Turtle Graphics

on the BBC Microcomputer and Acorn Electron

The Turtle Graphics program was written by Dominic Verity and this book by Barry Morrell.

Note: Within this publication, the term 'BBC' is used as an abbreviation for 'British Broadcasting Corporation'.

Turtle Graphics

on the BBC Microcomputer and Acorn Electron

About this book

The Acornsoft Turtle Graphics package provides users with the turtle drawing commands of the language LOGO. This book has been designed to take the user slowly through these commands, introducing them in a sensible order.

Turtle Graphics is the way into LOGO for most beginners, whether child or adult. The approach works for almost everyone as all the commands draw immediately on the screen what you have asked them to draw. In this way, mistakes can be quickly corrected and the final result can then be saved for future use.

About the author

Barry Morrell has worked in the computing industry since 1970. Since 1976, he has been a technical author, managing a team of authors for a national computer manufacturer before becoming a freelance writer.

With his wife, who is a teacher, the author runs a company specialising in the development of literature and software relevant to education in the home and in schools.

Acornsoft Limited, Betjeman House, 104 Hills Road, Cambridge CB2 1LQ, England.
 SLD17

Contents

Introduction

The Acornsoft Turtle Graphics package gives you the turtle drawing commands of the language LOGO. This book is designed to help you get to know them.

Turtle graphics is the way into LOGO for most beginners, whether child or adult. Through it, they get to know how to 'think' LOGO and 'think' turtle. The approach works for most people from six to ninety-six because turtle graphics commands draw *immediately* on the screen what you have asked them to draw. This means that you can try out your ideas and see at once whether they do what you intending, something interesting that you did not intend or something horrible. In the first case, you will be able to save your drawing for future use; in the other two cases you have the fun of finding out what went wrong and changing the commands to get something interesting.

Learning to use a package like this one involves spending time to become familiar with the 30-odd commands from which you can build everything else. The main purpose of this book is to take you slowly through these commands, introducing them in a sensible order. We hope that Appendix B will serve as a quick reminder of what each command does. You might find it useful to copy these pages and keep them to hand when you are turtling.

In this book we can't show you all the things that other people have drawn and found exciting. However, a lot has been published about LOGO and turtle geometry and we have provided a list of books and papers that we have found interesting, both at the fun level and the research level. We earnestly suggest that you try to get hold of some of them and see what ideas other people have had. Meanwhile, chapter 13, 'Some hints for turtlers', gives some ideas to stimulate your imagination and encourage you to invent other things for yourself.

Now, to discover the power of the turtle, look at figure 1; this is a well-known turtle graphics picture called 'SUN'. Drawing such a picture requires substantial thought, imagination and trial and error. However, the commands needed take up less than one page. When you have mastered the commands which make up turtle graphics, this and lots more are possible.

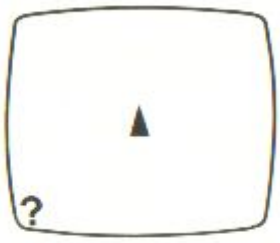
1 What are turtle graphics?

Computers are not antisocial devices. They can lead to antisocial behaviour if used improperly, but they have great potential in teaching social behaviour as well as other concepts.

If you put a small group of children around a computer they will share their experiences and learn from one another's mistakes. They will often learn to communicate with each other, as well as with the computer.

Turtle graphics has been shown to have this effect on children and it puts both the child and the computer in the right context: it puts the child in command of the computer, rather than the reverse. It is part of a computer language called LOGO. Computer languages are merely ways of telling the computer how to solve problems. There are several languages because solving problems in engineering, for example, needs a different approach from solving problems in accounting or education.

Anyway, LOGO is a computer language which helps children develop their learning ability and it uses the concept of the 'turtle'. This is a creature that the children can identify with and can move around by giving it instructions. For younger children, the turtle can take the form of a robot that moves across the floor (known as the 'floor turtle'). For older children, it takes the form of a triangle upon a television screen (the 'screen turtle'):



This is the type that is used in the Acornsoft Turtle Graphics package.

The turtle can be made to move by the children giving it instructions. As it moves, it can leave a 'trail' that is used to build up patterns as simple as a square or as complex as the children wish. In doing this, they will learn about geometry and mathematics in an enjoyable way and at their own pace. They will make mistakes and with guidance will learn to work out where things went wrong. They will also learn how to put them right.

As well as developing their critical features in this way, the children can be guided into designing something in a structured manner by relating

complicated patterns to things they already know and building them up gradually.

2 Getting under way

Turtle Graphics is designed for use on the BBC Microcomputer Model B or Acorn Electron.

You will have bought your copy of the program on either a cassette or disc. On the cassette or disc are the following files:

TURTLE	The turtle graphics system.
COVER DRAGON HILBERT	These demonstration programs are loaded from within the turtle graphics system - see Appendix C of this manual.

Please note that should you catalogue your Turtle Graphics cassette or disc, you will see a number of files that are not referred to in this book. These files port part of the turtle graphics system but should not be used (`CHAIN`ed or `LOAD`ed) directly as they are accessed automatically by the system.

The procedure for getting started depends upon which version of the program you have. In this book, instructions which apply only to the tape version of the program are shown by the cassette symbol; instructions which apply only to the disc version are shown by the disc symbol. Instructions without a symbol apply to both versions.

 You must have a cassette recorder and a television or monitor connected to your microcomputer. They should be set up as described in your microcomputer user guide.

(If your microcomputer is also connected to a disc drive you must tell it to accept instructions from the tape by typing `*TAPE` and pressing RETURN.)

Put your Turtle Graphics cassette into your cassette recorder. Make sure it is fully rewound, then type

```
CHAIN "TURTLE"
```

and press RETURN.

When you see the message 'Searching', press the PLAY button on your cassette recorder. The Acornsoft banner will be displayed in less than a

minute. After a further minute, you will see the title sequence and, after a further two to three minutes, you will see:

```
TURTLE GRAPHICS VERSION 1.0
```

A few lines further down, the character ? should appear and should be followed by a flashing underline symbol (the cursor). The screen will also contain a triangle near its centre; this triangle is the turtle and you are now ready to make it perform. However, before you go any further, stop your cassette recorder (unless you have automatic motor control, in which case it stops automatically).

You must have a disc drive and television or monitor connected to your microcomputer. The disc drive should be connected as described in the *Disc Filing System User Guide*.

Place the disc containing Turtle Graphics in the disc drive, then type

```
CHAIN "TURTLE"
```

and press RETURN.

After about three seconds you will see the Acornsoft banner, followed immediately by the title sequence. After about ten more seconds you will see this text:

```
TURTLE GRAPHICS VERSION 1.0
```

A few lines further down, the character ? should appear and should be followed by a flashing underline symbol (the cursor). The screen will also contain a triangle near its centre; this triangle is the turtle and you are now ready to make it perform.

One final point: if you want to get back to BBC BASIC at any time, press the BREAK key.

The demonstration programs are loaded from within the turtle graphics system; they are described in Appendix C.

3 Turning the turtle

The turtle on your screen performs two functions:

- It shows you its current position
- It shows you the direction in which it is pointing

You can alter both of these by giving the turtle instructions, or 'commands'. These can be typed in directly after the ? symbol (this is called a 'prompt' and it is displayed whenever the turtle is waiting for instructions).

Now try typing the instructions given below. At the end of each line, check that the instruction you have typed is what you intended and, if it is not, correct it using the DELETE key (DELETE remove the character immediately to the left of the cursor). If it is what you intended, press the RETURN key (remember, you should always press RETURN at the end of a command).

```
FD 200
LT 90
RT 90
BK 200
CLEAR
```

Your turtle should have:

- Moved up the screen, leaving a trail
- Turned left by 90 degrees.
- Turned right again by another 90 degrees.
- Moved backwards to its original position.
- Cleared the screen and returned the turtle to its original, 'home' position.

You might have noticed that the last four commands are printed at the bottom of the screen. This is to help you keep track whilst you are programming.

The commands you typed could have been input as:

```
FORWARD 200
LEFT 90
RIGHT 90
```

```
BACK 200
CLEAR
```

and this is much more readable than the earlier, shortened version. However, the latter still has its uses if you or your children prefer hitting fewer keys. Experience has shown that most people start by using the longer commands, whose meaning is immediately clear, but soon start using the 'shorthand' form.

Where a command can also be given by a two-letter shorthand, we will name the two letters by putting them in brackets after the full command name when we first define the latter. For example, `LEFT(LT)` shows that `LT` can be used as a short form of `LEFT`.

Now let us take a look at the commands in a little more detail.

`FORWARD(FD)` moves your turtle in a straight line in the direction it is facing. To tell you how far to go, you must follow the command with a number. For example

```
FORWARD 1
```

will move the turtle forward a small distance, whilst

```
FORWARD 200
```

will move it forward much further.

The turtle carries a 'pen' which it uses to draw a trail as it moves. Notice that, whilst the position of the turtle changes, its heading remains the same.

`BACK(BK)` moves your turtle backwards. Again, you must follow the command with a number. For example

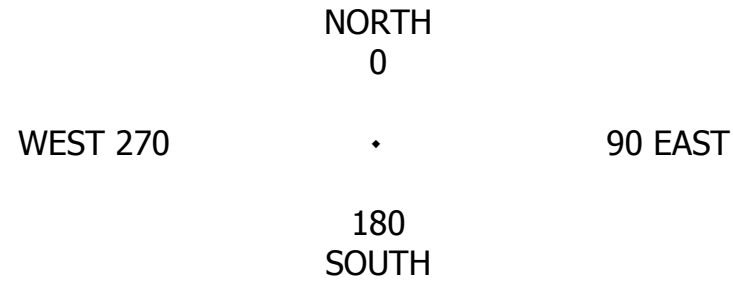
```
BACK 430
```

Moves the turtle backwards, away from the direction in which it is heading, by 430 steps. A similar effect can be achieved by using `FORWARD` with a negative number. As with the `FORWARD` command, the position of the turtle changes but its heading does not.

`RIGHT(RT)` turns the turtle clockwise (to the right) without altering its position. To tell it how far you want it to go, you must follow the command with a number, for example

```
RIGHT 55
```

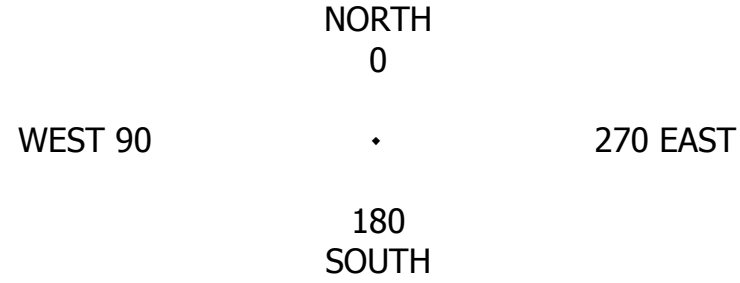
By experimenting, you will probably recognise this number as a measure (in degrees) of the angle of turn. Children will need to explore the meaning of this number and, by doing so, will understand it well. It is a good idea to ask them to try 90 degrees and its multiples. The following diagram shows the effect of different `RIGHT` turns upon the turtle:



`LEFT(LT)` turns the turtle anticlockwise (to the left) without altering its position. As with the `RIGHT` command, you use a number to specify the angle of the turn.

```
LEFT 120
```

turns your turtle 120 degrees to the left. A similar effect can be achieved by using `RIGHT` with a negative number. The following diagram shows the effect of different `LEFT` turns upon the turtle:



`CLEAR` clears the screen and returns the turtle to its 'home' position, which is at the centre of the screen, with the turtle pointing upwards.

Now try drawing some simple pictures, using the above commands. It is fun to explore them and find out what you can do; they are the basis of turtle graphics. It is also important to understand exactly how the instructions you give relate to how the turtle moves. Try to draw a square and a rectangle for a start.

What if I make a mistake?

You will make mistakes; everybody does. But it is easy to work out what you did wrong and modify the commands accordingly.

The most common mistake everyone makes is mistyping what they intended to type. For example, you might type:

```
FORWRAD 100
```

instead of

```
FORWARD 100
```

If you notice the mistake before you press the RETURN key at the end of the line, you should correct it using the DELETE key as described previously. If you do not, the program will notice the error and reply with the message:

```
Command not understood
```

You can then retype the command.

Whenever you type something wrong, or ask the program to do something it does not understand, it will reply with a warning message that tells you clearly what it thinks is wrong, relating it to the line that caused the problem. It is usually obvious how to put things right by looking at what you typed in the light of the complaint it made.

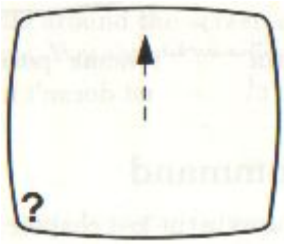
A full list of warning messages is given in Appendix A.

4 Improving your drawing

This chapter introduces you to more of the commands that you will need when you draw pictures.

The PENUP and PENDOWN commands

Up to now, whenever you have moved the turtle it will have left a trail. This can be a nuisance if you want to draw figures like the one below, without any connecting lines.



You can get round this problem by using the PENUP(PU) and PENDOWN(PD) commands. Type the following and notice what happens. Try using the shorthand commands, as well as the full ones, when you are trying out the ideas in this chapter.

```
CLEAR
FORWARD 50
PENUP
FORWARD 50
PENDOWN
FORWARD 50
PENUP
FORWARD 50
PENDOWN
FORWARD 50
PENUP
```

You should end up with a picture like the one shown below. PENUP stops the turtle from drawing whilst PENDOWN starts it drawing again.

The HIDETURTLE and SHOWTURTLE commands

If you have produced any interesting pictures up to now, you may have been irritated by the presence of the turtle sitting on your picture at the end. You can remove it by using the HIDETURTLE(HT) command, then put it back again using the SHOWTURTLE(ST) command. Try typing the following and notice what happens:

```
HIDETURTLE
LEFT 180
FORWARD 250
LEFT 180
SHOWTURTLE
```

The turtle should have returned to its 'home' position near the centre of the screen. Just because you can't see it, that doesn't mean it can't move!

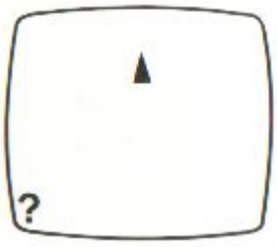
The PENERASE command

Whilst you were drawing pictures in the last chapter, you probably spoiled some of them with lines that didn't quite seem to go where you expected. You can correct these by typing

```
PENERASE
```

Then retracing over the unwanted line. PENERASE(PX) turns the 'pen' into an 'eraser'. To see how it works, type the following:

```
PENERASE
FORWARD 250
```



This should have erased the previous lines. To stop using the eraser and begin using the pen again, you need to type PENDOWN (but don't do this yet!).

The HOME command

You could return the turtle to its 'home' position by typing the following:

```
LEFT 180
FORWARD 250
LEFT 180
```

Alternatively, you could type the following:

```
HOME
```

Try doing this now. This `HOME` command returns the turtle to its home position from wherever it lies on the screen. It also makes the turtle point upwards.

If you were to move the turtle around the screen again, no trail would be left because the eraser is still in use. You should remove it now and reinstate the pen by typing

```
PENDOWN
```

`HOME`, like some other commands to come, is used infrequently so we have not provided a shorter version.

5 Putting colour into your pictures

There are two ways of putting colour into your pictures:

- Filling your diagrams with the pen colour.
- Changing the pen colour in different parts of your diagrams.

The commands you need for both of these actions are described below. These commands belong to the BBC Microcomputer and Acorn Electron; other machines may or may not have been.

The FILL command

This is a simple and useful way of improving your pictures by colouring in areas which are surrounded by a line you have previously drawn.

First, type:

```
PENUP
```

Next, move the turtle so that it is wholly within the area you want to colour. Finally, type the following two commands:

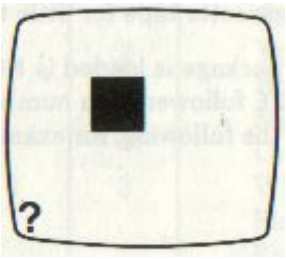
```
HIDETURTLE
FILL
```

The area will slowly be filled in with the current pen colour.

Try typing the following commands and watch what happens:

```
FORWARD 200
LEFT 90
FORWARD 200
LEFT 90
FORWARD 200
LEFT 90
FORWARD 200
LEFT 130
PENUP
FORWARD 100
HIDETURTLE
FILL
```

You should end up with a picture like the following:



Remember that if you try to fill a shape and the turtle is sitting at the end of a trail, nothing will happen. Always move the turtle into the area you wish to fill with the pen *up*. In addition, unless you want the turtle shape in the centre of your filled area, use the `HIDETURTLE` command.

If you don't wish to continue with a fill for some reason, simply press the ESCAPE key to stop filling. ESCAPE is generally used to stop a program which is running, and it returns control to the command prompt `?`.

Now, try drawing some interesting shapes and then fill them.

Changing the pen colour

The pen colours that you can use depend upon the screen mode, and this depends upon the equipment you have. Eight screen modes are available with the BBC Microcomputer and seven with the Acorn Electron, but only five are relevant to Turtle Graphics. They are summarised below:

Mode	Description
4	This uses two colours with high resolution graphics and needs 10K of memory.
5	This uses four colours with medium resolution graphics and needs 10K of memory.
0	This uses two colours with very high resolution graphics and needs 20K of memory. For this reason, it can only be used with a BBC 6502 Second Processor, or with cassette tape for little turtle programs.
1	This uses four colours with high resolution graphics and needs 20K of memory. For this reason, it can only be used with a BBC 6502 Second Processor, or with cassette tape for little turtle programs.

2 This uses 16 colours with medium resolution graphics and needs 20K of memory. For this reason, it can only be used with a BBC 6502 Second Processor, or with cassette tape for little turtle programs.

The mode in use when the package is loaded is `MODE 4`. You can change the screen mode by typing `MODE` followed by a number which corresponds to the mode you want. Try typing the following, for example:

```
MODE 5
COLOUR 2
FORWARD 200
LEFT 90
FORWARD 200
LEFT 90
FORWARD 200
LEFT 90
FORWARD 200
LEFT 130
PENUP
FORWARD 100
HIDETURTLE
FILL
```

You should end up with a square shape similar to the last one you drew, but with a yellow colour. The COLOUR command defines the pen colour, and the numbers you should put after it are defined in the following table, for each valid screen mode:

Colour numbers					Pen colour
Mode 0	Mode 1	Mode 2	Mode 4	Mode 5	
0	0	0	0	0	Black
	1	1		1	Red
		2			Green
	2	3		2	Yellow
		4			Blue
		5			Magenta (blue/red)
		6			Cyan (blue/green)
1	3	7	1	3	White
		8			Flashing black/white
		9			Flashing red/cyan
		10			Flashing green/magenta
		11			Flashing yellow/blue
		12			Flashing blue/yellow
		13			Flashing magenta/green
		14			Flashing cyan/red
		15			Flashing white/black

Now try drawing a few simple pictures in different modes and colours until you are familiar with them.

Further reading

In this chapter we have shown you how to use some of the general facilities of the BBC Microcomputer and Acorn Electron; more extensive information than we can give here can be found in the machine's User Guide. You might find it helpful to look there for information on other methods you can use to draw pictures.

6 Teaching the turtle

So far, we have been using only the commands which are built into the Turtle Graphics program; these form the turtle's basic vocabulary. However, you can extend this vocabulary yourself by 'teaching' the turtle new words.

To do this you must first name the new command which you want the turtle to obey. You must then type in the set of actions needed to carry out the command; this is called 'writing a procedure'.

A procedure is basically like all of the other commands that you have used so far, except that you create and name it yourself. We will describe the way procedures work by using one to draw a square.

You have already used the turtle to draw a square. The instructions you used were as follows:

```
FORWARD 200
LEFT 90
FORWARD 200
LEFT 90
FORWARD 200
LEFT 90
FORWARD 200
LEFT 90
```

and the turtle draws a fixed-size square with a side of 200 steps. Now it seems a bit silly for us to have to type out those eight commands each time we want to draw a square, so what we do is define a procedure called SQUARE to do it for us.

To do this we start by typing in the line:

```
TO SQUARE
```

And then press RETURN. The TO at the beginning of the line is a new command; it simply tells the program that you wish to define a procedure. In this case we are calling the procedure SQUARE, but we could have easily called it BOX by typing

```
TO BOX
```

The program now displays the following:

```
Type M for Menu or type an edit command.

Add lines
>
```

> is the editing prompt (just like ?, the command prompt). It reminds you that you are in edit mode (until now you have been in command mode). Now you can type in the commands that you wish to put into your procedure, so type in the list of commands you used to draw the square (they are listed below for you). Check each line before you press RETURN and, if you make a mistake, use DELETE to remove it. If you fail to do this and input an incorrect line, you will be able to correct it. If this happens, just type in the rest of the procedure and then look at the two short sections 'Listing procedures' and 'Replacing lines', in chapter 11 to see how to do so.

```
FORWARD 200
LEFT 90
FORWARD 200
LEFT 90
FORWARD 200
LEFT 90
FORWARD 200
LEFT 90
```

When you have typed in your commands, type the new command

```
STOP
```

This tells the computer that you have finished adding lines. There must be a STOP at the end of every procedure.

The computer will now display the message:

```
Type M for Menu or type an edit command.
```

Now, if you wish to see what your procedure looks like, type L to LIST it; your procedure will be listed for you as follows:

```
PARAMETER(S):
None
PRIVATE IDENTIFIER(S):
None

1  FORWARD 200
```

```
2  LEFT 90
3  FORWARD 200
4  LEFT 90
5  FORWARD 200
6  LEFT 90
7  FORWARD 200
8  LEFT 90
```

Don't worry about the top line; the use of parameters (you have none at present) will be described later in chapter 10, 'Using identifiers with procedures'. Here, we are beginning by learning how to define a simple procedure with no parameters.

Notice the numbers shown at the left of each line; these are to allow you to change specific lines in the procedure easily and their use is explained further in chapter 10. If there are more lines in your procedure than will fit on the screen in one go, the program will display one screenful; to look at subsequent screens, press the SHIFT key.

If you are satisfied with the procedure definition, type `E` when the computer asks for an edit command; this makes the program leave edit mode.

To test your procedure, type the following:

```
CLEAR
SQUARE
```

The turtle should automatically draw a square. `SQUARE` is now like any other command, except that you have defined it.

Now, if you wanted to, you could list your procedure when you are not in edit mode by using the LIST command. If you just type

```
LIST
```

and press RETURN, the program will display the names of the procedures currently defined. Try it, and you will see that the only procedure defined at the moment is `SQUARE`. The display you get should look like the following:

```
Procedures you've defined:-

SQUARE
```

If you type LIST followed by a procedure name, the contents of the procedure will be displayed, so type

```
LIST SQUARE
```

and press RETURN. Your procedure, SQUARE, will be displayed for you and the display should be the same as that given when you listed SQUARE from edit mode. If there are more lines to your procedure than can appear on the screen, you can display subsequent screens in the same way as you do in edit mode, ie by using the SHIFT key. TEXTSCREEN (described in chapter 12, 'Printing with the turtle') is useful when listing procedures in command mode.

Speeding things up

Sometimes a sequence of commands can take a long time to run. If you want to speed things up you can do so using the SPEED command. Try typing the following, for example:

```
SPEED 255
SQUARE
```

The number after the SPEED command determines the rate of movement and can be from 0 to 255. The program assumes a value of 230 unless you alter it using this command. The SPEED command is deliberately preset to slow so that beginners can easily follow what happens on the screen.

You can use the SPEED command in conjunction with any other turtle commands, not just with procedures, but it is most effective with procedures because that is when you are sitting back and watching the turtle perform.

7 Saving and retrieving procedures

In the previous chapter, you learned how to define procedures to save you typing commands every time you want to draw a figure. Unfortunately, this only helps you as long as the computer is switched on. Once you turn it off, you've had it; or at least your procedure has! To keep your procedures for later use you need to use the SAVE and RETRIEVE commands.

The SAVE command

This stores your procedures onto a floppy disc or cassette tape and it is very easy to use.

If you want to save your procedure onto cassette tape, you must first wind the tape you are going to use to a free space.

If you want to use a floppy disc, you should ensure that the disc does not have a write protect label on it.

Next, whether you are using disc or tape, you should type the following (but don't do it yet):

```
SAVE <filename>
```

A word of explanation is needed here to define <filename>. To give the format or general form of a command we use the convention that words in angle brackets <> stand for any instance of what those words refer to. Don't type the angle brackets; type what their contents refer to. In the case above, this will be a filename which you choose yourself. The filename can be any name of up to ten characters on tape and seven characters on disc. All of the procedures you have typed in will now be stored on the disc or tape.

Try doing this with the procedure SQUARE that you have just defined. Prepare your tape or disc as described above, then type the following:

```
SAVE PROCFIL
```

The procedure will be saved into the file PROCFIL.

The RETRIEVE command

You will, of course, want to retrieve the procedures you have saved. You can do this using the RETRIEVE command.

If you are using cassette tape, you need to rewind the tape to the beginning of the file you want to load. If you are using floppy disc, you needn't do anything special. Next, you should type the following (but don't do it yet).

```
RETRIEVE <filename>
```

<filename> is the name you used when you saved the file. If you are using cassette tape, you should now start your tape recorder running. Your procedures will then be retrieved.

Now try retrieving your procedure from PROCFIL by typing

```
RETRIEVE PROCFIL
```

then starting your cassette recorder (if you are using one).

You should now be able to save any procedures that you have written and retrieve them again at any time.

Using the *CAT command

You can check that you've saved a file as follows:

Rewind the tape and type *CAT. If you then press PLAY, this displays all the names of your files, and PROCFIL should be included.

If you are using disc, you just need to type *CAT or *. and press RETURN.

If you can't remember the name of a file that you want to load, you can check it by typing:

```
*CAT
```

*CAT is not the only operating system command that you can access from the program. You can access each of them just by typing * followed by its name as given in your microcomputer user guide.

8 Learning more about procedures

The REPEAT and ENDLOOP commands

If you look at the listing of SQUARE shown below, you will see that we are repeating the same thing four times.

```
FORWARD 200
LEFT 90
FORWARD 200
LEFT 90
FORWARD 200
LEFT 90
FORWARD 200
LEFT 90
```

You have typed in four copies of

```
FORWARD 200
LEFT 90
```

Wouldn't it be easier if you could type these commands once only, then tell the program that you want them repeated four times? Well, you can do this.

Try defining a new procedure called BOX. First type the following:

```
TO BOX
```

Next, type the following commands:

```
REPEAT 4
  FORWARD 200
  LEFT 90
ENDLOOP
```

Finally, type

```
STOP
```

then E, to get out of the editor.

The REPEAT command tells the program to repeat the lines between it and the next ENDLOOP command four times (in other words, it forms a loop). To test this, run your procedure and see what happens. Type the following:

```
CLEAR
BOX
```

The REPEAT command is followed by a number which tells the computer how many times to repeat the statements in the loop; ENDLOOP simply marks the end of the loop. The two commands are interpreted by the computer in the following way:

- REPEAT marks the top of a loop and the computer ignores it until it finds an ENDLOOP.
- If the loop has been performed the same number of times as the number after the REPEAT, the loop is left; otherwise, the lines inside the loop will be repeated.

If you use REPEAT 0 or just REPEAT, then an interesting thing happens: the REPEAT...ENDLOOP loop will not stop. Using REPEAT 0 gives a non-terminating loop. At the present state of your knowledge, you can only get out of a non-terminating loop by pressing the ESCAPE key. Later, in chapter 14, 'More advanced control of the turtle', other ways will be explained.

Getting rid of unwanted procedures

You already have one of these: it is unlikely that you will ever want to use SQUARE again. You can get rid of a procedure by using the ERASE command. This has the format:

```
ERASE <procedure-name>
```

To get rid of SQUARE, you type

```
ERASE SQUARE
```

Do not confuse ERASE with PENERASE; they do two different thing. PENERASE can remove the trail drawn by a procedure; ERASE gets rid of the procedure itself.

If you wanted to get rid of all your procedures that are held in memory, you could do so with the NEW command. You simply type

```
NEW
```

The computer will display the message:

```
THIS DESTROYS ALL PROCEDURES. SURE?
```

If you type `Y`, then all your procedures will be wiped out. Typing anything else will leave them intact.

9 Using numbers

The Turtle Graphics program has a built-in calculator and can work out the value of numeric expressions for you. Try typing the following and see what happens:

```
CLEAR
FORWARD 100+50+300-100
```

The program first calculates the value of the numeric expression.

```
100+50+300-100
```

and produces the result 350. It then uses this value as the number in the FORWARD command and moves forward 350 steps. You can prove this by typing:

```
BACK 350
```

and watching the turtle return to its home position.

Numeric expressions are not restricted to use after commands. You can, if you wish, type a numeric expression on its own. If you want to see the result of calculating the expression, precede it with a #; the answer will be displayed on the next line. Try typing the following, for example, and see what happens:

```
#100+50+300-100
```

You should get the result 350 again.

+ and – are called arithmetic operators, because they operate upon the numbers associated with them to produce a result. Some other arithmetic and logical operators are given below:

Operator	**What it does**
+	Adds the values on each side of it together
-	Subtracts the value on its right from the one on its left
-	Produces the negative of the number it is given; this is known as the unary minus, because it operates upon one number.
*	Multiplies the values on each side of it together.
/	Divides the value on its left by the one on its right.

%	Divides the value on its left by the one on its right and gives the remainder.
&	Takes the values on each side of it and produces the logical AND of the two.
|	Takes the values on each side of it and produces the logical OR of the two.
~	Gives the logical NOT of the number it is given.

The values used with these operators should be whole numbers for the arithmetic operators and truth values for the logical operators.

Don't worry if you do not understand the last two operators; their use will be described later in this book. Instead, try using a few examples of the other operators before you go any further.

A number of points need to be made here about the division (`/`) operator. When you are using this, you should remember that the answer is always 'rounded down'; for example, the following expressions all result in the answer `1`:

```
8/8
12/8
15/8
```

Also, if the result is less than one it will be given as zero. Find your yourself, by experimenting, the rules if either or both are negative.

The order of expressions

When calculating the values of expressions, the computer doesn't just go from the left of the expression to the right, calculating as it goes. Some operators are always done before others; for example, in the expression

```
10+30*5-4
```

the multiplication `30*5` is done first then the rest of the expression is worked out.

Operators are worked out in the order shown below. Those at the top have the highest priority and are worked out first.

```
*
/
%
+
```

-
~
&
|

You can make the program work out an expression in the order you want by using brackets; anything inside brackets is calculated before anything outside them. You can also put brackets inside other brackets; this is called 'nesting' brackets and those on the inside will be dealt with first. For instance, consider the rather complicated expression:

```
#(10+5(*3+(50*(30-3))
```

First of all, the program would calculate the value `10+5` (the sum enclosed by the first brackets), leaving the expression as

```
15*3+(50*(30-3))
```

Next, the contents of the innermost brackets (`30-3`) will be evaluated to give

```
15*3(50*27)
```

Finally, the contents of the remaining set of brackets is calculated giving

```
15*3+1350
```

Since multiplication always have priority over any other operation, `15*3` is now worked out and 45 is added to 1350, giving the answer 1395. Now type the expression into the computer (remembering the # sign for printing) and check the answer.

Try predicting the value of each of these expressions and then type them in to check your answers. Remember to precede each with a # so that the result is printed out.

```
10+30
10*20
34-4
45/5
10330*5-1
10+30*(5-1)
(10+30)*5-1
(10+30)*(5-1)
6/3*4
4*6/3
```

```
6%4   (remember, this is the remainder of 6 divided by 4)
30*(10/(10-5))
14/7
14/8
```

Now try some expressions of your own.

Identifiers

In turtle graphics, an identifier simply provides a way of referring to an item which can be changed at will in arithmetic expressions. You can have up to 48 identifiers and each can be up to six characters long; you can use more characters than this, but they will be ignored. Characters which you can use are the letters 'A' to 'Z', the letters 'a' to 'z' and the numbers 0 to 9; the identifier must, however, start with a letter. Identifiers which differ only in their letter case are still different identifiers. For example, '`SIDE`' and '`side`' are different.

Some words cannot be used as identifiers; these are known as 'reserved words' because they are reserved by the program to mean something specific (usually a command). For example, `LEFT` and `COLOUR` are reserved words.

Values can be assigned to the operators using the `:=` operator. For example, type the following:

```
A:=100
```

This sets the value referred to by the identifier A to 100. You can consider it as saying 'A becomes equal to 100'.

If you want to display the value to which an identifier refers, you can do this at any time by typing # before it. For example, type

```
#A
```

and see what happens. You should get the result 100.

You can use the identifier A in arithmetic expressions. For example, typing

```
#A*5
```

will give the result 500, whilst typing

```
#A*7/2
```

will give the result 350. Try these, and define a few identifiers yourself.

We will now look at how you can use identifiers to help you in your procedures. First, define the procedures 'TRI' and 'POLYTRI' using the commands given below; don't forget to use the E command when you leave the editor:

```
TO TRI
REPEAT 3
  FORWARD 300
  LEFT 120
ENDLOOP
STOP

TO POLYTRI
MODE 5
REPEAT 8
  A:=1
  REPEAT 3
    COLOUR A
    A:=A+1
    LT 15
    TRI
  ENDLOOP
ENDLOOP
STOP
```

Now run them by typing the following:

```
CLEAR
POLYTRI
```

You should get a three-colour pattern like that shown in figure 2.

There are a number of things which may be unfamiliar in this example. First of all, think about what is happening in general terms. The procedure TRI draws an equilateral triangle and it is called from within the procedure POLYTRI to draw 24 triangles, each rotated by 15 degrees from the previous one. This idea of calling one procedure from within another is new, but it should not come as too much of a surprise: any procedure you define can be regarded as a new command and it can be used in the same way as one of the original commands. Indeed, you can even call the same procedure from itself, though this is more useful with parameters, as you will see in Chapter 14, 'More advanced control of the turtle'.

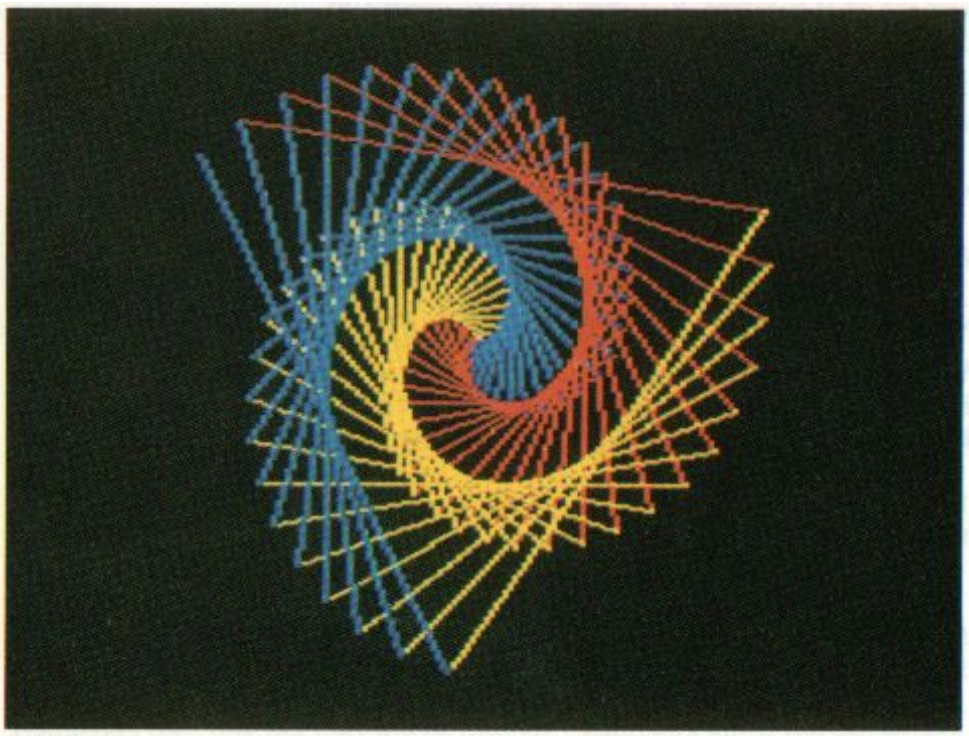


Figure 2

Next, we have two `REPEAT...ENDLOOP` loops, one within the other. What happens is that the inner loop is obeyed three times for each time the outer loop is obeyed; this gives us a total of 24 inner loops, the part which draws the triangle.

Now, think about the way the identifier A is used: it helps you to choose the colour that the 'pen' uses (in the `COLOUR` command). If you look at the table in chapter 5 you will see that there are four colours available in `MODE 5`: black, red, yellow and white. We want to draw alternate triangles in different colours, so we use the identifier A to identify the colour number (0 to 3 in `MODE 5`). However, we don't want to use colour 0, black, since this will be lose in the background colour; hence, we initially define A as being '1' and then add one to it after it is used.

Don't worry at this stage if you do not understand all of these ideas. The important thing is to remember the way the identifier A is used; you can come back to the rest later.

Now, before you go any further, get rid of the procedures `TRI` and `POLYTRI` by typing

```
NEW
```

followed by

Y

10 Using identifiers with procedures

When you define a procedure, you have the option of giving it a number of 'parameters'. These are simply identifiers which can be given values from outside the procedure when you use it. For example, suppose you wanted to draw a number of squares with sides of different sizes. You could do this by using the side of the squares as a parameter to a procedure. First of all, define the procedure by typing

```
TO SQUARE SIDE
```

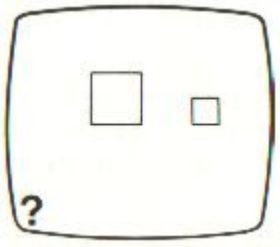
then press RETURN. Notice that the identifier `SIDE` is used as a parameter to the procedure. Now, type the following commands

```
REPEAT 4
  FORWARD SIDE
  LEFT 90
ENDLOOP
STOP
```

then press `E` to get out of the editor.

Notice that `SIDE` is used within the procedure to define the length of the sides. Suppose you wanted to draw squares with sides 200 and 100 steps long; you could do this by typing

```
CLEAR
SQUARE 200
PENUP
RIGHT 90
FORWARD 300
LEFT 90
PENDOWN
SQUARE 100
```



The values that you use when you call the procedure will be substituted within the body of the procedure when the turtle draws the squares. Try this for yourself.

```
SQUARE 10
```

is a small one; try others.

You can use as many procedure identifiers as you need, separating each one with a comma as you write it. A procedure which draws a polygon with NUMBER sides, each of length SIDE, would start as follows:

```
TO POLYGON NUMBER,SIDE
```

Each time you enter POLYGON, these parameter identifiers are set from outside, with the values given by you, so that:

```
POLYGON 3,1        gives a small triangle
POLYGON 8,12       draws a big octagon
```

The identifiers we have looked at are known as global or public identifiers; they can be used in any part of your program and will have the same meaning in each.

Procedures also have private identifiers which cannot be 'seen' from the outside but are made available for you to use each time you enter the procedure. The system gives them each an initial value of 0, but it is up to you to give them the values you want.

The following example uses some logical ideas from chapter 14, 'More advanced control of the turtle' to show the use of private identifiers. You can probably work out what they mean for yourself without looking ahead, since they mean what they say. Basically, the example draws concentric circles, changing the colour every five circles. It assumes that you already have a procedure

```
TO CIRCLE RADIUS
```

which draws circles of size `RADIUS` for you.

```
TO CIRCLES Size AND PRIVATELY Count,Col,No
Count:= Size
Col:= 0
No:= 0

\
\The above lines are not strictly necessary,
\but it is sensible to initialise the identifiers
\yourself and not rely upon the system.
\
REPEAT Size
  CIRCLE Count
  Count:= Count-1
  No:= No+1
  IF No=5
```

```
      No:=1
      Col:= Col+1
      IF Col =4
         Col:= 0
      ENDIF
      COLOUR Col
  ENDIF
ENDLOOP
STOP
```

Notice the lines which start with \ and have text on them. These are comment lines and they contain explanatory text. You can put them anywhere in your programs and they are especially useful in long programs.

You must remember that each time you call a procedure, a new set of spaces for parameter and private identifiers is created for you. Thus, you cannot use one of the private identifiers of the procedure to count how many times the procedure has been called. Instead, you must use a global or private identifier of the calling procedure. On the other hand, when you call a procedure from itself, you have a completely new set, which is just what you want. The formal rules for writing a procedure are as follows:

```
TO <procedure-name> <identifier 0>,...,
<identifiern-1> AND PRIVATELY <identifiern>,...,
<identifierm>
```

`<identifier0>,...,<identifern-1>` are the parameter identifiers of the procedure.

`<identifiern>,...,<identifierm>` are its private identifiers.

If you have no private identifiers, you need not write '`AND PRIVATELY`'.

Because the system can handle only 48 distinct identifier names, it can be very useful to have several private identifiers which happen to have the same name, but belong to different procedures and are therefore given distinct 'spaces' by the Turtle Graphics package. This is a neat trick to get round this limitation, but you do have to remember which identifier you are talking about where!

Now try typing the following:

```
LIST SQUARE
```

Assuming you haven't switched your machine off since you typed this procedure, you should get the result:

```
PARAMETER(S):
SIDE
PRIVATE IDENTIFIER(S):
None

1     REPEAT 4
2       FORWARD SIDE
3       LEFT 90
4     ENDLOOP
5     STOP

PRIVATE IDENTIFIER(S):
None
```

You should now see the significance of the top line, which we left unexplained in chapter 6, 'Teaching the turtle': it lists the parameters to the procedure.

As a final point, don't forget that you can use a procedure you have defined from within another procedure.

11 Changing your procedures

You have already looked at how you can use the editor to create a new procedure. We will now look at how it can be used to change or correct procedure and will use it to change your procedure SQUARE. First of all, enter the editor by typing

```
EDIT SQUARE
```

The following text will be displayed:

```
Type M for Menu or type and edit command.
```

Now type M to display the edit menu (shown below):

```
A to Add lines
L to List procedures
D to Delete a line
I to Insert a line
R to Replace a line
C to Change the identifier list
E to End edit
```

Most of these lines are self-explanatory. We will now explore how they work by modifying your procedure SQUARE. If you make a mistake whilst editing, you will get an error message. You can then continue from where you left off.

Listing procedures

First of all, we will list the procedure. You should always do this as the first step when you use the editor; it makes life much easier for you to have the text in front of you. You should also list your procedure after each edit operation, to ensure that you get the results you expected! Now, list your procedure by typing L; you do not need to press RETURN.

The listing consists of:

- The parameter identifiers.
- The private identifiers.
- The body of the code.

Replacing lines

Next, we will make the turtle turn to the right instead of the left by replacing the line:

```
3      LEFT 90
```

by the following line:

```
3      RIGHT 90
```

You can do this using the editor's REPLACE command. All you need to do is press R and the editor will reply with the message:

```
Replace a line
Type in line number:-
```

Next, you press the number key corresponding to the line you want replaced (3). The editor will reply with the prompt:

```
>
```

and you should type

```
RIGHT 90
```

List the procedure to ensure that your change was what you expected, then type E to edit from the editor. Now try the changed procedure by typing

```
CLEAR
SQUARE 300
```

You should end up with a square drawn on the right of the screen, instead of on the left.

Adding lines to a procedure

Now let us fill the square with the pen colour. You can do this by adding the following commands to your current procedure:

```
PENUP
RIGHT 45
FORWARD SIDE
HIDETURTLE
FILL
```

First of all get into the editor again and list the new procedure by typing

```
EDIT SQUARE
```

again, then L. You now need to use the editor's ADD command; this adds lines at the end of a procedure, and you get into it by typing A. It replies with the edit prompt

```
Add lines
>
```

and you can add the lines by typing the following, ending each line by pressing RETURN.

```
PENUP
RIGHT 45
FORWARD SIDE
HIDETURTLE
FILL
STOP
```

List the procedure to ensure that you have changed it correctly. Now try running it by typing E to get out of the editor, then

```
CLEAR
SQUARE 300
```

You should end up with a block of colour on your screen and the turtle should be hidden.

Adding procedure identifiers and inserting lines

The next thing we can do is make the turtle draw and fill your square in a different colour each time. The best way of doing this is to add a new parameter, `COL`, that will let you pass the colour number into the procedure. You will use `COL` by adding the following commands at the start of the procedure:

```
MODE 5
COLOUR COL
```

We make these changes in two stages. First, we use the editor's `C` command, to add a parameter, then we use the `I` command to insert the two new lines.

Begin by getting into the editor again; then type `C`. The following text will be displayed:

```
Change the identifier list
Type in your new identifier list:
```

You will want to type `SIDE, COL`. The new parameter will now have been added, as you will see if you list the procedure again. But it won't be much use without the two new lines inside the procedure! You can insert these without leaving the editor by typing `I`. The editor will reply with the following text:

```
Insert a line
Type in line number:-
```

You now have to type the number of the line before which you new lines are to be inserted, and you need to do this twice, because only one line can be inserted for each use of `I` command. In fact, you want to insert the following two lines before the present line one:

```
MODE 5
COLOUR COL
```

so you should reply with '`1`' then press RETURN. All the other lines will be renumbered to make space for the new line and the editor will prompt you with:

```
>
```

You should now type the line

```
COLOUR COL
```

Next you should type I again, and go through the same actions to insert the line:

```
MODE 5
```

Notice that, if you are inserting a number of lines, it is easiest to do so in reverse order, like the above. If you do it this way, you can use the same line number each time.

The lines will have been renumbered again to make space for the new line. Check that this is so by listing the procedure.

Finally, get out of edit mode by typing E and run your new procedure by typing the following:

```
CLEAR
SQUARE 300,2
```

You should see a yellow square printed out on your screen. Try using the numbers 0, 1 and 3 instead of 2 and see what results you get.

If you want to, you could now try adding a new parameter, MODE, to change the screen mode. If you would like to do this you should do it now, since, in order to demonstrate the last editor command, we are about to delete the parameter COL and the two lines we last inserted.

Deleting procedure identifiers and deleting lines

Before you delete any lines, it is always a good idea to list the procedure first. You can then type the editor command D and the editor will reply with the text:

```
Delete a line
Type in line number:-
```

You should now type the number of the line you want deleted, in our case '1', then press RETURN. You will have to go through these actions again for the other line (note that this will itself have become line number '1', since the lines will have been renumbered).

Now use the C command to delete the parameter COL by typing

```
SIDE
```

in response to the prompt:

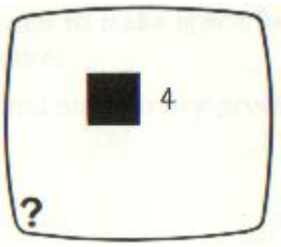
```
Type in your new identifier list:
```

Listing the procedure SQUARE will produce an identifier list with a single parameter SIDE. So in order to delete parameters you simply have to change the parameter list missing out the identifiers to be removed.

12 Printing with the turtle

You can print a message or number on the screen at the current cursor position using the PRINT command. Suppose you wanted to draw a square on the screen and print its area at the side. You could use your procedure SQUARE as the basis of this and calculate the area using an arithmetic expression. Look at the following commands:

```
TO AREA SIDE
SQUARE SIDE
PENUP
LEFT 45
FORWARD SIDE
HIDETURTLE
FILL
RIGHT 135
FORWARD SIDE
PRINT (SIDE/100)*(SIDE/100)
STOP
```



The second line calls your procedure SQUARE, the following five lines fill the square with ink. Underneath these, the lines

```
RIGHT 135
FORWARD SIDE
```

merely take the turtle out of the square and to its right. The last line,

```
PRINT (SIDE/100)*(SIDE/100)
```

prints the area of the square. Note the scaling factors of 100 that are built in. If these were not present, you could only define a square with sides up to about 180 steps, since the largest number that can be used (32767) would otherwise be exceeded.

PRINT can display either a number, as in:

```
PRINT 143
```

or it can display the value of a complicated numeric expression, built up from identifiers and numbers, as shown above. It can also display a message, as in the following line

```
PRINT “AREA IS “
```

Here, the text will be displayed as follows:

```
AREA IS
```

When you use a message in a PRINT command it must be enclosed in quotes; these are found at the top of the keyboard, above the '2' key and you get them by pressing the SHIFT key and '2' key at the same time.

Using different text windows

You may have noticed that, initially, when you typed in some text, you were able to type into only the bottom four lines of the screen. But that is not always the case; when you try to edit or define a procedure, for example, the program allows you to write on the entire screen.

In fact, you can use the screen in a number of ways. The commands that allow you to change the way in which the screen is used are SPLITSCREEN, TEXTSCREEN and FULLSCREEN.

SPLITSCREEN allows you to mix text and drawings. Text is displayed only in the bottom four lines of the screen and the four latest commands are shown; the rest of the screen is left for drawing. This is the way the system operates when switched on, and you can get back to it at any time by typing

```
SPLITSCREEN
```

TEXTSCREEN opens up the entire screen for text; it is the display used by the editor. You can change to it by typing

```
TEXTSCREEN
```

In this mode, the turtle is normally turned off but you can turn it on again by typing

```
SHOWTURTLE
```

FULLSCREEN prevents text from being printed on the screen. You can still type commands into the program as before, but the letters that are typed will not be printed on the screen. This type of display would be very useful if you wanted to take a photograph of a turtle drawing. Try using it with your procedure SQUARE by typing the following:

```
FULLSCREEN
```

```
SQUARE 300
```

Now try experimenting with these commands to see what you can do with them.

13 Some hints for turtlers

We do not presume to tell experienced turtlers how to teach. Rather, this chapter should be seen as suggestions which may help you and your children. The essence of Turtle Graphics is that you can adapt its use to fit in with your own ideas.

Group learning

It is a good idea to have two or three children working together; they can try out ideas on one another and also encourage each other. Above three, some of the group get shut out by the 'experts'; it is even worthwhile seeing that this doesn't happen with a group of three.

Learning sequence

The learning sequence used in this book can work in home or classroom teaching. It has been organised so as to introduce a few commands at a time and each group of commands can open a new horizon for the children.

Children should first be allowed to explore the screen for themselves. By trial and error they can work out the size of the screen; they can also find out what happens when they go 'off' the screen. Then, they can go on to explore the idea of angles and can be guided into drawing a square. You can help them to discover the movements needed by getting them to move their bodies in the same way as they want the turtle to move on the screen. This is known as 'playing turtle'.

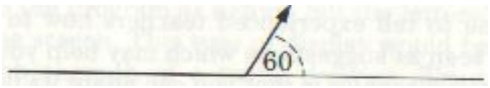
At this point you could introduce the concept of colour and that will give the children a new creative element to play with.

The commands already introduced give the children plenty of scope; when you feel that they are tired of typing in the same instructions a number of times, you could show them how to use procedures and the `REPEAT...ENDLOOP` command.

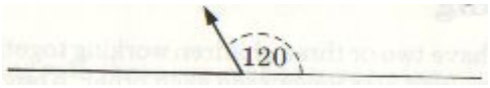
You will now have introduced them to the basis of programming and one of the most important ideas to get over at this point is the importance of designing programs. They should think about what they are going to do before they do it, and consider what the outcome will be. If things go

wrong, they will then have something to measure their progress against and assess what went wrong.

At this point you could get them to draw an equilateral triangle and this will probably lead to their first major programming 'bug'. The children may know that an equilateral triangle has angles of 60 degrees and they will probably draw something like the following:



This is because they should be drawing the 'outside' angle rather than the 'inside' angle. To produce an equilateral triangle they will have to turn though 120 degrees from the direction in which they are travelling:



When you introduce the concept of parameters you open up a lot of possibilities. You could, for example, get them to produce a number of different-sized equilateral triangles.

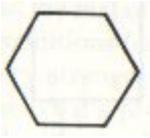
```
TO TRI SIDE
REPEAT 3
  LEFT 120
  FORWARD SIDE
ENDLOOP
HIDETURTLE
STOP
```

A point to bring out here is that the turtle turned through 360 degrees min both the triangle and the square. In fact, if the turtle goes round the boundary of an area and ends up in the position where it started, the sum of all turns will be 360 degrees. This is what is known as the *Total Turtle Trip Theorem*. If the angles at each corner of the area are similar, each angle is, thus, 360 degrees divided by the number of sides. For example:

360/4 = 90 degrees for a square
360/3 = 120 degrees for an equilateral triangle

The children could then extend this discovery to produce polygons with any number of sides. For example, a hexagon has six sides and angles of 60 degrees. Its procedure could be:

```
TO HEX SIDE
REPEAT
  LEFT 60
  FORWARD SIDE
ENDLOOP
HIDETURTLE
STOP
```



Alternatively, they could be guided into producing the following procedure, POLY. This simply moves the turtle forward a constant distance, turns it left a constant angle and repeats the procedure again. The angle of turn and the distance to move are given to POLY as parameters.

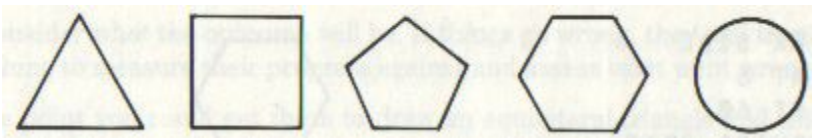
```
TO POLY SIDE,ANGLE
REPEAT
  FORWARD SIDE
  LEFT ANGLE
ENDLOOP
STOP
```

With this procedure you would have to use the ESCAPE key to stop the turtle, because it runs indefinitely.

The next logical step is to remind them of the relationship between the number of sides and internal angles, then get them to write a procedure to produce the angle automatically and stop after the given number of sides:

```
TO NUPOLY SIDE,NUMBER
REPEAT NUMBER
  LEFT 360/NUMBER
  FORWARD SIDE
ENDLOOP
STOP
```

The pictures below show some examples of polygons. If the children are guided into producing polygons with an increasing number of sides they might notice that they are getting closer to a circle. This is finally reached when the number of sides (NUMBER) equals 360.

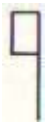


One idea you may be familiar with is Piaget's: that younger children more readily identify with concrete rather than abstract concepts. For example, at a very early stage you could get them to draw houses, using squares

and triangles. In fact, you may have difficulty in dragging them away from doing this and getting they to draw, say, hexagons! If you could introduce the procedure for a circle fairly soon, you could widen their scope: you could get them to draw an Eskimo's igloo and then talk about geography for a while.

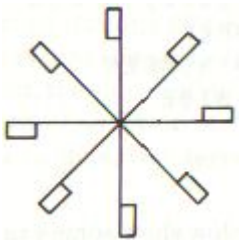
The idea of using one procedure inside another is a useful one. For example, have a look at the following:

```
TO PADDLE
FORWARD 200
REPEAT 3
  LEFT 90
  FORWARD 50
ENDLOOP
LEFT 90
BACK 150
STOP
```



This produces the paddle shape shown on the right. It could be used to do a picture of a paddle wheel:

```
TO WHEEL
REPEAT 8
  PADDLE
  RIGHT 45
ENDLOOP
HIDETURTLE
STOP
```



You could now talk about paddle steamers, for example, and this might lead to them thinking of new shapes.

With this paddle wheel example you are also introducing structured programming, an idea that is well worth developing. It involves dividing up the program into natural parts (in the present case, `PADDLE` and the rest of `WHEEL`) so that each part can be debugged separately. The worst conditions for debugging are when several bugs are present simultaneously; very strange results can often be achieved. The debugging process is most effective if a program is divided into small parts, or procedures, and each procedure is small enough for it to be unlikely that it contains more than one bug.

In fact, the method of structured programming described above is called the 'bottom up' approach, because we are effectively designing a shape then looking to see what other, more complicated shapes can be built from it. There is another, more common method called the 'top down'

approach; this involves looking at a problem and breaking it down into smaller, distinct units. Both methods have their uses, as you will find out for yourself.

Now try using POLYSPI. As its name implies, it is simply a derivative of the procedure POLY which was defined above. In this case, the angle of turning stays constant but the distance to move through, which we called SIDE, is changed.

```
TO POLYSPI SIDE,ANGLE
REPEAT
  FORWARD SIDE
  LEFT ANGLE
  SIDE:=SIDE+20
ENDLOOP
STOP
```

Type this in and try it out.

Now, in the definition of POLYSPI above, the amount by which SIDE is incremented each time is always 20. It might be useful if you could change this increment and you could do this by adding a new parameter to POLYSPI's parameter list. Try calling this INC and then replace '20' by INC in the fourth line. When you call POLYSPI with various parameters the range of results is quite marvellous.

By this point you and the children will probably have invented some new procedures and found several avenues to explore. This is the most rewarding part of turtle graphics: you are very often 'breaking new ground'. The areas that have been covered give you a lot of scope for teaching, whether at home or in the classroom. However, don't stop there. When you are ready, try looking through chapter 14, 'More advanced control of the turtle'. This will introduce you to some of the most powerful items in the turtle's vocabulary.

14 More advanced control of the turtle

Many people will, for quite a time, use only the commands which have been described up to the present. However, there are other commands available that help you to exploit the facilities of turtle graphics and your microcomputer to the full. They are described below.

Recursion

Once concept which has something of an aura of mystery about it is recursion, probably because it is about the only word in the turtle vocabulary which is not familiar from everyday use. However, it is both simple and fascinating. All that it involves is writing a procedure which calls the same procedure, usually with another parameter, to do part of its work.

Such a procedure could, of course, go on for ever, and there is little point in writing a procedure like SQUARE in this way. Recursion is most effective when it is used together with an arithmetic operation on one or more identifiers. A square which redraws itself continually is dull and valueless; a square which grows, or which tilts a few degrees each time it is drawn, is quite another thing. In fact, recursion can offer a very wide range of teaching ideas in itself.

One example of the use of recursion is the ability to replace REPEAT...ENDLOOP loops with procedures that call themselves. In other words, a procedure which employs recursion calls another version of itself as a sub-procedure.

In the previous chapter, we defined the procedure POLYSPI. This can be defined in another, perhaps cleaner, way using recursion:

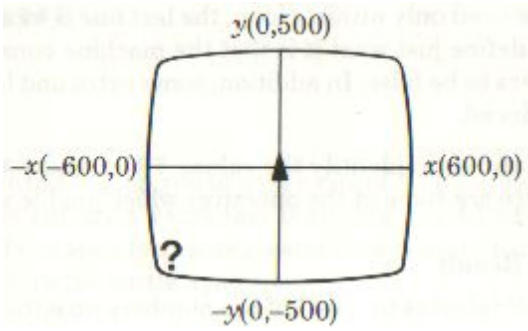
```
TO POLYSPI SIDE,ANGLE,INC
FORWARD SIDE
LEFT ANGLE
POLYSPI SIDE+INC,ANGLE,INC
STOP
```

The last line but one keeps the process repeating by including the command to call POLYSPI as part of POLYSPI's definition.

Try this new version of POLYSPI and compare it with the original.

SETTURTLE and TURTLESTATE

These commands allow you to check the position of the turtle and the direction in which it is pointing, and to alter these values. The home position is regarded as the origin, with *x*- and *y*-coordinates of zero. The initial direction of the turtle is regarded as 0 and it increases in a clockwise direction; for example, when it points downwards it has a direction of 180.



You can check the state of the turtle using the TURTLESTATE(TS) command. For example

```
TURTLESTATE A,X,Y
```

will return the direction of the turtle and its *x*- and *y*-coordinates in A, X and Y respectively. You can use other identifiers, instead of these, if you wish. Try moving the turtle around and use the command a few times.

SETTURTLE(STP) is the reverse of this command; it allows you to move the turtle to an exact location on your screen.

```
A:=45
X:=100
Y:=100
SETTURTLE A,X,Y
```

will move the turtle 45 degrees to the right. Try it and see.

A use for these commands will be shown in the next section.

Logical operations

Some BASIC-like facilities have been included to let you check if an expression is true or not, and then act upon the result. The commands available are as follows:

```
WHILE <logexpr>
UNTIL <logexpr>
IF <logexpr>...ENDIF
```

The first two can be used only within a loop; the last one is more general. First, though, we should define just what it is that the machine considers to be true and what it considers to be false. In addition, some extra and logical operators must now be introduced.

We can use the computer to identify the values TRUE and FALSE, as well as numeric values. Here are some of the operators which enable you to do so:

Operator	Result
=	Evaluates to TRUE if the numbers on either side of it are equal, otherwise it evaluates to FALSE.
<	Evaluates to TRUE if the number on the right is greater than the one on the left, otherwise it evaluates to FALSE.
>	Evaluates to TRUE if the number on the left is greater than the one on the right, otherwise it evaluates to FALSE.
&	Logical AND; evaluates to TRUE if and only if the expressions on both sides of the & symbol are TRUE.
|	Logical OR; evaluates to TRUE if one of the expressions that surrounds the | symbol is TRUE.
~	As we said earlier, when we were talking about number operators, ~ works as NOT with truth values.

It is often important to be able to compare the sizes of identifiers, and these functions could come in handy. Note that <, > and = are always worked out before & and | (except where brackets are used) but are always done *after* all of the other operators.

The WHILE and UNTIL operators

Until now, you have only been able to get out of loops using the ESCAPE key. The other way of doing this is by using either the WHILE or the UNTIL command. Try defining the following procedure, which includes the WHILE command:

```
TO SQUARE SIDE
REPEAT 4
  WHILE SIDE<350
  FORWARD SIDE
  LEFT 90
ENDLOOP
STOP
```

Here, WHILE is used to stop you using a value of SIDE greater than or equal to 350. WHILE you specify a value less than this, the REPEAT...ENDLOOP runs as normal; if you specify a value greater than or equal to 350, an exit is made from the loop before the turtle moves.

You could use UNTIL in a similar way, as shown below with another procedure:

```
TO ENDSPIRAL ANGLE AND PRIVATELY SIDE,A,X,Y
SIDE:= 0
REPEAT
  TURTLESTATE A,X,Y
  UNTIL X>500 | X<-500 | Y>500 | Y<-500
  FORWARD SIDE
  RIGHT ANGLE
  SIDE:= SIDE+10
ENDLOOP
STOP
```

This produces a spiral polygon, as in chapter 13, 'Some hints for turtlers'. UNTIL SIDE becomes equal to 500, the loop runs as normal; once it becomes larger than that value, an exit is made.

The IF command

The command

```
IF <logexpr>
```

is followed by any number of lines of commands which are to be performed only if <logexpr> is true. The lines that are to be performed conditionally are completed by the ENDIF statement. If <logexpr> is false, the flow of control of the program will go to the largest enclosing ENDIF statement; that is, the conditional statements will be left out.

Below is a procedure that uses IF...ENDIF expressions and the TURTLESTATE command mentioned earlier. It is called CHECKFORWARD. Basically, it takes a single parameter, which gives a distance to move forward, and checks to see if moving the turtle that distance would take it off the screen (this is considered to be a 1000 x 1200 box centred on the origin). If it does, the procedure moves the turtle back to its old position, otherwise, it plots the line.

```
TO CHECKFORWARD SIDE
TURTLESTATE B,C,D
PENUP
HIDETURTLE
FORWARD SIDE
TURTLESTATE A,X,Y
SETTURTLE B,C,D
PENDOWN
SHOWTURTLE
IF X>500 |X<-500 |Y>500 |Y<-500
  FORWARD SIDE
ENDIF
STOP
```

If you want to type in the program, do so and try to think about how it works before you try running it.

The RAND command

The RAND command will return at random a numeric value in a named identifier. For example

```
RAND A
```

Will return a number from –32768 to 32767, inclusive, in A.

The command has the formal definition:

```
RAND <identifier>
```

Look at the following procedure, RANCOL:

```
TO RANCOL
MODE 5
RAND COL
COL:=(COL & 32767) % 3+1
COLOUR COL
REPEAT 3
  FORWARD 300
  LEFT 120
ENDLOOP
STOP
```

This draws a triangle of side 300 steps, but before it starts to draw, the ink colour is chosen at random. Work out for yourself what is happening, then try it out.

The GET command

The `GET` command is an input command and it will wait for a key to be pressed whilst the procedure is running. For example

```
GET A
```

Will place the value of the key which you press in the identifier A. Look at the following procedure, `TRICOL`:

```
TO TRICOL
MODE 5
GET COL
COLOUR COL
REPEAT 3
  FORWARD 300
  LEFT 120
ENDLOOP
STOP
```

This also draws a triangle of side 300 steps, but before it starts to draw it waits for you to press a key. If you press a number key, it will draw the triangle in the colour corresponding to that number for `MODE 5`. Try it and see.

Producing special effects

The BBC Microcomputer and Acorn Electron can produce very sophisticated sound effects when running Turtle Graphics by using the SOUND and ENVELOPE commands. They can also produce impressive visual effects using the VDU command.

A description of the full potential of these commands is outside the scope of this book. Instead, a brief outline of the SOUND command is given below, together with a simple example of its use, so that you can see what can be done. Further information can be obtained from the User Guide. If you are interested in this subject and want to know more, see the books listed in the Bibliography.

The SOUND command

There are three elements to a simple sound:

- Its frequency
- Its amplitude, or loudness
- Its duration

You can define each of these using the SOUND command. This has the following format:

```
SOUND <channel no>,<amplitude>,<pitch no>,<duration>
```

There are four channels (numbered 0 to 3) and all can be on at the same time. For normal tones, channels 1 to 3 are used; 0 is used for special effects.

The amplitude can have a value from –15 to 4, but the positive values have special uses.

The pitch number can have any value between 0 and 255 and the duration can have any value between –1 and 254 (measured in twentieths of a second). A duration value of –1 means 'sound continuously'.

The following procedure shows how you can use the SOUND command. It generates the notes C, E (almost) and G as each side of a triangle is drawn.

```
TO TRI SIDE
A:=53
  SOUND 1,-12,A,10
  FORWARD SIDE
  LEFT 120
  A:=A+14
ENDLOOP
STOP
```

Try it out and see what happens.

Appendix A

Warning messages

`Division by zero`	You have attempted to divide a number by 0.
`Bad calculation`	You have used a bad numeric expression.
`Number too big`	The result of an expression is too large for the turtle to cope with.
`Too many brackets`	The brackets in a numeric expression are nested to a depth of more than 18.
`Run out of memory`	You have tried to use `FILL` when there is not enough memory to do so.
`Bad parameter list`	The parameter list after the `TO` definition is not quite correct.
`Not got enough memory`	You have tried to add a procedure or procedure line which is larger than spare memory.
`Procedure name already used`	The procedure name after `TO` has been used before.
`Bad line number`	You tried to replace a line that didn't exist in a procedure.
`Command not understood`	You have typed in a command that the turtle doesn't understand.
`Escape`	You have pressed the ESCAPE key.
`IF without an ENDIF`	Your `IF` statement is not followed by an `ENDIF`.

`Bad mode`	You are not allowed to use that display mode.
`REPEAT without an ENDLOOP`	Your `REPEAT` statement is not followed by an `ENDLOOP`.
`ENDLOOP without a REPEAT`	An `ENDLOOP` has appeared without a preceding `REPEAT`.
`WHILE/UNTIL without a loop`	A `WHILE` or `UNTIL` statement has been used outside a loop.
`Bad procedure name`	The procedure name after the `TO` expression contains a numeric operator or begins with a command name.
`Bad variable list`	There was a mistake in the list of identifiers following a `GET` or `TURTLESTATE` command.
`Illegal REPEAT command`	A `REPEAT` was followed by an illegal expression, e.g. `-4`
`Too many identifiers`	The upper limit of 48 identifiers has been exceeded in your program.
`Too many REPEATs/ procedure calls`	There is no more memory left in the computer to deal with a new `REPEAT` or procedure call.
`Undefined identifier`	You have tried to use an identifier which isn't a parameter, private variable or global.
`Wrong number of parameters`	A procedure was called with more or fewer parameters that mentioned in its `TO` line.

Appendix B

Summary of commands

Basic movements

`FORWARD(FD)<distance>`	Move forward
`BACK(BK)<distance>`	Move backwards
`LEFT(LT)<angle>`	Turn left
`RIGHT(RT)<angle>`	Turn right
`HOME`	Move to home position
`CLEAR`	Clear the screen and home turtle
`SPEED`	Change speed of turtle movement

Drawing and printing commands

`MODE`	Changes screen mode
`PENUP(PU)`	Lift pen
`PENDOWN(PD)`	Restore pen
`HIDETURTLE(HT)`	Remove turtle from screen
`SHOWTURTLE(ST)`	Restore turtle
`PENERASE(PX)`	Replace 'pen' with 'eraser'
`FILL`	Fill area with pen colour
`COLOUR <colour number>`	Change pen colour
`PRINT  <expression>`	Prints a number or text
`SPLITSCREEN`	Changes text window (text on lower four lines)
`TEXTSCREEN`	Changes text window (text on entire screen)
`FULLSCREEN`	Changes text window (no text displayed)

Procedures

Defining procedures

```
TO <procedure-name> <identifier0>,...,
<identifiern-1> AND PRIVATELY <identifiern>,...,
<identifierm>
```

Saving procedures

```
SAVE <filename>
```

Retrieving procedures

```
RETRIEVE <filename>
```

Commands used with procedures

`EDIT <procedure-name>`	Enters the editor
`ERASE <procedure-name>`	Erases procedure
`NEW`	Erases all procedures in memory
`LIST`	Lists names of all procedures
`LIST <procedure-name>`	Lists contents of a procedure

Logical operations

`IF <logical-expression>`
`..........`
`..........` commands
`..........`
`ENDIF`
	Commands will be obeyed as long as the logical expression is `TRUE`

`WHILE <logical-expression>`	The `REPEAT...ENDLOOP` loop will be executed while the logical expression is `TRUE`
`UNTIL <logical-expression>`	The `REPEAT...ENDLOOP` loop will be executed until the logical expression is `TRUE`

Additional commands

TURTLESTATE(TS)A,X,Y	Gives the position and heading of the turtle in the identifiers named
SETTURTLE(STP)A,X,Y	Allows you to set the turtle's position and heading. A, X and Y are expressions.
SOUND	BBC BASIC SOUND command
ENVELOPE	BBC BASIC ENVELOPE command
GET	Waits for key to be pressed and returns value
RAND <identifier>	Returns a pseudo random number in identifiers
VDU	BBC BASIC VDU command

Appendix C

The demonstration programs

On the cassette or disc you received with your Turtle Graphics package, there should be three example programs. These are:

COVER
DRAGON
HILBERT

This Appendix describes each of the programs in turn.

To load the programs, follow the instructions given in chapter 7, 'Saving and retrieving procedures'. If you are using tape, it is probably easiest to go through the programs in the order given above, as this will save you having to search through the cassette to find the files.

COVER – The design on the cover of the pack

On loading and the listing the file 'COVER'; you will see that there is a procedure called `POLSPI`. This stands for poly-spiral. The procedure requires no parameters, but uses two private identifiers, `L` and `N`. These are used to store the current length of the line drawn by the turtle, and the ink colour in which the line is drawn.

The procedure uses (at line 2) the `VDU` command. In this example, it is being used to change the colour produced by ink colour 3 (which is usually white) to blue. See the User Guide for more information about the various types of `VDU` command; they are all accessible from Turtle Graphics.

Notice the way in which `IF` is used to make `N` cycle through the values 1, 2, 3, 1, 2... and so on. Can you change the procedure so that it uses the remainder operator `%` instead?

DRAGON – The dragon curve

The next two programs contain examples of using recursion in procedures. The dragon curve is a mathematical object which can look very beautiful if the right numbers are used. When the file 'DRAGON' has loaded, there will

be three procedures in the computer: LDRAGON, RDRAGON and DRAGON. The first two procedures do all the work; DRAGON just calls them to produce a particular pattern. Thus, to see a typical dragon curve, just type

```
DRAGON
```

and press RETURN after loading the file.

LDRAGON and RDRAGON are what is known as mutually recursive procedures. This is because LDRAGON uses RDRAGON to do its job, and in turn, RDRAGON also calls LDRAGON. To experiment with different dragon curves, you must call RDRAGON with different parameters. It takes two of them: the step length by which the turtle must be moved, and an 'order'. The higher the order, the more complex the pattern appears. For example, a very simple dragon curve might be:

```
RDRAGON 32,2
```

and a more complex one is:

```
RDRAGON 8,10
```

After experimenting you will see that the higher the order (the second parameter), the smaller the step must be to keep the turtle on screen.

HILBERT – The Hilbert curve

The Hilbert curve is like the dragon curve in that it uses recursive procedures, but the patterns obtained are very different. If you load the file 'HILBERT', there will be two procedures in the computer. These are HILBERT and HIL. To see the effect of Hilbert curves immediately, just type

```
HIL
```

and press RETURN.

This procedure calls HILBERT with specific parameters to obtain the pattern on the screen. Like RDRAGON, HILBERT requires a step and an order. It also needs a third parameter, which must always be set to 1. Some examples of simple and complex Hilbert curves for you to try are:

```
HILBERT 32,4,1
HILBERT 8,6,1
```

Bibliography

Turtle graphics

ABELSON, Harold
DiSESSA, Andrea
Turtle Geometry: The Computer as a Medium for Exploring Mathematics (MIT Press)

ABELSON, Harold
LOGO for the Apple II (BYTE/McGraw-Hill)

PAPERT, Seymour
Mindstorms: Children, Computers and Powerful Ideas (The Harvester Press)

Special effects

McGREGOR, Jim
WATT, Alan
The BBC Micro Book, BASIC Sound and Graphics (Addison-Wesley)

McGREGOR, Jim
The Electron Book, BASIC Sound and Graphics (Addison-Wesley)

Index