BBBBBBBBBBBBBBBBBBB
OOOOOOOOOOOOOOOOOOOOO
NNNNNNNNNNNNNNNNNNNNN
```

PLAYING IT BACK

Our final program in this chapter shows one very effective use of string manipulation, 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, you will have a limited amount of time in which to press the key yourself.

If you've pressed the right key, a beep will sound, and the letter will be replaced with a new one. This will stay on the screen for a shorter time than the previous one.

Each time a new letter appear, you will be given less time to see it, before you have to press that particular key on the keyboard. If you make a mistake, the "SORRY, THAT IS WRONG" message will appear, along with your score. If you manage to get all the list of letters right, you'll be rewarded with a "YOU'RE THE CHAMP!!" message.

Here's the listing:

```
5 REM - ECHO GULCH -
7 CLS
10 LET S=0
15 LET A$="ABSCHDEUFKJHJHEUSKCJKMLKESJKDHC"
20 PRINT@(32*12+15),
25 PRINT MID$(A$,1,1)
30 GOSUB 95
35 LET I$=INKEY$
40 IF I$<"A" OR I$>I$ THEN GOTO 35
45 PRINT@(32*12+15),I$
```

```
50 IF I$=MID$(A$,1,1) THEN GOSUB 100
55 PRINT@(32*5+1),"YOUR SCORE IS";S
60 PRINT@(32*12+15)," "
65 IF I$<>MID$(A$,1,1) THEN GOTO 85
70 LET A$=MID$(A$,6)
75 IF LEN(A$)=1 THEN GOTO 105
80 GOSUB 95: GOTO 20
85 PRINT: PRINT "SORRY, THAT'S NOT QUICK
                        ENOUGH"
90 PRINT: PRINT "YOU SCORED";S: END
95 FOR D=0 TO 19*LEN(A$): NEXT D: CLS
                      : RETURN
100 FOR N=0 TO 3: SOUND 31,2: NEXT N:
                       LET S=S+1: RETURN
105 PRINT "YOU'RE THE CHAMP!"
```

The variable S, which holds your score, is set to zero in line 10, and line 15 sets the string variable A$ to a long line of letters. Line 25 prints the first letter only of the string, and line 30 inserts a short delay loop via a subroutine, 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, in our program A$ is reduced by one character by line 70 each time the program cycles, LEN A$ is a smaller number each time. Therefore, the delay produced by line 30 (which dictates how

long the character will be on the screen before it vanishes) becomes shorter.

INKEY$

Line 35 uses INKEY$ to read the keyboard. INKEY$, as you've probably worked out by this point in the book, does not demand that you press RETURN after touching a key. INKEY$ always returns the key you have pressed as a string. INKEY$ does not, in contrast to INPUT, WAIT until you have pressed a key before the program continues.

If you are not touching a key when the program comes to an INKEY$, it simple 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 40 looks at I$, the variable which is set equal to whichever key is being pressed as the program goes through line 35. As you can see in line 60, you can use the greater than (>) and the 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.

As well, you can compare strings using equals (=) and not equals (<>) as it shown by the next few lines of the program. Line 50 compares the key you have pressed with the first element of A$, and if they are the same, continues through the line to subroutine 100, which adds one to your score (variable S).

Line 60 then blanks out the letter, in preparation for the next one to appear. Line 65 compares I$ with the first element of A$ again, and if it finds they are not equal, sends the program to lines 85 and 90 where you are told "SORRY THAT'S NOT QUICK ENOUGH" and your score is given.

Line 70 strips the string A$ of its first character, by setting A$ equal to MID$(A$,6). Line 75 checks to see if the length of A$ equals 1 (that is, if LEN A$ = 1) and if it finds that it is, goes to line 105 to print out the "YOU'RE THE CHAMP!!" message. If not, the program cycles back to line 20 to print out the next letter for you.

Chapter Ten - 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:

```
5 REM - RD -
7 CLS
10 DIM A(5)
15 FOR B=1 TO 5
20 READ A(B): PRINT A(B)
25 NEXT B
30 DATA 3676,335,-89878,2142.8787,45
```

Using line 20, the program READs through the DATA statement in line 30 in order, printing up each item of DATA with line 20.

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 22, so it reads as follows:

```
5 REM - RDR -
7 CLS
10 DIM A(5)
15 FOR B=1 TO 5
20 READ A(B): PRINT A(B)
22 IF B=3 THEN RESTORE
25 NEXT B
30 DATA 3676,335,-89878,2142.8787,45
```

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 22
5 REM - SCAT RD -
7 CLS
10 DIM A(5)
15 DATA 111
20 FOR B=1 TO 5
25 READ A(B): PRINT A(B)
27 DATA 246
30 NEXT B
32 DATA 123.456,9876543210
```

READ and DATA work just as well with string information:

```
5 REM - STRING RD -
7 CLS
10 FOR B=1 TO 5
15 READ A$: PRINT A$
20 NEXT B
25 DATA VZ200,COMPUTE,PROGRAM,EXECUTE,WIN
```

Note that string DATA does not have to be enclosed within quote marks, unless leading or trailing spaces, and/or punctation and symbols are significant and must be considered part of the DATA.

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, a number comes next in the program, and when it wants a

string item, it finds it:

```
5 REM -ALPHNUM RD -
7 CLS
10 FOR B=1 TO 5
15 READ A$,A: PRINT A$,A
20 NEXT B
25 DATA MAY,2,SIMPLE,1,BY,5,HOW,7,STREET,4
```

Chapter Eleven - Adding Life to Programs

As you're becoming increasingly aware, you have a very versatile computer on your hands, one which is capable of some really innovative functions. It's possible to write some great programs using some or all of these features. So far in this book, we've looked at most of these features in isolation. In this chapter, we're going to try to put as many of them as possible into one program. This process should give you an insight into how a simple program can be elaborated into a major one.

We are going to take a very simple concept and develop a versatile games program around it. We'll start with a 'skeleton' program — one which is the just the bare bones of the final product — and gradually build around it to create a much more complex game.

We do this by adding features such as color, sound, scoring, error traps and a better display. Our basic aim is to use as many as possible of the things we've learned in one program, and to use them efficiently. There is no point in just 'dressing up' a program and wasting memory; each thing we add to the program must earn its keep by making a real contribution.

The program we are going to develop is called NUMGESS. As your brilliant mind has probably already deduced, it is a number guessing game. The aim of the game is to discover the number the computer has thought of. The machine helps by providing feedback on the accuracy of your guesses. You have a limited number of guesses per game.

Here's the listing of the first version of the program:

```
5 REM  -## NUMGESS ##-
7 CLS
10 PRINT@108,"<NUMGESS>": PRINT@293,"A
        NUMBER GUESSING GAME"
15 FOR DELAY=0 TO 999: NEXT DELAY: CLS
20 PRINT@70,"INSTRUCTIONS:-": PRINT
25 PRINT "YOU WILL HAVE 10 CHANCES TO GET"
30 PRINT "THE NUMBER THAT I'M THINKING OF,";
35 PRINT "WHICH IS BETWEEN 1 AND 100."
40 PRINT@327,"HIT 'S' TO START."
45 A$=INKEY$: IF A$<>"S" THEN 45
50 CLS
55 NUMB=RND(100)
60 FOR ROUND=1 TO 10
65   PRINT: INPUT "   WHAT IS YOUR GUESS";GUESS
70   IF GUESS=NUMB THEN 105
75   IF GUESS>NUMB THEN 85
80   PRINT: PRINT TAB(4);"HIGHER": GOTO 90
85   PRINT: PRINT TAB(4);"LOWER"
90 NEXT ROUND
95 FOR DELAY=0 TO 99: NEXT DELAY: CLS
100 PRINT@260,"SORRY - YOU MISSED IT.": END
105 CLS: PRINT@64,"THAT'S CORRECT! YOU'VE
        WON A","FREE GAME!"
110 RUN 50
```

We'll go briefly through this first version. Lines 10 to 35 are the instructions, telling you what the program is about. They inform you that you'll have ten chances to guess the number the computer is thinking about. This number is between 1 and 100. Lines 40 and 45 ask you to hit the 'S' key when you've read the instructions. Note line 45. It scans the keyboard over and over again until it discovers

that you have hit the 'S' key. If you don't hit 'S', it loops through itself again. Line 50 clears the screen.

The following line sets the variable name NUMB to a random number between 1 and 100. The next lines, 60 through to 90, begin the playing loop. The FOR variable ROUND is set from 1 to 10. Line 65 asks you for, and accepts, your guess.

The next two lines decide what to do with your input. If your guess is equal to the variable NUMB, then you've won, and the program jumps out of the loop to line 105 (the winner's box). If, however, your guess does not equal NUMB, then you are moved to line 75. This decides whether or not your number is greater than NUMB, and if so, moves action to line 85. If not, the program 'falls through' to 80, which tells you you are too low, then moves onto the NEXT statement in line 90.

If you have ten tries, and do not guess the number, line 100 is used to tell you you've failed. You get a free game only if you manage to guess the number.

The game is OK as it is, but it could easily be enhanced. Try the next version, which adds a score facility:

```
5 REM  -## NUMGESS2 ##-
7 CLS
10 PRINT@108,"<NUMGESS>": PRINT@293,"A NUMBER
        GUESSING GAME"
15 GOSUB 140
20 PRINT@70,"INSTRUCTIONS:-": PRINT
25 PRINT "YOU WILL HAVE 10 CHANCES TO GET"
30 PRINT "THE NUMBER THAT I'M THINKING OF.";
```

```
35 PRINT "WHICH IS BETWEEN 1 AND 100."
40 PRINT@327,"HIT 'S' TO START."
45 I$=INKEY$: IF I$<>"S" THEN 45
47 SC=20
50 CLS
55 NUMB=RND(100)
60 FOR ROUND=1 TO 10
62 PRINT: PRINT TAB(2);"THIS IS ROUND NUMBER"
                       ;ROUND
65   PRINT: INPUT "   WHAT IS YOUR GUESS"
                       ;GUESS
70   IF GUESS=NUMB THEN 105
75   IF GUESS>NUMB THEN 85
80   PRINT: PRINT TAB(4);"HIGHER": GOTO 90
85   PRINT: PRINT TAB(4);"LOWER"
90 GOSUB 140: NEXT ROUND
95 GOSUB 140
100 PRINT@260,"SORRY - YOU MISSED IT.":
                      SC=SC-5: GOTO 115
105 CLS: PRINT@64,"THAT'S CORRECT! YOU'VE
                      WON A","FREE GAME!"
110 SC=SC+5
115 PRINT: PRINT TAB(3);"YOUR SCORE IS NOW";SC
120 GOSUB 140: IF GUESS=NUMB THEN 50
125 PRINT@256,"WOULD YOU LIKE ANOTHER GAME";
130 INPUT A$: IF A$="Y" THEN RUN 50
135 PRINT: PRINT TAB(5);"VERY WELL.": END
140 FOR DELAY=0 TO 999: NEXT DELAY: CLS: RETURN
```

Well, that's quite a difference. The first obvious change is that the delay has been made into a subroutine in line 140. Lines 15, 90, 95 and 120 now have access to it, but the subroutine in line 95 should really be eliminated.

Other additions include line 62 which tells you

which round you are up to. Line 47 sets SC (for score) to 20. Additions have been made to lines 100 and 110 to reset the score according to the outcome, that is, whether you have won or lost. Your new score is shown by line 115. Losers now have the chance to play again with the help of line 125, while winners go straight back to 50 (see line 120). If winners want to play again, they are sent back to line 50; if not, the game ends with the words "VERY WELL".

One serious problem which can occur with programs is a result of the player misreading the instructions, and entering, for example, a letter when a number has been requested. In the next version of NUMGESS, several 'error traps' (or 'mug traps' as they are often called) are included to prevent user error causing the program to crash. Here's the third version:

```
5 REM  -## NUMGESS3 ##-
7 CLS
10 PRINT@107,"<NUMGESS>": PRINT@293,"A
        NUMBER GUESSING GAME"
15 GOSUB 140
20 PRINT@70,"INSTRUCTIONS:-": PRINT
25 PRINT "YOU WILL HAVE 10 CHANCES TO GET"
30 PRINT "THE NUMBER THAT I'M THINKING OF,";
35 PRINT "WHICH IS BETWEEN 1 AND 100."
40 PRINT@327,"HIT 'S' TO START."
45 I$=INKEY$: IF I$<>"S" THEN 45
47 SC=20
50 CLS
55 NUMB=RND(100)
60 FOR ROUND=1 TO 10
62 PRINT: PRINT TAB(2);"THIS IS ROUND
        NUMBER";ROUND
```

```
  65   PRINT: INPUT "   WHAT IS YOUR GUESS"
                    ;GUESS
  66 IF ROUND=10 THEN 100
  67 IF GUESS<0 OR GUESS>100 THEN GOSUB
                    145: GOTO 62
  70   IF GUESS=NUMB THEN 105
  75   IF GUESS>NUMB THEN 85
  80   PRINT: PRINT TAB(4);"HIGHER": GOTO 90
  85   PRINT: PRINT TAB(4);"LOWER"
  90 GOSUB 140: NEXT ROUND
  100 PRINT@260,"SORRY - YOU MISSED IT.":
                    SC=SC-5: GOTO 115
  105 CLS: PRINT@64,"THAT'S CORRECT! YOU'VE
                    WON A","FREE GAME!"
110 SC=SC+5
115 PRINT: PRINT TAB(3);"YOUR SCORE IS NOW";SC
120 GOSUB 140: IF GUESS=NUMB THEN 50
125 PRINT@256,"WOULD YOU LIKE ANOTHER
                    GAME";: DIM A$(1)
130 INPUT A$: IF A$="Y" THEN 50
135 PRINT: PRINT TAB(5);"VERY WELL.": END
140 FOR DELAY=0 TO 999: NEXT DELAY: CLS
                    : RETURN
145 CLS: PRINT@166,"STICK TO THE RULES!": PRINT
150 PRINT: PRINT "THAT NUMBER WAS UNACCEPTABLE."
155 PRINT "YOUR INPUT SHOULD HAVE BEEN"
                ,"BETWEEN 1 AND 100."
160 PRINT: PRINT TAB(4);"TRY AGAIN.":
                GOSUB 140: RETURN
```

Again you'll see this provides a considerable improvement. The additions in this version guard against most errors. However, if you find someone using your program manages to get past the traps you've built in, try and write traps which will stop this happening again. Trying to mug-trap programs

proves the truth of the old saying: THE TROUBLE WITH TRYING TO MAKE THINGS FOOLPROOF, IS THAT FOOLS ARE SO INVENTIVE.

You'll see that this version of the program has a 'number barrier' check. This means that if your input is lower than 0 or greater than 100, the program will enter the subroutine at line 145 and tell you off. After a short delay, it will return you to the program again at line 62. Notice how the delay subroutine is accessed from inside the 'telling off' subroutine at line 160.

Another addition is line 66. This is only activated when the FOR variable ROUND is equal to ten. Instead of telling you whether you were too high or too low, it cuts out that delay and jumps straight to line 100, where you are told that you've lost.

The final addition lies at the end of line 125. The DIM statement sets the reply to one character only, so you can write YES, NO, Y, N, NO WAY, YO, NOT LIKELY or even YESSIRREE. The computer will read - and act on - the first character only.

The listing is, as you can see, getting quite complicated, although the game itself is still pretty straightforward. There are a few leaps and conditional actions, but nothing random to confuse you, or challenging to test you. The next version remedies this. It provides several variations of the basic game format to keep you on your toes.

You'll see this, when you enter and run version four of NUMGESS:

```
5 REM  -## NUMGESS4 ##-
7 CLS
10 PRINT@107,"<NUMGESS>": PRINT@293,"A NUMBER
                  GUESSING GAME"
15 GOSUB 140
20 PRINT@70,"INSTRUCTIONS:-": PRINT
25 PRINT "YOU WILL HAVE 10 CHANCES TO GET"
30 PRINT "THE NUMBER THAT I'M THINKING OF,";
35 PRINT "WHICH IS BETWEEN 1 AND 100."
40 PRINT@327,"HIT 'S' TO START."
45 I$=INKEY$: IF I$<>"S" THEN 45
46 PLAY=10
47 SC=20
48 LIM=100
50 CLS
55 NUMB=RND(LIM)
60 FOR ROUND=1 TO PLAY
62   PRINT: PRINT TAB(2);"THIS IS ROUND
                  NUMBER";ROUND
64   IF ROUND=PLAY THEN PRINT TAB(2);"THIS
                  IS YOUR LAST ROUND."
65   PRINT: INPUT "   WHAT IS YOUR GUESS"
                  ;GUESS
66   IF ROUND=PLAY THEN CLS: GOTO 100
67   IF GUESS<1 OR GUESS>LIM THEN GOSUB 145:
                  GOTO 62
70   IF GUESS=NUMB THEN 105
75   IF GUESS>NUMB THEN 85
80   PRINT: PRINT TAB(4);"HIGHER": GOTO 90
85   PRINT: PRINT TAB(4);"LOWER"
90   GOSUB 140
95 NEXT ROUND
100 PRINT@196,"SORRY - YOU MISSED IT. IT"
              ,"    WAS";NUMB
102 LIM=LIM-10
103 IF LIM<=30 THEN LIM=30
104 GOSUB 165: GOTO 115
```

```
  105 PRINT@64,"THAT'S CORRECT! YOU'VE WON A",
                    "FREE GAME"
  110 SC=SC+5: PLAY=PLAY-2: LIM=LIM+50
  112 IF PLAY<=4 THEN PLAY=4: GOSUB 165
  115 PRINT: PRINT TAB(3);"YOUR SCORE IS NOW";SC
  120 GOSUB 140: IF GUESS=NUMB THEN 50
  125 PRINT@256,"WOULD YOU LIKE ANOTHER GAME";
  130 INPUT A$: IF LEFT$(A$,1)="Y" THEN 50
  135 PRINT: PRINT TAB(5);"VERY WELL.": END
  140 FOR DELAY=0 TO 999+RND(299): NEXT DELAY:
                    CLS: RETURN
145 CLS: PRINT@166,"STICK TO THE RULES!": PRINT
150 PRINT: PRINT "THAT NUMBER WAS UNACCEPTABLE."
155 PRINT "YOUR INPUT SHOULD HAVE BEEN"
                  ,"BETWEEN 1 AND 100."
160 PRINT: PRINT TAB(4);"TRY AGAIN.": GOSUB 140:
                  RETURN
165 PRINT: PRINT "  YOUR NEXT GAME WILL HAVE";
                PLAY,"  ROUNDS,";
170 PRINT " AND THE LIMIT WILL BE",;" ";LIM
175 GOSUB 140: RETURN
```

As you can see, the listing is getting quite crowded. Some of the line numbers have no room between them and the next line (such as 64, 65, 66 and 67) leaving no room to add new lines between them. It is best to try and write your programs with line numbers 5 or 10 apart so you can add extra material if the need to do so arises.

As the game progresses in this version, certain conditions are followed. The limit will become higher or lower according to whether you win or lose, and you will have a larger or smaller number of chances in your next game, depending on the outcome. If you win, you'll be given fewer guesses

in the following game, and the upper limit of the number to be guessed will rise by 50. If you lose, you'll be given two extra rounds, and the limit will fall by 10.

Notice that some of the variables have been changed. The FOR/NEXT loop which starts in line 60 has an upper limit of a variable called PLAY, rather than 10, as in the previous versions. NUMB now equals a random number less than LIM, rather than less than 100. The DIM statement has been removed and has been replaced by LEFT$. The major addition is the section in line 165 onwards. This subroutine tells you about the new situation which faces you in your next game.

Well, we've almost finished, but not quite. We thought the program was too quiet, so decided to add some noise. Our next version takes care of the sound, and adds a few other things as well (note that you should replace the 32 spaces in line 12 with the graphics character you get from shifted J, to provide a colored line) :

```
5 REM  -## NUMGESS5 ##-
7 CLS
10 PRINT@107,"<NUMGESS>": PRINT@261,"A NUMBER
                GUESSING GAME"
12 COLOR RND(8): PRINT "                              "
15 GOSUB 140
20 PRINT@70,"INSTRUCTIONS:-": PRINT
25 PRINT "YOU WILL HAVE 10 CHANCES TO GET"
30 PRINT "THE NUMBER THAT I'M THINKING OF,";
35 PRINT "WHICH IS BETWEEN 1 AND 100."
40 PRINT@327,"HIT 'S' TO START."
45 I$=INKEY$: IF I$<>"S" THEN 45
46 PLAY=10
47 SC=20
```

```
 48 LIM=100
 50 CLS
 55 NUMB=RND(LIM)
 60 FOR ROUND=1 TO PLAY
 62   PRINT: PRINT TAB(2);"THIS IS ROUND
          NUMBER";ROUND
 64   IF ROUND=PLAY THEN PRINT TAB(2);"THIS
          IS YOUR LAST ROUND."
65   PRINT: INPUT "   WHAT IS YOUR GUESS";GUESS
66   IF ROUND=PLAY THEN CLS: GOTO 100
67   IF GUESS<1 OR GUESS>LIM THEN GOSUB
          145: GOTO 62
70   IF GUESS=NUMB THEN 105
75   IF GUESS>NUMB THEN 85
80   PRINT: PRINT TAB(4);"HIGHER": GOTO 90
85   PRINT: PRINT TAB(4);"LOWER"
90   GOSUB 140
95 NEXT ROUND
100 SOUND 9,4: PRINT@192,"SORRY - YOU MISSED
          IT. IT","WAS";NUMB
102 LIM=LIM-10
103 IF LIM<=30 THEN LIM=30
104 GOSUB 165: GOTO 115
105 GOSUB 180
110 SC=SC+5: PLAY=PLAY-2
111 LIM=LIM+50
112 IF PLAY<=4 THEN PLAY=4
113 GOSUB 165
115 PRINT: PRINT TAB(3);"YOUR SCORE IS NOW";SC
120 GOSUB 140: IF GUESS=NUMB THEN 50
125 PRINT@256,"WOULD YOU LIKE ANOTHER GAME";
130 INPUT A$: IF LEFT$(A$,1)="Y" THEN 50
135 PRINT: PRINT TAB(5);"VERY WELL.": END
140 FOR DELAY=0 TO 999+RND(299): NEXT DELAY:
          CLS: RETURN
```

```
145 CLS: SOUND 1,5: PRINT@166,"STICK TO THE
              RULES!": PRINT
150 PRINT: PRINT "THAT NUMBER WAS UNACCEPTABLE."
155 PRINT "YOUR INPUT SHOULD HAVE BEEN",
          "BETWEEN 1 AND 100."
160 PRINT: PRINT TAB(4);"TRY AGAIN.": GOSUB
           140: RETURN
165 PRINT: PRINT "  YOUR NEXT GAME WILL HAVE"
          ;PLAY,"  ROUNDS,";
170 PRINT " AND THE LIMIT WILL BE",;" ";LIM
175 GOSUB 140: RETURN
180  FOR WIN=0 TO 5: CLS: SOUND 31,1:
          PRINT@RND(492),"CONGRATS!"
185 NEXT WIN
190 PRINT@64,"THAT'S CORRECT! YOU'VE WON A",
          "FREE GAME": RETURN
```

There's quite a difference between that version, and the one which first appeared in this chapter, isn't there? If you win in this version, you are greeted by the display in subroutine 180.If you lose, the computer lets you know what it thinks of you with the noise created by line 100. Another result awaits you if you try to enter numbers outside the limit.

To emphasise certain words, you should put the strings in lines 20, 40, 64, 125, 145 and 180 in inverse video. The word NUMGESS in line 107 should also be in inverse.

The program is now in pretty good shape, although the listing is very cluttered. Some of the line numbers are very close to each other, and the subroutines are not clearly marked. So here is the final, properly-formatted, version of NUMGESS:

```
5 REM -##  NUMGESS6  ##-
7 CLS
10 PRINT@106,"-<NUMGESS>-": PRINT@261,
       "A NUMBER TRAPPING GAME"
15 COLOR RND(8): PRINT "                              "
20 GOSUB 190
25 PRINT@68,"INSTRUCTIONS:-": PRINT
30 PRINT "YOU WILL HAVE 10 CHANCES TO GET"
35 PRINT "THE NUMBER THAT I'M THINKING OF,";
40 PRINT "WHICH IS BETWEEN 1 AND 100."
45 PRINT@324,"HIT 'S' TO START."
50 I$=INKEY$: IF I$<>CHR$(83) THEN 50
55 PLAY=10: SC=20: LIM=100
60 CLS: NUMB=RND(LIM)
65 FOR ROUND=1 TO PLAY
70   PRINT: PRINT TAB(3);"THIS IS ROUND NUMBER"
                    ;ROUND
75   IF ROUND=PLAY THEN PRINT TAB(3);"THIS IS
               YOUR LAST CHANCE"
80   PRINT: INPUT "  WHAT IS YOUR GUESS";GUESS
85   IF ROUND=PLAY THEN CLS: GOTO 125
90   IF GUESS<1 OR GUESS>LIM THEN GOSUB 200:
                 GOTO 70
95   IF GUESS=NUMB THEN GOSUB 245: GOTO 145
100   IF GUESS>NUMB THEN 110
105   PRINT: PRINT TAB(4);"HIGHER": GOTO 115
110   PRINT: PRINT TAB(4);"LOWER"
115   GOSUB 190
120 NEXT ROUND
125 SOUND 9,4: PRINT@192,"SORRY - YOU MISSED
        IT. IT","WAS";NUMB
130 LIM=LIM-10
135 IF LIM<=30 THEN LIM=30
140 GOSUB 225: GOTO 165
145 SC=SC+5
150 PLAY=PLAY-2: IF PLAY<=4 THEN PLAY=4
155 LIM=LIM+50
```

```
160 GOSUB 225
165 PRINT: PRINT TAB(3);"YOUR SCORE IS NOW";SC
170 GOSUB 190: IF GUESS=NUMB THEN 60
175 PRINT@256,"WOULD YOU LIKE ANOTHER GAME";
180 INPUT A$: IF LEFT$(A$,1)=CHR$(89) THEN 60
185 PRINT: PRINT TAB(5);"VERY WELL. GOODBYE."
                : END
190   REM - DELAY SUBROUTINE -
195 FOR DELAY=0 TO 999+RND(299): NEXT DELAY:
              CLS: RETURN
200   REM - SCOLD SUBROUTINE -
205 CLS: SOUND 1,5: PRINT@166,"STICK TO THE
                RULES!"
210 PRINT: PRINT "THAT ENTRY IS UNACCEPTABLE.
                 YOUR";
215 PRINT "INPUT SHOULD HAVE BEEN BETWEEN
                1AND";LIM
220 PRINT: PRINT TAB(4);"TRY AGAIN.":
              GOSUB 190: RETURN
225   REM - SITUATION SUBROUTINE -
230 PRINT: PRINT TAB(2);"YOUR NEXT GAME WILL
               HAVE";PLAY
235 PRINT TAB(2);"ROUNDS, AND THE LIMIT
             WILL BE",;" ";LIM
240 GOSUB 190: RETURN
245   REM - WIN SUBROUTINE -
250 FOR WIN=0 TO 5: CLS: SOUND 31,1:
          PRINT@RND(492),"CONGRATS!"
255 NEXT WIN
260 PRINT@64,"THAT'S IT! YOU'VE WON A",
          "FREE GAME!": RETURN
```

As you can see from what we've done to NUMGESS in this chapter, any program can be changed and developed. If you see a program in a book or magazine and think you'd like to have it running

on your own computer, by all means enter it. But don't stop there.

There is always something which can be added to a program to make it more interesting to run. This does not apply only to games programs. You might like to work on some of the other programs in this book, or even continue to work on NUMGESS, putting into practice the ideas outlined in this chapter.

Chapter Twelve - Getting Listed

An array is used when you want to create a list of items, and refer to the item by just mentioning the position within the list the item occupies. You set up an array by using the command DIM (for dimension). If you type in DIM A(20), the computer will set up a list in its memory called A, and will save space for twenty-one items: A(0), A(1), A(2)...and so on....up to A(20). Each of these items -- the A(7) and the rest -- 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 told your computer to PRINT A(3) it would print a 0. You fill the items in an array with a statement like A(2) = 1000, or by using READ and DATA as we saw in chapter ten. Once you've given an element a value, you can get the computer to tell you what value the element has by saying PRINT A(n). You can also manipulate the element as though it was the number. That is, A(4)*6 is valid, as is 45 - A(6) and so on.

The computer will let you use an array of up to 11 elements (that is A(0) through to A(10)) without having to use the DIM statement first. The moment it comes across a reference to an element of an array, where the subscript (the number which follows in parentheses) is between 0 and 10, it automatically creates an array. However, it is good practice to always dimension arrays, even if you are using less than 12 elements.

You may like to 'forget' about the element which has

the subscript zero, and pretend that the array starts at one. Many times you'll find it simpler to assume DIM A(80) gives you an array of 80 elements (rather than 81 as is the case), and that the first element is A(1) rather than A(0).

The first program in this chapter dimensions (sets up, or creates) an array called A with room for sixteen elements. We will ignore the element with the subscript 0. The B loop, from lines 15 to 25, fills the array with random digits between 1 and 9, and then prints them back for you with the loop from 30 to 45 (with a slight pause being created by line 40).

Here is the listing:

```
5 REM - ARRAYS -
7 CLS
10 DIM A(15)
15 FOR B=1 TO 15
20 LET A(B)=RND(9)
25 NEXT B
30 FOR Z=1 TO 15
35 PRINT TAB(4);"A(";Z;") IS ";A(Z)
40 FOR D=0 TO 499: NEXT D
45 NEXT Z
```

And here's one example of it in use:

```
A( 2 ) IS  1
A( 3 ) IS  2
A( 4 ) IS  3
A( 5 ) IS  8
A( 6 ) IS  5
A( 7 ) IS  3
```

```
A( 8 ) IS  7
A( 9 ) IS  9
A( 10 ) IS  1
A( 11 ) IS  2
A( 12 ) IS  6
A( 13 ) IS  7
A( 14 ) IS  2
A( 15 ) IS  3
```

This is called a one-dimensional array, because a single digit follows the letter or name which labels the array.

You can also have multi-dimensional arrays, in which more than one number follows the array label after DIM. In our next program, for example, the computer sets up a two-dimensional array called A again, consisting of five elements by five elements (that is, it is dimensioned by DIM A(4,4) as you can see in line 10):

```
5 REM - MULTARRAY -
7 CLS
10 DIM A(4,4)
15 FOR B=1 TO 4: FOR C=1 TO 4
20 LET A(B,C)=RND(9)
25 NEXT C: NEXT B
30 COLOR 8: PRINT "     1  2  3  4"
     : PRINT TAB(3);"############"
35 FOR B=1 TO 4: PRINT B;"#";
40 FOR C=1 TO 4: PRINT A(B,C);
45 NEXT C: PRINT: NEXT B
```

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

```
     1   2   3   4
   ############
1  #  1   1   9   4
2  #  9   3   2   5
3  #  3   6   1   2
4  #  6   7   8   9
```

You specify the element of a two-dimensional array by referring to both its numbers, so the element 1,1 of this array (the element in the top left hand corner of the printout above) is 3 and is referred to as A(1,1). The 8 in the printout is A(4,3), and the 9 next to it is A(4,4).

Your computer also supports string arrays. Enter and run the following short program to see string arrays in operation:

```
5 REM - STRING ARRAY -
7 CLS
10 DIM A$(5)
15 FOR B=1 TO 5
20 LET A$(B)=CHR$(RND(26)+65)+CHR$(RND(26)+65)
25 NEXT B: PRINT
30 FOR B=1 TO 5
35 PRINT TAB(4); "A$(";B;") IS ";A$(B)
40 NEXT B
```

Here's one printout of the program:

```
A$( 1 ) IS RE
A$( 2 ) IS WN
A$( 3 ) IS EO
A$( 4 ) IS QI
A$( 5 ) IS UG
```

You can, of course, fill the elements of an array — string or numeric — via DATA or INPUT statements. Here is a string array which is filled by a DATA statement:

```
5 REM - DATARRAY -
7 CLS
10 DIM A$(5)
15 FOR B=1 TO 5
20 READ A$(B)
25 NEXT B: PRINT
30 FOR B=1 TO 5
35 PRINT TAB(3);"A$(";B;") IS ";A$(B)
40 NEXT B
45 DATA COMPUTING,IS,A,FUN,HOBBY
```

This is the result of running it:

```
A$( 1 ) IS COMPUTING
A$( 2 ) IS IS
A$( 3 ) IS A
A$( 4 ) IS FUN
A$( 5 ) IS HOBBY
```

ESCAPE FROM MURKY MARSH

The next program demonstrates the use of a two-dimensional array for 'holding' an object, and for moving it around within the array. The shape is trapped in a murky marsh, and by moving totally at random, it hopes one day to be able to escape from the marsh. The shape is free if it manages to stumble onto the outer rows.

(By the way, the shape in this program demonstrates Brownian motion, the random movement shown by such

things as tiny particles in a drop of water when viewed under a microscope, or of a single atom of gas in a closed container. Brownian motion explains why a drop of ink gradually mixes into the water into which it has been placed.)

Here is the program listing:

```
5 REM - MURKY MARSH -
7 CLS
10 DIM A(10,10)
15 LET M=0
20 GOSUB 130
25 FOR W=0 TO 99: IF Q>9 OR P>9 THEN GOTO 125
30 IF RND(0)>.35 THEN LET P=P+1 ELSE LET P=P-1
35 IF RND(0)>.35 THEN LET Q=Q+1 ELSE LET Q=Q-1
40 IF Q<1 THEN LET Q=1
45 IF Q>10 THEN LET Q=10
50 IF P<1 THEN LET P=1
55 IF P>10 THEN LET P=10
60 LET M=M+1
65 PRINT@(32*3+1),"ATTEMPT#";M;" ";
70 LET A(P,Q)=94
75 PRINT@(32*7+20),
80 FOR X=1 TO 10
85 FOR Y=1 TO 10
90 PRINT CHR$(A(X,Y));" ";
95 NEXT Y
100 PRINT: PRINT TAB(20);
105 SOUND 1+INT(X/5)*Y,1
110 NEXT X
115 LET A(P,Q)=94
120 CLS: NEXT W
125 GOTO 165
130 REM ARRAY ORDER SUBROUTINE
135 LET Q=RND(3)+4
140 LET P=RND(3)+4
```

```
145 FOR X=1 TO 10
150 FOR Y=1 TO 10
155 LET A(X,Y)=254
160 NEXT Y,X: RETURN
165 PRINT: PRINT TAB(5);"WHEW - FREE AT LAST!"
170 FOR D=0 TO 2999: NEXT D: RUN
```

Chapter Thirteen - Graphic Modes

Your computer can produce some very effective displays, using the different modes and colors.

THE MODES

As you know, your computer can operate in two display ways, called modes. The first mode, and the one most frequently used, is mode 0, which is predominately a text mode. The computer is in this mode when it is first turned on.

You can use numbers, letters and the onboard 'chunky' graphics in mode 0. This mode has a resolution of 32 by 16. That is, the screen can fit 32 characters across and 16 down, as if it was divided into 512 little rectangles, each of which can only hold one character. Using the chunky graphics gives you the effect of slightly higher resolution (64 by 32).

The second mode, mode 1, is the high resolution graphics mode. Very fine individual points can be printed on the screen to create very clear pictures. You cannot, however, print text in this mode.

The resolution in this mode is 128 by 64; that's 128 points across and 64 down. The co-ordinates of these dots begin in the top left hand corner of the screen (at co-ordinates 0,0).

USING COLOR

You can use color in both modes. The background screen color can be green or orange. As you know,

the screen is green when the computer is first turned on, but it can be change to orange as follows:

```
COLOR,1
```

To turn the screen green again, without turning the unit off, enter this:

```
COLOR,0
```

There are eight colors which can be used in both modes. In mode 0, these colors can only be used on any one of the onboard graphics characters. High resolution color is slightly different from that used in mode 0. You have to assign a color to the plotted points, or they won't show up.

The high resolution background colors are green and grey. You need to use two numbers, as well as the two co-ordinates, to assign a color to a pixel (which is what a high resolution point is called).

Try this:

```
10 MODE(1): COLOR 7,1: SET(100,50): GOTO 10
```

The SET function assigns the co-ordinates for a point. SET is below the H key. Two numbers, or variables, allow SET to plot a point at the co-ordinates given. RESET clears these points from the screen.

Finally, in this brief chapter, you might like to try the following programs to see the two modes in

action. As you can see the first one, for mode 0, uses PRINT@ to position the colored blobs, while the second program uses SET.

```
5 REM - TEXT RES DEMO -
10 COLOR,1
15 INPUT "WHAT'S YOUR NAME";N$: CLS
20 COLOR RND(8)
25 PRINT@RND(512)-(LEN(N$)+2),(CHR$
                  141);N$;(CHR$ 142)
30 FOR D=0 TO 299: NEXT D
35 CLS: GOTO 20
```

```
5 REM - HI RES DEMO -
10 INPUT "WHAT'S THE BACKGROUND
              COLOR    (0,1)";B
12 IF B<0 OR B>1 THEN 10
15 INPUT "WHAT'S THE PIXEL COLOR (1-8)";P: CLS
17 IF P<1 OR P>8 THEN 15
20 MODE(1)
25 FOR X=0 TO 99
30 COLOR P,B: SET (RND(127),RND(63))
35 FOR D=0 TO 99: NEXT D
40 NEXT X
45 RUN
```

That brings us to the end of the book. We hope you've enjoyed learning about the use of your computer, and are now ready to continue on your own. Good programming.

Appendix - Glossary of Computer Words

Address - a number which refers to a location, generally in the computer's memory, where information is stored

Algorithm - the sequence of steps used to solve a problem

Alphanumeric - generally used to describe a keyboard, and signifying that the keyboard has alphabetical and numerical keys. A numeric keypad, by contrast, only has keys for the digits one to nine, with some additional keys for arithmetic operations, much like a calculator

APL - this stands for Automatic Programming Language, a language developed by Iverson in the early 1960s, which supports a large set of operators and data structures.

Application software - these are programs which are tailored for a specific task, such as word processing, or to handle mailing lists

ASCII - stands for American Standard Code for Information Exchange. This is an almost universal code for letters, numbers and symbols, which has a number between 0 and 255 assigned to each of these, such as 65 for the letter A

Assembler - this is a program which converts another program written in an assembly language (which is a computer program in which a single instruction, such as ADD, converts into a single instruction for the computer) into the language the computer uses directly

BASIC - stands for Beginner's All-purpose Symbolic Instruction Code, the most common language used on microcomputers. It is easy to learn, with many of its statements being very close to English

Baud - a measure of the speed of transfer of data. It generally stands for the number of bits (discrete units of information) per second

Benchmark - a test which is used to measure some aspect of the performance of a computer, which can be compared to the result of running a similar test on a different computer

Binary - a system of counting in which there are only two symbols, 0 and 1 (as opposed to the ordinary decimal system, in which there are ten symbols, 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9). Your computer 'thinks' in binary

Bit - an abbreviation for 'binary digit', a bit is the smallest unit a computer handles. It may have a value of 0 or 1.

Boolean Algebra - the algebra of decision-making and logic, developed by English mathematician George Boole, and at the heart of your computer's ability to make decisions

Bootstrap - a program, run into the computer when it is first turned on, which puts the computer into the state where it can accept and understand other programs

Buffer - a storage mechanism which holds input from a device such as keyboard, then releases it at a rate which the computer dictates

Bug - an error in a program

Bus - a group of electrical connections used to link a computer with an ancillary device, or another computer

Byte - the smallest group of bits (see bit) which makes up a computer word. Generally a computer is descibed as being 'eight bit' or '16 bit', meaning the word consists of a combination of eight or sixteen zeros or ones

Central Processing Unit (CPU) - the heart of the computer, where arithmetic, logic and control functions are carried out

Character code - the number in ASCII (see ASCII) which refers to a particular symbol, such as 32 for a space and 65 for the letter 'A'

COBOL - stands for Common Business Orientated Language, a standard programming language, close to English, which is used primarily for business

Compiler - a program which translates a program written in a high level (human-like) language into a machine language which the computer understands directly

Concatenate - to add (adding two strings together is known as 'concatenation')

CP/M - stands for Control Program/Microcomputer, an almost universal disk operating system developed and marketing by Digital Research, Pacific Grove, California

Data - a general term for information processed by a computer

Database - a collection of data, organised to permit rapid access by computer

Debug - to remove bugs (errors) from a program

Disk - a magnetic storage medium (further described as a 'hard disk', 'floppy disk' or even 'floppy') used to store computer information and programs. The disks resemble, to a limited extent, 45 rpm sound records, and are generally eight, five and a quarter, or three inches in diameter. Smaller 'micro-disks' are also available for some systems

Documentation - the written instructions and explanations which accompany a program

DOS - stands for Disk Operating System (and generally pronounced 'doss'), the versatile program which allows a computer to control a disk system

Dot-matrix printer - a printer which forms the letters and symbols by a collection of dots, usually on an eight by eight, or seven by five, grid

Double-density - adjective used to describe disks when recorded using a special technique which, as the name suggests, doubles the amount of storage the disk can provide

Dynamic memory - computer memory which requires constant recharging to retain its contents

EPROM - stands for Erasable Programmable Read Only Memory, a device which contains computer information in a semi-permanent form, demanding sustained exposure to ultra-violet light to erase its contents

Error messages - information from the computer to the user, sometimes consisting only of numbers or a few letters, but generally of a phrase (such as 'Out of memory') which points out a programming or operational error which has caused the computer to halt program executions

Field - A collection of characters which form a distinct group, such as an indentifying code, a name or a date; a field is generally part of a record

File - A group of related records which are processed together, such as an inventory file or a student file

Firmware - The solid components of a computer system are often called the 'hardware', the programs, in machine-readable form on disk or cassette, are called the 'software', and programs which are hard-wired into a circuit, are called 'firmware'. Firmware can be altered, to a limited extent, by software in some circumstances

Flag - this is an indicator within a program, with the 'state of the flag' (i.e. the value it holds) giving information regarding a particular condition

Floppy disk - see disk

Flowchart - a written layout of program structure and flow, using various shapes, such as a rectangle

with sloping sides for a computer action, and a diamond for a computer decision, is called a flow chart. A flowchart is generally written before any lines of program are entered into the computer

FORTRAN - a high level computerlanguage, generally used for scientific work (from FORmula TRANslation)

Gate - a computer 'component' which makes decisions, allowing the circuit to flow in one direction or another, depending on the conditions to be satisfied

GIGO - acronym for 'Garbage In Garbage Out', suggesting that if rubbish or wrong data is fed into a computer, the result of its processing of such data (the output) must also be rubbish

Global - a set of conditions which effects the entire program is called 'global', as opposed to 'local'

Graphics - a term for any output of computer which is not alphanumeric, or symbolic

Hard copy - information dumped to paper by a printer

Hardware - the solid parts of the computer (see 'software' and 'firmware')

Hexadecimal - a counting system much beloved by machine code programmers because it is closely related to the number storage methods used by computers, based on the number 16 as opposed to our 'ordinary' number system which is based on 10)

Hex pad - a keyboard, somewhat like a calculator,

which is used for direct entry of hexadecimal numbers

High-level languages - programming languages which are close to English. Low-level languages are closer to those which the computer understands. Because high-level languages have to be compiled into a form which the computer can understand before they are processed, high-level languages run more slowly than do their low-level counterparts

Input - any information which is fed into a program during execution

I/O - stands for Input/Output port, a device the computer uses to communicate with the outside world

Instruction - an element of programming code, which tells the computer to carry out a specific task. An instruction in assembler language, for example, is ADD which (as you've probably guessed) tells the computer to carry out an addition

Interpreter - converts the high-level ('human-understandable') program into a form which the computer can understand

Joystick - an analog device which feeds signal into a computer which is related to the position which the joystick is occupying; generally used in games programs

Kilobyte - the unit of memory measurement; one kilobyte (generally abbreviated as K) equals 1024 bytes

Line printer - a printer which prints a complete line of characters at one time

Low-level language - a language which is close to that used within the computer (see high-level language)

Machine language - the step below a low-level language; the language which the computer understands directly

Mainframe - the term for 'giant' computers; computers are also classed as minicomputer and microcomputer (such as the computer you own)

Memory - the device or devices used by a computer to hold information and programs being currently processed, and for the instruction set fixed within a computer which tells it how to carry out the demands of the program. There are basically two types of memory (see RAM and ROM)

Microprocessor - the 'chip' which lies at the heart of your computer. This does the 'thinking'

Modem - stands for MOdulator/DEModulator, and is a device which allows one computer to communicate with another via the telephone

Monitor - (a) a dedicated television-screen for use as a computer display unit, contains no tuning apparatus; (b) the information within a computer which enables it to understand and execute program instructions

Motherboard - a unit, generally external, which has

slots to allow additional 'boards' (circuits) to be plugged into the computer to provide facilities (such as high-resolution graphics, or 'robot control') which are not provided with the standard machine

Mouse - a control unit, slightly smaller than a box of cigarettes, which is rolled over the desk, moving an on-screen cursor in parallel to select options and make decisions within a program. 'Mouses' work either by sensing the action of their wheels, or by reading a grid pattern on the surface upon which they are moved

Network - a group of computers working in tandem

Numeric pad - a device primarily for entering numeric information into a computer, similar to a calculator

Octal - a numbering system based on eight (using the digits 0, 1, 2, 3, 4, 5, 6 and 7)

On-line - device which is under the direct control of the computer

Operating system - this is the 'big boss' program or series of programs within the computer which controls the computer's operation, doing such things as calling up routines when they are needed and assigning prioritories

Output - any data produced by the computer while it is processing, whether this data is displayed on the screen or dumped to the printer, or is used internally

Pascal - a high level language, developed in the late 1960s by Niklaus Wirth, which encourages disciplined, structured programming

Port - an output or input 'hole' in the computer, through which data is transferred

Program - the series of instructions which the computer follows to carry out a predetermined task

PILOT - a high level language, generally used to develop computer programs for education

RAM - stands for Random Access Memory, and is the memory on board the computer which holds the current program. The contents of RAM can be changed, while the contents of ROM (Read Only Memory) cannot be changed under software control

Real-time - when a computer event is progressing in line with time in the 'real world', the event is said to be occurring in real time. An example would be a program which showed the development of a colony of bacteria which developed at the same rate that such a real colony would develop. Many games, which require reactions in real time, have been developed. Most 'arcade action' programs occur in real time

Refresh - The contents of dynamic memories (see memory) must receive periodic bursts of power in order for them to maintain their contents. The signal which 'reminds' the memory of its contents is called the refresh signal

Register - a location in computer memory which holds data

Reset - a signal which returns the computer to the point it was in when first turned on

ROM - see RAM

RS-232 - a standard serial interface (defined by the Electronic Industries Association) which connects a modem and associated terminal equipment to a computer

S-100 bus - this is also a standard interface (see RS-232) made up of 100 parallel common communication lines which are used to connect circuit boards within micro-computers

SNOBOL - a high level language, developed by Bell Laboratories, which uses pattern recognition and string manipulation

Software - the program which the computer follows (see firmware)

Stack - the end point of a series of events which are accessed on a last in, first out basis

Subroutine - a block of code, or program, which is called up a number of times within another program

Syntax - as in human languages, the syntax is the structure rules which govern the use of a computer language

Systems software - sections of code which carry out administrative tasks, or assist with the writing of other programs, but which are not actually used to carry out the computer's final task

Thermal printer - a device which prints the output from the computer on heat-sensitive paper. Although thermal printers are quieter than other printers, the output is not always easy to read, nor is the used paper easy to store

Time-sharing - this term is used to refer to a large number of users, on independent terminals, making use of a single computer, which divides its time between the users in such a way that each of them appears to have the 'full attention' of the computer

Turnkey system - a computer system (generally for business use) which is ready to run when delivered, needing only the 'turn of a key' to get it working

Volatile memory - a memory device which loses its contents when the power supply is cut off (see memory, refresh, ROM and RAM)

Word processor - a dedicated computer (or a computer operating a word processing program) which gives access to an 'intelligent typewriter' with a large range of correction and adjustment features

If you've never programmed a computer before, and you want to be able to program your new Dick Smith VZ200 in just a few hours, then this book is certainly for you.

Written by VZ200 Users' Club co-ordinator Tim Hartnell, and experienced VZ programmer Neville Predebon, the book assumes no prior knowledge of computer programming. However, in just a few hours, you'll be writing worthwhile programs of your own.

From mastering the keyboard, through games playing, to making effective use of the graphics, you'll find it all here, just waiting to help you.

For a start, you'll learn just how few commands are needed to work your new VZ200. Your computer will be up and running - under your control - in less than a minute after you start reading the first page of chapter one.

ISBN 0 949772 21 6
Dick Smith catalogue number B 7206