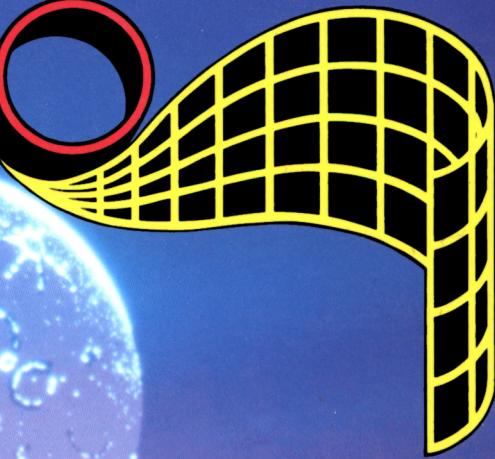


LOGO



USING DR LOGO ON THE AMSTRAD



GLENTOP
COMPUTER BOOKS

by MARTIN SIMS

USING DR LOGO ON THE AMSTRAD

USING

DR LOGO

ON THE

AMSTRAD

by

Martin Sims

Glentop Publishers Ltd

NOVEMBER 1985

All programs in this book have been written expressly to illustrate specific teaching points. They are not warranted as being suitable for any particular application. Every care has been taken in the writing and presentation of this book but no responsibility is assumed by the author or publishers for any errors or omissions contained herein.

COPYRIGHT © Glentop Publishers Ltd 1985
 World rights reserved.

No part of this publication may be copied, transmitted or stored in a retrieval system or reproduced in any way including but not limited to photography, photocopy, magnetic or other recording means, without prior permission from the publishers, with the exception of material entered and executed on a computer system for the reader's own use.

ISBN 0 907792 56 1

Published by:

Glentop Publishers Ltd
Standfast House
Bath Place
High Street
Barnet
Herts EN5 1ED
Tel: 01-441-4130

Contents

Learning Logo

Chapter 1

Introduces the reader to the language Logo and the computer. The reader gains confidence by means of the immediate responses obtained via the keyboard and screen. The main objective of this chapter is to help the reader gain control of Logo and the graphics turtle by means of commands entered in direct mode.

Chapter 2

Having mastered some fundamental commands, the reader is now equipped with the ability to use procedures (programs). Readers are encouraged to 'nest' these procedures as subroutines. The need for planning and forethought in the design of procedures is established. The concept of angular rotation is introduced and Logo's ability to use both text and graphics acknowledged. Recursive activities are introduced with the use of the 'REPEAT' command.

Chapter 3

In this chapter, greater flexibility and innovation are encouraged with the introduction of further commands. These include the tools to enable the reader to edit previously defined procedures, allowing them to be used with increased freedom.

Chapter 4

Greater use is made of Logo's ability to use both text and graphics. Readers are also invited to learn about the analysis and creation of circles and curves.

Chapter 5

The reader is shown how to position the turtle at an exactly defined position on the screen. The use of absolute as opposed to relative commands is described. Use is made of Logo's ability to incorporate scale factors (passing parameters) within procedures. Also, the reader is shown how to take full advantage of the disc system and file editing.

Chapter 6

This chapter introduces random variables and the use and control of variables in general. Colour and sound are used. The reader is also shown how to use conditional testing to allow procedures to make simple decisions. Advantage is taken of Logo's ability to use recursion.

Chapter 7

Readers are involved in the logical representation and planning of procedural requirements by the development of two interactive games. These involve numeric inputs from the reader, and some limited text processing.

Chapter 8

In this chapter, use is made of Logo's arithmetical abilities to produce a simple tables-testing game.

Chapter 9

Use is made of previously encountered techniques to produce an animated clock, with chimes. Throughout the chapter, readers will be experimenting with a 'real time' application.

Chapter 10

This chapter introduces new commands to improve text handling, allowing random sentence and poetry generation.

Chapter 11

In this chapter nested procedures are used to form an interactive game of logic. Readers are also introduced to the idea of manually executing programs in order to find faults.

Solutions

These are the solutions to the various projects and exercises suggested in the text designed to reinforce newly learned material

Index

Learning Logo

This course is suitable for users of the Amstrad **CPC 464** and **664** computers using the DR LOGO provided free with the Amstrad disc drives. It also covers the Amstrad **CPC6128** computer using LOGO2 with CP/M 2.2, provided free with the computer.

Modern computers perform many complex tasks. They can land aeroplanes, control spacecraft, drive machine tools, create music and tell a driver when the engine oil in a car is low. They are an important part of modern life, and can no longer be ignored. Proof of this is the fact that you are using one and are about to embark upon the very special task of teaching yourself or your children to use, program and enjoy learning through a computer language called **LOGO**.

As well as learning to use and enjoy using your computer, concepts will be developed and extended in many ways as basic mathematical knowledge and concepts of length, area, direction and volume are explored. This book is also designed for adults who, with no computing experience, are looking for a way into the world of computer programming.

To make a computer do something, however complex or simple, it must be told what to do. This sequence of commands must describe every step in a logical way that the computer can understand and must do so in a language the computer understands.

The language home computers normally understand is called BASIC which is adequate for beginners and enthusiastic amateurs.

Many different languages are available for use with computers, some more sophisticated than others, some for specialised use and others for general-purpose use. As most microcomputers are sold with the BASIC language built in, many beginners use this to communicate with a computer for the first time. LOGO is unique, however, as it is the first powerful language designed specifically for the beginner.

It has been designed as an introductory language for everyone regardless of age or academic ability, to use and enjoy. It enables the worlds of mathematics, physics, art, linguistics and strategic approaches to thinking to be glimpsed while exploring fundamental aspects of computer programming.

Each chapter of this book outlines a new stage of learning and describes further projects to be carried out on your computer. The aims of each chapter are outlined at the beginning of the book and there will be plenty of exercises and 'things to do' at the end of each of the earlier chapters. As you read this book, you will be learning a computer language called LOGO whilst developing educational concepts which will become the cornerstone of future learning development.

A major characteristic of LOGO is that it allows the novice to create complicated procedures from a series of simple steps, with immediate results. The core of LOGO is its use of GRAPHICS. Pictures appeal to children and this package offers an enormous range of audio-visual activities which will delight both children and their parents. The central 'character' of the language is a TURTLE which is the pen-point of the graphics-based learning system. 'Turtle' was the name given to the pen-point by the Massachusetts Institute of Technology Artificial Intelligence Laboratory in honour of cybernetic animals called turtles made by Walter Grey, an English neurophysiologist. By using familiar concepts like 'forward' and 'back', 'left' and 'right', children can draw on the screen and via 'TURTLE GRAPHICS' start using LOGO and their computer. Research demonstrates that this approach has a great deal to offer in terms of educational development. If this is the case, both parents and children will benefit from their romance with LOGO.

Throughout this introductory guide to Logo a large proportion of the available syntax is considered. The user, having read this book, will be in a position to make good use of the full language as described in the sourcebook provided with the Logo. A further book in this series will cover the more advanced features of Logo.

Chapter 1

The computer usually understands a language called BASIC. If we are to use a new language, it must be loaded into the computer whenever it is to be used.

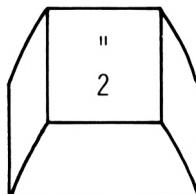
The Logo language is contained on the disc accompanying the machine and should be loaded in the usual way. If you do not know how to do this, you should consult your machine's manual or follow the procedure called 'starting off' which follows the introduction to the keyboard below.

Introduction to the Keyboard

Just in case you're not familiar with the major keyboard characteristics, they are reviewed here. Other keyboard functions will be introduced as required throughout the book. If you are already conversant with the computer's keyboard then you should skim rapidly through this section.

The keyboard is similar to that of a typewriter and can be used in the same way. For the moment you can ignore the blue 'ENTER' key and the 17 keys on the right with arrows and numbers on them and simply concentrate on the dark grey keys that make up the main keyboard.

Some of the keys have two symbols on their top face, for example the number '2' key looks like this:



In order to type a '2', you simply tap the key once and '2' will appear on the screen. Hit it a few times and see what happens. Try holding the key down and you will see that it 'repeats': you don't get that sort of result with a mechanical typewriter! If you hold it down long enough the computer will begin to fill up the screen. Hold down the light green 'DELETE' button in the top right hand corner of the keyboard to remove all these twos. The 'DELETE' key is useful for correcting typing mistakes! Of course you don't have to DELETE the full line when you make a mistake, just the offending letters.

Now, let's suppose you wanted to type the character shown on the key above the '2', i.e. the inverted commas. Well, just like a typewriter, you have to hold down SHIFT with one hand and hit the '2' key with the other.

Try holding down the '2' key and just tap one of the SHIFT keys. Then DELETE them all, please.

Throughout the introductory chapters of this book, all instructions to be typed in at the keyboard will be surrounded by single quotation marks ('') when included within the body of the text. Things like <ENTER> denote that you should hit the key named – in this case, the <ENTER> key at the right of the main keyboard. If you need to type a space, hit the long key at the bottom of the keyboard and this will produce one space.

For example: 'NEW <ENTER>' means type in the three letters N, E and W and then hit the ENTER key. The ENTER key lets the computer know that you've finished typing the line in. The computer has to be told this because lines can be much longer than simply 'NEW'. If the items to be typed in are not within the body of the text, but on separate lines of their own (in a slightly larger typeface) then the single quotes are unnecessary and will be left out. For example

NEW <ENTER>

Starting Off (Disc Loading)

If you have a CPC6128, see Appendix 1 instead.

Your DR Logo program is on side 2 of the System/utilities disc. To load Logo you must insert the disc with side 2 upwards and then press:

CTRL – SHIFT – ESC

This means you must hold down the CTRL key and a SHIFT KEY, hit ESC briefly, (top left of keyboard) with your other hand and then release the CTRL and SHIFT keys. The disc operating system is now automatically set up. Now to load your Logo.

Now you must type:

| cpm <ENTER>

This is SHIFT – @

After some clunking noises from the disc drive, the screen will come up with a message like this:

Welcome to

Amstrad LOGO V1.1
Copyright (c) 1983, Digital Research
Pacific Grove, California

Dr. Logo is a trademark of
Digital Research

Product No. 6002-1232

Please wait

When the program is fully loaded, the screen will look like Figure 1.1 below.

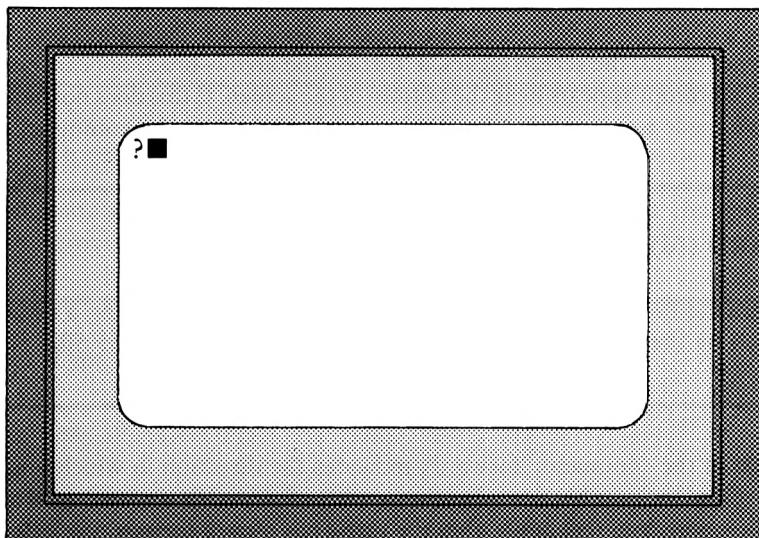


FIGURE 1.1

When the computer is awaiting your command, it displays the Logo prompt ‘?’ followed by the square cursor where your command will appear. The computer expects all its instructions to be in lower case – at this stage, it should be in lower-case mode, but if you start getting capital letters when you type things in, just tap the CAPS LOCK key once.

As you can see, the computer is requesting instructions, so let's give it something to do.

How about: ‘forward 50 <ENTER>’. This means type forward, hit space, type 50, hit <ENTER>.

REMEMBER – THERE IS A DIFFERENCE BETWEEN THE LETTER “O” AND THE NUMBER “0”. BE CAREFUL TO USE EACH CORRECTLY OR THE COMPUTER WILL NOT UNDERSTAND.

It must be typed in ‘forward 50 <ENTER>’
not ‘forward 5o <ENTER>’

Try it! Don’t forget the space between ‘forward’ and ‘50’, or the computer will not understand that either.

You will now receive a message saying:

I don't know how to forward

This is because DR Logo only uses abbreviated versions of the various commands, so try

fd 50 <ENTER>

When you type that in, the screen clears, a ‘turtle’ (the arrow-head) appears and draws a line, and the ‘?’ prompt appears near the bottom of the screen. If this does not happen, then you may have made a typing mistake. Try again! Let’s look at the command.

FORWARD (fd)

This command moves the turtle forward in the direction it is facing a distance determined by the number following the command, for example fd 50.

The first part of the command, ‘forward’, indicates the direction in which you want the turtle to move and the number describes the number of steps that the turtle will take in that particular direction. You will notice that the turtle has moved forward towards the top of the screen leaving a line behind it, a bit like a line drawn with a pencil or a pen.

Let’s get the turtle back. Try typing in the following command.

bk 50 <ENTER>

As you can see the pencil point (the turtle) has returned to the middle of the screen, leaving the line drawn on the screen, as shown below in Figure 1.2.



FIGURE 1.2

Let's give the turtle another command and tell it to go back from its starting position. Try typing in:

```
bk 50 <ENTER>
```

This might tell us something about the height of the screen. That is, that it is more than 100 (50+50) screen units high. Let's look at that command:

BACK (bk)

This command moves the turtle back a given distance without changing the direction in which the turtle is facing, for example bk 50.

Try it again. Type:

```
bk 10 <ENTER>
```

To get the turtle back to the middle of the screen, type:

fd 60 <ENTER>

remember the space

Now that the turtle is in the middle of the screen again, let's make it do something new. Make it turn to the left by typing in:

```
lt 90 <ENTER>
```

That instruction will have turned the turtle to the left so it is sitting facing in that direction, awaiting further commands. Summarising this new command:

LEFT (lt)

LEFT when used with a number is a command that rotates the turtle anticlockwise through a given angle. For example: lt 90.

Now that the turtle is facing to the left try moving it in that direction; type in:

```
fd 80 <ENTER>
```

Then, try the command:

```
bk 160 <ENTER>
```

At this point, your screen should look like the diagram shown below in Figure 1.3 (plus the turtle, of course).

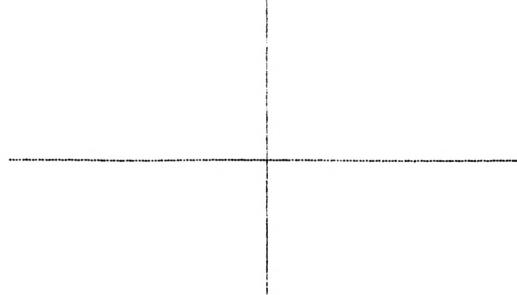


FIGURE 1.3

Clearly, the screen is more than 160 screen units wide.

As you would expect, the 'left' command has a right-handed colleague:

RIGHT (rt)

This command rotates the turtle clockwise through a given angle, for example:
`rt 90.`

Before going on to explore new command words, let's get rid of the old drawing. We need a new piece of paper, i.e. a way to:

CLEARSCREEN (cs)

This erases all the graphics and puts the turtle back in its starting position.

To try this out type:

```
cs <ENTER>
```

'cs' (clearscreen) tells the computer to erase the graphics and put the turtle back in the middle of the screen. Once again, the prompt and the cursor in the bottom left-hand corner reminds us that it is waiting for its next command.

This time, let's make the turtle draw a square. Before doing this however, we must think very carefully about the steps or commands we give the turtle, and the size of the square we want to draw. Let's say that we want to draw a square measuring 80 screen units by 80 screen units like the one shown below in Figure 1.4.

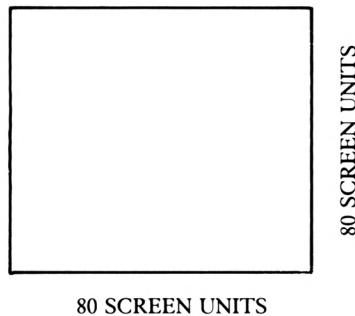


FIGURE 1.4

It is usually best to write down a list of computer commands on paper before you type anything into the computer. The commands will be as follows:

```
cs <ENTER>
fd 60 <ENTER>
lt 90 <ENTER>
fd 60 <ENTER>
lt 90 <ENTER>
fd 60 <ENTER>
lt 90 <ENTER>
fd 60 <ENTER>
```

This list of commands needs entering carefully. If you do make an error before pressing ENTER, you can correct it with the help of the DELETE key. Otherwise, you will have to start again with ‘cs’. By the time you press the last ENTER, your screen will look as in Figure 1.5 below:



FIGURE 1.5

Once again, the prompt is indicating that the computer is waiting for the next command.

With the five commands used so far, all sorts of rectangular shapes can be drawn and they don’t all need to start at a corner! Try typing in the following commands:

```
cs <ENTER>
fd 60 <ENTER>
rt 90 <ENTER>
fd 100 <ENTER>
rt 90 <ENTER>
fd 60 <ENTER>
rt 90 <ENTER>
fd 100 <ENTER>
```

Having followed those commands, the turtle will have drawn a rectangle and the screen will look like that shown in Figure 1.6 below:

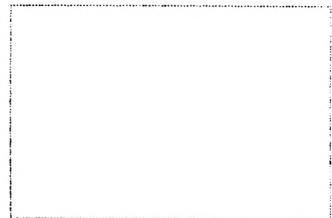


FIGURE 1.6

Clear the screen ('cs <ENTER>') and try the next set of commands.

```
fd 60 <ENTER>
rt 90 <ENTER>
fd 20 <ENTER>
rt 90 <ENTER>
fd 40 <ENTER>
lt 90 <ENTER>
fd 40 <ENTER>
rt 90 <ENTER>
fd 20 <ENTER>
rt 90 <ENTER>
fd 60 <ENTER>
rt 90 <ENTER>
fd 20 <ENTER>
```

Your screen should now proudly display “L” for Logo and should look like this, as shown below in Figure 1.7:

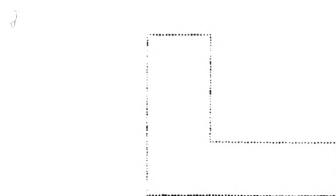


FIGURE 1.7

What about the poor old turtle sitting there in the middle? Can't we give it a rest? The Logo command to do this is:

HIDETURTLE (ht)

This command makes the turtle invisible; it can also draw a little faster when invisible.

This command does just what it says! It hides the turtle. In order to get the turtle back, a further command is used:

SHOWTURTLE (st)

This command makes the turtle visible to the user.

This command brings the turtle out of hiding, enabling the user to see just where it is. Let's investigate this process a little. First of all get the turtle to the centre of the screen. Type in:

cs <ENTER>

Now that you know just where the turtle is, type in:

ht <ENTER>

Next, move it around a little, say by means of:

fd 80 <ENTER>

As you can now see, even though the turtle is hiding, it still leaves a trail. To get it back just use:

st <ENTER>

Now, some exercises. Try to complete these without looking at the possible answers contained in the solutions chapter. The answers provided are not the only correct solutions to the problems because many possible solutions exist. What is a good idea though is to practise using the turtle and the commands learned so far. REMEMBER: It is best to plan and write down your list of commands before you type them into your computer.

Projects

PROJECT 1 By experimenting with ‘forward’ and ‘back’ try to discover how high the Logo screen is.

PROJECT 2 Using ‘left’, ‘right’, ‘forward’ and ‘back’, find out how many screen spaces wide the Logo screen is.

PROJECT 3 Using the Logo commands introduced so far find out how high and how wide the turtle is.

PROJECT 4 A little puzzle!
First draw a little box; type in

```
cs <ENTER>
lt 90 <ENTER>
fd 20 <ENTER>
rt 90 <ENTER>
fd 18 <ENTER>
rt 90 <ENTER>
fd 20 <ENTER>
lt 90 <ENTER>
fd 50 <ENTER>
rt 90 <ENTER>
fd 60 <ENTER>
```

Now the task is to get the turtle back into the box in only four Logo commands. You can make a guess or use some mathematics. Only if you’re really stuck resort to the solutions!

PROJECT 5 A slightly more complex puzzle! First draw a maze by typing in:

cs <ENTER>	fd 60 <ENTER>
lt 90 <ENTER>	rt 90 <ENTER>
fd 20 <ENTER>	fd 110 <ENTER>
rt 90 <ENTER>	rt 90 <ENTER>
fd 20 <ENTER>	fd 80 <ENTER>
rt 90 <ENTER>	rt 90 <ENTER>
fd 50 <ENTER>	fd 130 <ENTER>
rt 90 <ENTER>	rt 90 <ENTER>
fd 40 <ENTER>	fd 10 <ENTER>
rt 90 <ENTER>	rt 90 <ENTER>
fd 80 <ENTER>	
rt 90 <ENTER>	

Now see how quickly you can get the turtle home without crossing any lines. Twelve commands should be the minimum required.

Chapter 2

So far, we have been able to get the computer to obey instructions as we type them in. This is called programming in ‘direct mode’. However, once the computer has obeyed the Logo command, that particular line is discarded, never to be used again. If you wish to create something that you can use repeatedly, then it must be stored in a:

Procedure

A procedure is a named list of instructions that the computer will understand and obey. It stores each command as you type it in and only acts on these when you tell it to. To give the list of instructions a name, the word ‘to’ is used. Try this:

```
to box <ENTER>
```

You will notice that the shape of the prompt has changed from a ‘?’ to a ‘>’. This means that you are now in procedural mode and are able to type in a list of commands which are defined by, in this case, the word ‘box’. When you have finished your list of commands you will be able to make the computer execute them simply by typing ‘box <ENTER>’.

The first box is going to measure 50 screen units by 25 and will be in the top left hand corner of the screen. The first instruction will be ‘forward 50’, so type in:

```
fd 50
```

You will see that the prompt is there, waiting for more instructions, but since this is the end of the first command you must hit <ENTER>.

Well, that’s the first line in. The cursor tells you that the computer is waiting for another instruction. To get our second command in, all you have to do is type it in and then hit <ENTER>.

Have a go at typing in the rest of the instructions necessary to draw the box. You can make up your own or type in the following:

```
lt 90 <ENTER>
fd 25 <ENTER>
lt 90 <ENTER>
fd 50 <ENTER>
lt 90 <ENTER>
fd 25 <ENTER>
lt 90 <ENTER>
```

To finish a procedure, you must tell the computer that you have come to the end of your list of instructions and this is done by typing in:

```
end <ENTER>
```

Once <ENTER> is pressed, you will receive the message ‘box defined’ which means that the list of commands defined as ‘box’ is safely in the computer’s memory and will be available for use whenever required. Since the computer is now ready to obey the procedure, let’s make it do so by typing:

```
cs <ENTER>
box <ENTER>
```

and the machine does as it’s told.

Procedure ‘box’ is now safely stored in the computer’s memory and will be obeyed whenever the command ‘box <ENTER>’ is typed in. Having performed the procedure, the computer returns to direct mode and if you want to, you can now add anything you like to the box drawn on the screen.

If you move the turtle to another part of the screen and then type ‘box <ENTER>’, the program will be obeyed wherever the turtle happens to be. This, of course, means that you can draw the same box as many times as you like in lots of different positions just by moving the turtle with a direct command and then typing:

```
box <ENTER>
```

Try it and see. It’s almost certain that the turtle will disappear off the screen occasionally but it will usually come back! If not, type ‘cs <ENTER>’ to clear the screen and put the turtle back in the centre.

As you can see, the word ‘box’, when typed in will tell Logo to draw a box just as if it were a real Logo command. However, it is a procedure and not part of the Logo language and so, to distinguish between real Logo commands and procedure names, the real Logo commands are called ‘primitives’. Thus a Logo primitive is one of Logo’s basic command words such as ‘forward’, ‘right’ etc.

In Logo, the program ‘box’ is referred to as a ‘procedure’. Any procedure is defined by use of the word ‘to’ and, once the procedure is complete, the process is terminated by means of the word ‘end’. You may try this out by making up some of your own procedures or ‘procs’ and storing them under various names.

By now, your screen will probably look a bit of a mess, something like the one shown below in Figure 2.1

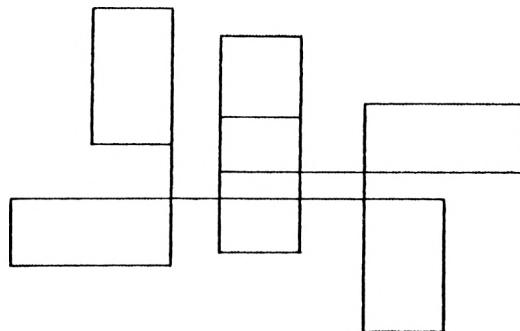


FIGURE 2.1

Let’s clear the screen by typing ‘cs <ENTER>’ and have a look at the program called ‘box’.

```
to box
  fd 50
  lt 90
  fd 25
  lt 90
  fd 50
  lt 90
  fd 25
  lt 90
end
```

Annotations for the 'box' procedure:

- 'to' says that a definition will follow
- 'box' is the command being defined
- Movement and distance commands

Just remember that every time you type ‘box <ENTER>’, the computer will follow the list of commands under the definition of ‘box’ regardless of where the turtle is positioned.

By the way, the computer will forget all procedures whenever you switch it off and it will also be necessary to re-load the ‘Logo’ program.

It might be a good thing now to learn how to draw some more complicated patterns without having to use direct mode.

When you look at patterns, you will see that often they consist of the same shape repeated in different positions, just like the one shown below in Figure 2.2.

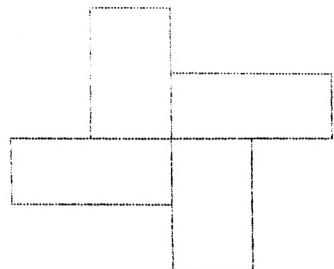
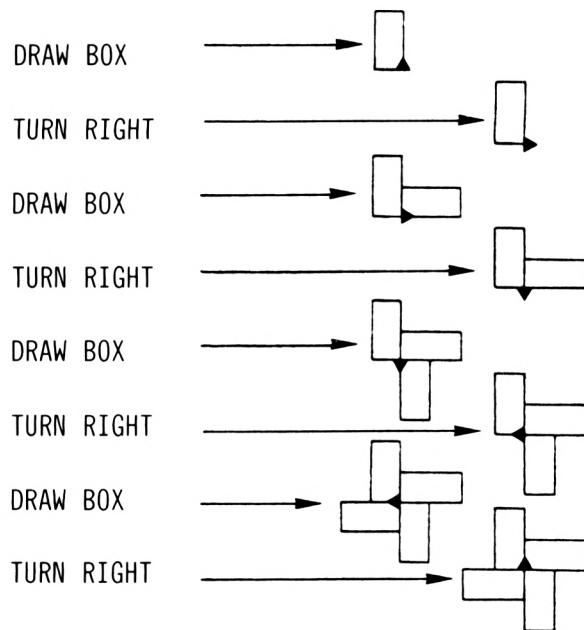


FIGURE 2.2

Notice that this simple pattern is made up of a rectangle like that drawn by the program, in different positions and orientations on the screen. The sequence that we would need to go through to create the pattern on the screen is something like:



Let's now write a short program to draw this pattern for us. The first thing to do is to think of a name: why not 'draw'? So, to start the process off, type:

```
to draw <ENTER>
```

The computer is now ready to receive a list of commands under the definition "draw". Remember that because the computer already has a procedure called "box" stored in its memory, you need only type "box" in the new procedure (called "draw") whenever you want the box to appear on the screen. Here are the necessary instructions to draw the above pattern. Type them in!

```
to draw <ENTER>
  box <ENTER>
  rt 90 <ENTER>
end <ENTER>
```

If there is something on the screen already, type 'cs' to clear it. To call the procedure or program, simply type in:

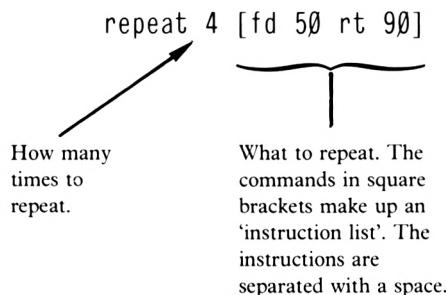
```
draw <ENTER>
```

Lo and behold your 'draw' procedure is executed. Notice that the 'box' procedure is still in the computer's memory and can be used by other procedures, as in this case it is used by procedure 'draw'.

What the 'draw' procedure does is to tell the computer to "draw 'box' and turn right, 4 times"; the program was 9 lines long. To save all this typing, we could use a new command called:

REPEAT (repeat)

REPEAT is a very powerful command which can tell the computer exactly what to repeat, when, and how many times. Its basic structure or ‘syntax’ is:



Just to test this out clear the screen with ‘cs <ENTER>’ and then type in the following direct command. To get the first square bracket hit the ‘[’ key on the right of the keyboard and to get the other square bracket hit the ‘]’ key.

```
repeat 4 [fd 50 rt 90] <ENTER>
```

Lo and behold, in one line, ‘repeat’ allows you to draw a complete square!

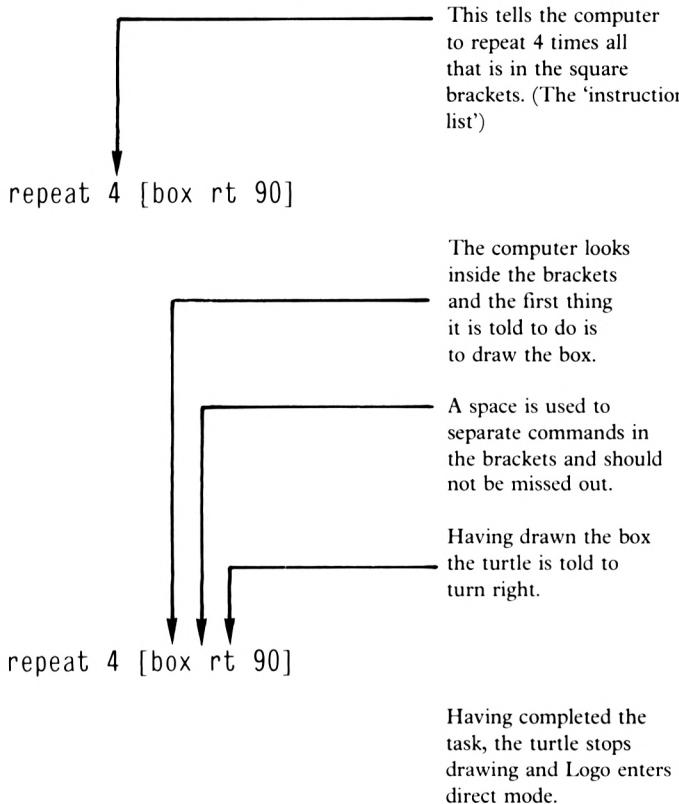
Now to use this new-found knowledge in the ‘draw’ procedure. First, let’s have a look at its structure:

```
to draw
box
rt 90 SECTION 1
box
rt 90 SECTION 2
box
rt 90 SECTION 3
box
rt 90 SECTION 4
end
```

Procedure ‘draw’ is made up of four identical sections and it could be shortened considerably by means of the ‘repeat’ command. All that is needed is a four times repeat of ‘box rt 90’, i.e:

```
cs <ENTER>
repeat 4 [box rt 90] <ENTER>
```

Either one of these programs will operate in exactly the same way. Let's look at the short version of the program to understand how it works:



You will have noticed that as you define procedures, the commands typed in earlier begin to disappear from the small text area of the screen. Well one way of overcoming this is to use a command called

TEXTSCREEN (ts)

This is a command that dedicates the whole of the screen to text.

This command will enable you to view the whole of the procedure as you type it in. You will notice that the graphics turtle is nowhere to be seen, because the turtle will not operate on the text screen. It is possible to locate text on the graphics screen however, as you will find out a little later in this book.

Try typing

```
ts <ENTER>
```

and you will notice the prompt in the top left hand corner of the screen awaiting your command.

Now, let's have a look at another use of repeat and investigate some different uses of the 'right' command. First of all, we'll make up a simple procedure called 'cross' as our raw material, so type in:

```
to cross <ENTER>
```

Next, type in this one line program which uses the 'repeat' function.

```
repeat 4 [fd 70 bk 70 rt 90] <ENTER>
end <ENTER>
```

Before you execute the program, look at the commands and work out (using pencil and paper) what the turtle will draw.-

Now you can run the procedure by typing in:

```
cross <ENTER>
```

and see if you were correct in your prediction.

Now have a look at the cross and examine the parts that make it up. One way of subdividing the cross would be into two angles as shown below in Figure 2.3.

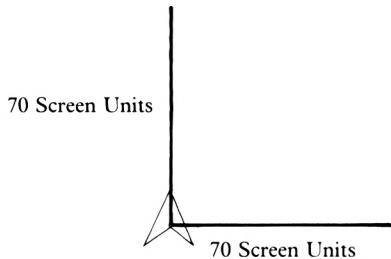


FIGURE 2.3

If we wanted the turtle to draw this, what we could make it do would be:

```
to go forward 70
and then back 70
next turn right
next go forward 70
and then back 70
and finally turn left.
```

As we will use this routine a few times, it will be useful to save it as a procedure, so type ‘ts <ENTER>’ then type

```
to angle
fd 70 <ENTER>
bk 70 <ENTER>
rt 90 <ENTER>
fd 70 <ENTER>
bk 70 <ENTER>
lt 90 <ENTER>
end <ENTER>
```

Type it in and try it out.

To turn the turtle to the left or right we have used the commands left (lt 90) and right (rt 90) with the result that in either case the turtle has made a full ninety degrees (right-angle) turn to the left or right respectively. Try using a series of ‘rt 90’ commands and you will notice that it takes four of these to turn the turtle all the way round to finish facing in its original position.

What number would you use if you only wanted to half of our normal turn to the left or right?

That’s right. A number halfway between 0 and 90, i.e. 45. Let’s try this out; type in:

```
rt 45 <ENTER>
```

You will see that the turtle has performed a half turn to the right and your screen should look as shown in Figure 2.4:

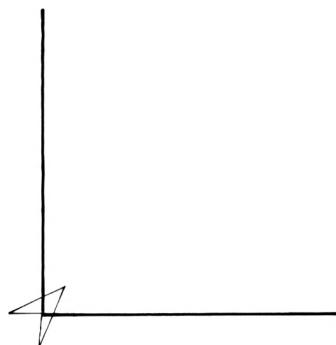


FIGURE 2.4

Now type in:

```
fd 70 <ENTER>
```

and the turtle draws a line 70 screen units long midway between the vertical and horizontal lines. Now return the turtle back to its starting point by typing:

```
bk 70 <ENTER>
lt 45 <ENTER>
```

Suppose we wanted to draw a similar line a quarter turn to the left, that is halfway between 0 and 45. As the angle required is now 45 divided by 2, type in:

```
lt 22.5 <ENTER>
fd 70 <ENTER>
bk 70 <ENTER>
rt 22.5 <ENTER>
```

and your screen should look as shown in Figure 2.5:

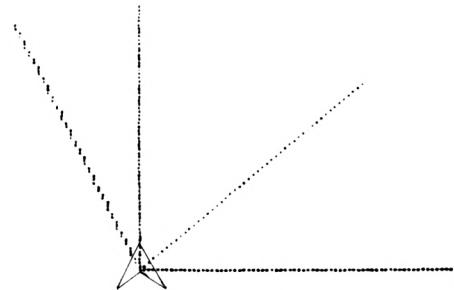


FIGURE 2.5

Thus, any number following the command ‘right’ or ‘left’ will face the turtle in a new direction. This will, of course, rotate any graphics that follow. Try it out by typing in:

```
cs <ENTER>
```

to clear the screen. Re-run ‘angle’ by typing:

```
angle <ENTER>
```

Next, turn the turtle to the right by typing:

```
rt 30 <ENTER>
```

and run ‘angle’ again.

If you do this a few more times, you will begin to see a sort of star pattern appear.

Let’s now use this knowledge to make up a program to draw a star. The first thing to do is to plan the program and draw out what is required on paper. This enables us to write out the program before typing in the commands. The proposed graphics screen is shown below in Figure 2.6 below.

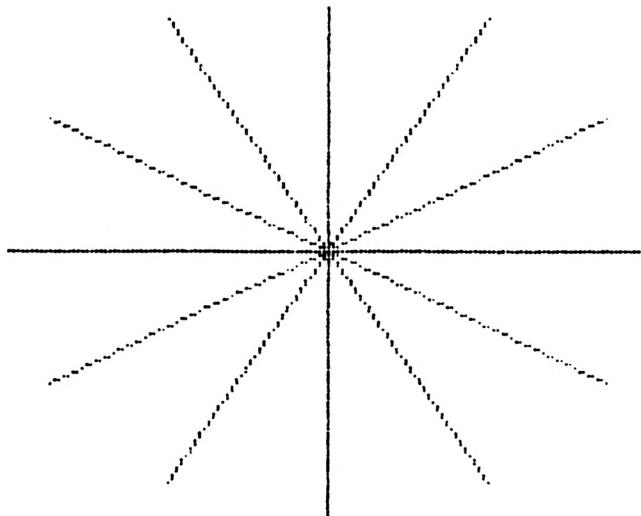


FIGURE 2.6

If you examine this star carefully, you will see that it could be made up of 3 separate cross programs. You will notice that each time the cross is drawn, it makes a $\frac{1}{3}$ turn to the right. As a full right turn is 90 degrees, the $\frac{1}{3}$ turn is $90 \div 3$ or 30 degrees.

So, to program this star, we could say:

to glow <ENTER>	or	to glow <ENTER>
cross <ENTER>	using the	repeat 3 [cross rt 30] <ENTER>
rt 30 <ENTER>	repeat	end <ENTER>
cross <ENTER>	function	
rt 30 <ENTER>		
cross <ENTER>		
rt 30 <ENTER>		
end <ENTER>		

Just for fun, let's turn the star procedure through 1/6 of a turn (15 degrees) and call the new procedure 'blob'.

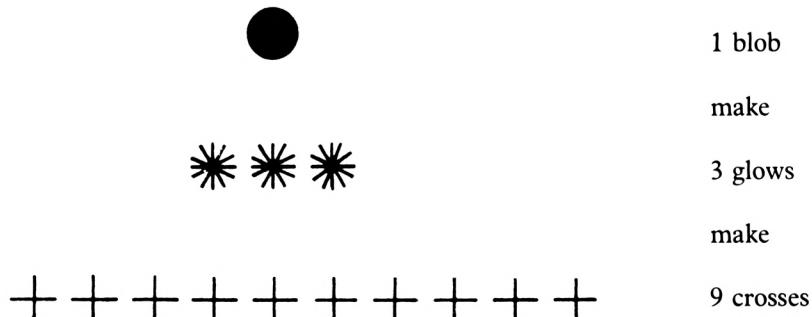
First type:

```
to blob <ENTER>
repeat 3 [glow rt 15] <ENTER>
end <ENTER>
```

This is one of the really clever features of Logo, this ability to define a procedure and then embed it in another procedure and so on and so on.

This process of putting one procedure inside another is called 'nesting'.

The final 'blob' program is thus made in the following way:



The final ‘blob’ program looks like this:

```
to blob <ENTER>
repeat 3 [glow rt 15] <ENTER>
    ↑
    ↑
end <ENTER>

to glow <ENTER>
repeat 3 [cross rt 30] <ENTER>
    ↑
    ↑
end <ENTER>

to cross <ENTER>
repeat 4 [fd 70 bk 70 rt 90] <ENTER>
    ↑
    ↑
end <ENTER>
```

3 glow progs. nested in the blob prog. at 15 degree intervals.

3 cross progs. in each of the stars at 30 degree intervals.

4 angle-type progs. in each of the crosses at 90 degree intervals.

Over 100 turtle movements obtained just by typing:

```
blob <ENTER>
```

That's the power of Logo!

Let's now imagine we wanted to draw a pattern similar to the one shown below in Figure 2.7

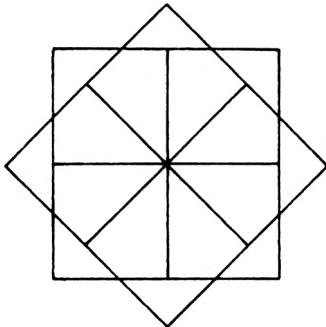


FIGURE 2.7

As you may have noticed it is a pattern made up of one square, so our first procedure might be ‘square’.

Notice that these instructions have not been followed by the usual reminder to hit <ENTER>. It is still necessary to do so, but you will probably have realised by now that you must hit <ENTER> at the end of each complete command, so further reminders will be left out.

```
to square
fd 50
rt 90
fd 50
rt 90
fd 50
rt 90
fd 50
rt 90
end
```

Of course, another way of defining exactly the same procedure would be:

```
to square
repeat 4 [fd 50 rt 90]
end
```

Having defined a basic shape, we can now ‘nest’ it into a procedure to draw:

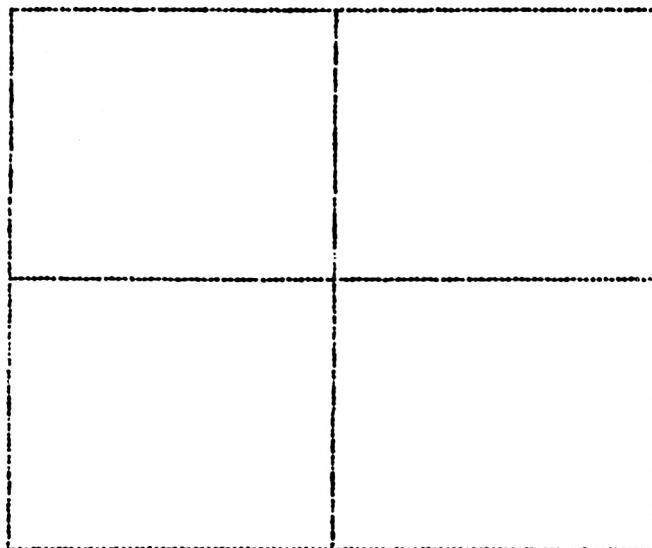


FIGURE 2.8

For example:

```
to flag
repeat 4 [square rt 90]
end
```

Thus we have commanded the turtle to

```
then      1) draw the square
          2) turn right
then      3) draw the square
then      4) turn right
etc.
```

Now, for the final procedure:

```
to rotate
repeat 2 [flag rt 45]
end
```

This is a fairly standard procedure and should present no problems! Try it out with a direct mode entry:

```
cs
repeat 5 [rotate rt 36]
```

What you are doing with this line is to call ‘rotate’ which calls ‘flag’ which calls ‘square’:

So,

- 1) A line 50 units long is drawn and the turtle turns right.
- 2) This is carried out four times under the procedure ‘square’.

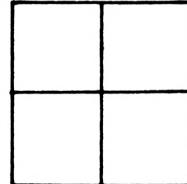
Result



- 2) The square procedure is carried out four times with a right turn after each square.

Result

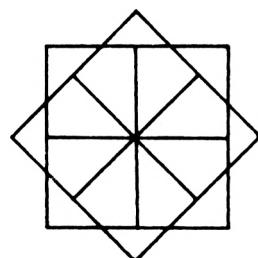
This is carried out under the procedure 'flag'.



- 3) The flag is drawn, the turtle turns 45 degrees and flag is redrawn finishing with a 45 degree turn

This is carried out under the procedure 'rotate'.

Result



- 4) Rotate is repeated five times with a 36 degree turn between each

With the command
repeat 5 [rotate rt 36]

This results in a picture that looks like the one shown below in Figure 2.9

Just to tidy the whole procedure up define this final direct line as ‘pattern’, i.e.

```
to pattern
repeat 5 [rotate rt 36]
end
```

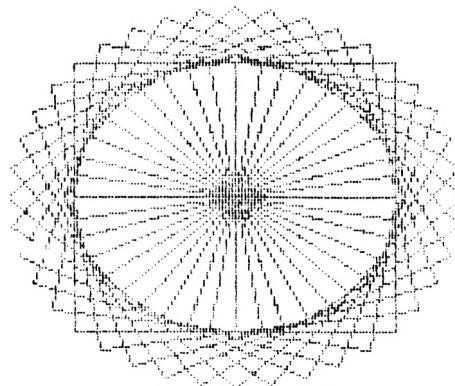


FIGURE 2.9

You might have noticed that whenever you start up Logo that you are automatically in ‘textscreen’ mode. If you give the computer a command like ‘fd 30’, the screen changes to a graphics screen on which the turtle can draw, with just five lines in which you can view any text. This is what is known as ‘splitscreen’ (ss). One of the disadvantages of splitscreen is that you lose these bottom five lines for graphics. For most programs this is no great problem but say, for instance, we wished to draw two patterns at different points on the screen; the lower of these might not appear fully. Try, for instance, moving the turtle by means of:

```
cs rt 135 fd 100 lt 135
```

Now when you run ‘pattern’ it will no longer fit fully onto the splitscreen. Try this out and you will see that the bottom of the pattern passes under the four lines of text. These four lines can be recovered for graphics by means of the Logo command

FULLSCREEN (fs)

This is a command that dedicates the whole of the screen to turtle graphics.

You can still type in commands in direct mode or even define new procedures on the ‘fullscreen’, but you won’t be able to see what you are typing in. Try a few commands and see (or, rather, don’t see!).

In order to tidy up the nomenclature the mode that you’ve been using before, ‘splitscreen’, is formally defined below:

SPLITSCREEN (ss)

This is a command that will produce a turtle field within which the turtle can operate leaving a five-line text field below.

If you now type ‘ss’ the cursor will once again appear on screen.

Projects

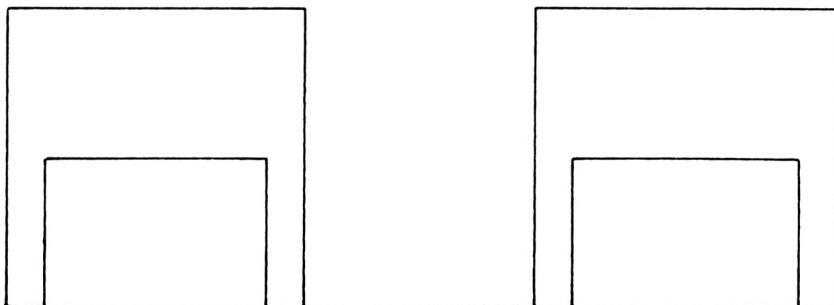
PROJECT 1 Use the procedure ‘box’ defined below for Project 1.

```
to box
  fd 30
  rt 90
  fd 30
  rt 90
  fd 30
  rt 90
  fd 30
end
```

Type in the following direct commands:

```
cs  
rt 90  
fd 120  
lt 90  
fd 60  
lt 90  
fd 50  
lt 90  
fd 60  
rt 90  
fd 120  
rt 90  
fd 60  
rt 90  
fd 50  
rt 90  
fd 60
```

Now, using procedure ‘box’ twice, create the drawing shown below:



PROJECT 2 Type in the following direct commands (with the turtle hidden)

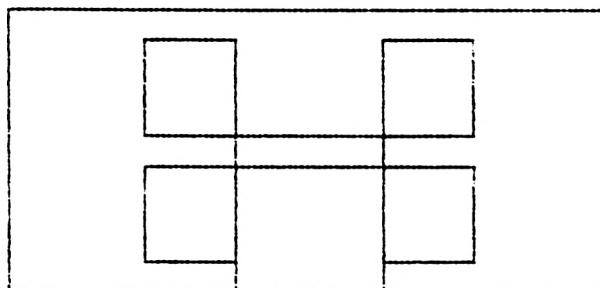
```
cs  
lt 90  
fd 10  
rt 90  
fd 16  
rt 90  
fd 10  
lt 45  
fd 14  
rt 45  
fd 14  
rt 45  
fd 14  
rt 45  
fd 26  
rt 45  
fd 14  
rt 45  
fd 44  
rt 45  
fd 14  
rt 45  
fd 26  
rt 45  
fd 28  
rt 45  
fd 44  
rt 45  
fd 28  
rt 45  
fd 36  
rt 45  
fd 44  
rt 45  
fd 54  
rt 90  
fd 16  
rt 90
```

Using direct commands, draw a line into the box without crossing any of the lines. If you're good, it can be done with 13 commands.

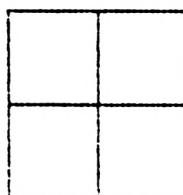
PROJECT 3 Type in the following direct commands:

```
cs  
rt 90  
fd 100  
lt 90  
fd 90  
lt 90  
fd 200  
lt 90  
fd 90  
lt 90  
fd 100
```

Using the procedure 'box', create the pattern shown below:

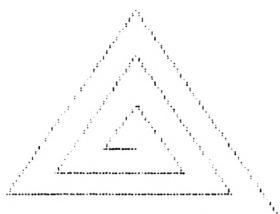


PROJECT 4 Using the procedure 'box' create the pattern shown below:



PROJECT 5 Try drawing this pattern:

CLUE: all the internal angles are 60 degrees.



Chapter 3

Everywhere the turtle has gone so far, it has left a trail behind it – a line on the screen. This has meant that all pictures on the screen have needed to be joined together and that it has not been possible, for instance, to draw a picture that contained two or more elements that were disconnected, such as Figure 3.1.

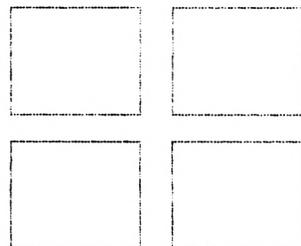


FIGURE 3.1

Logo has a command that allows this to be done, it is...

PENUP (pu)

On the command ‘pu’, the turtle lifts up its pen and from then on it can move without drawing.

To get it to draw on the screen once more, the command used is:

PENDOWN (pd)

This command tells the turtle to place the pen back on the paper (the screen), allowing the turtle to draw again when asked to move about the screen. This is the state that Logo is in when it is first run, known as its ‘default’ state.

Let's try using these two commands in the direct mode. Type in the following commands and watch what happens.

cs	
fd 50	Draw a line.
pu	Lift up pen.
fd 15	Move and
rt 90	Turn invisibly.
pd	Put down pen.
fd 50	Draw a line.
rt 90	Turn.
pu	Lift up pen.
fd 30	Move invisibly.
pd	Put down pen.
lt 145	Turn.
fd 100	Draw a line.

As you can see, by using these two commands we can now draw more than one separate shape on the screen; this gives us much greater flexibility in design.

Consider this simple example using the triangle procedure shown below. Let's draw the triangle so we can repeat it at different locations on the screen without losing any of the pattern.

```
to shape
  rt 30
  fd 70
  rt 120
  fd 70
  rt 120
  fd 70
  rt 90
end
```

Call the procedure 'shape' in direct mode, then if you want to you can pick up the turtle pen using 'pu', move the turtle using the 'forward', 'right', 'back' or 'left' commands, put the pen down using 'pendown' and re-call the triangle. Or, you could try this list of direct mode commands:

```
pu  
fd 25  
pd  
shape  
pu  
rt 60  
fd 50  
pd  
shape
```

Now have a look at the next procedure:

```
to pattern  
repeat 24 [shape lt 15]  
end
```

Clear the screen using ‘cs’ and if you now type ‘fs pattern’ the turtle will dutifully create the triangle at 15 degree intervals at the centre of the screen; the turtle will draw a pattern like the one shown below in Figure 3.2.

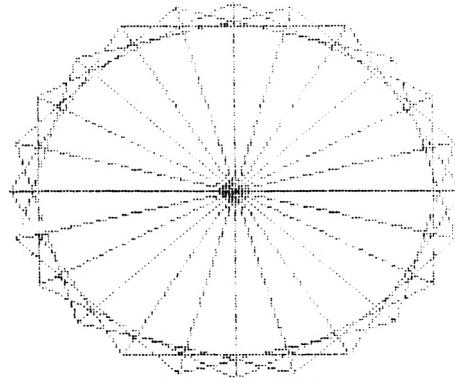


FIGURE 3.2

Now, in direct mode, use the following commands to move the turtle and re-draw the pattern.

```
pu  
rt 45  
fd 60  
lt 45  
pattern
```

There, task complete – or is it?

The turtle is actually going through the procedure without drawing anything. Can you see what the problem is? Well, let it finish and then type:

```
pd  
pattern
```

That's better, the pattern is being repeated in the same place but this time it's being drawn as well, just where we wanted it. Always remember if you pick up the pen to move the turtle you must put it down again to commence drawing. Can you calculate the distance between the centres of the two patterns? It must be 60 steps as the only move command used was 'fd 60'.

The screen now contains two versions of 'pattern', both the same except that one still has turtle sitting on its handiwork! Ungrateful as it may seem there's a Logo command to hide the turtle, not surprisingly called...

HIDETURTLE (ht)

This command hides the turtle by making it invisible. The turtle will do everything else just as normal.

If 'pendown' is selected then it will draw a line when moved. You won't see the turtle but you will see its trail. Test this out by typing in:

```
ht
```

There! The turtle is gone, leaving its handiwork behind. Of course there's a command to bring it back too, this is...

SHOWTURTLE (st)

This command simply persuades the turtle to show itself again.

With this pair of commands we are now in a position to create much more adventurous graphics procedures. Let's try one that sends the turtle roving around the screen a little. Its structure will be...

- (i) draw pattern
- (ii) lift pen up
- (iii) change angle
- (iv) move turtle
- (v) put pen down
- (vi) repeat procedure from (i)

Try this by typing in the following ‘rove’ procedure:

```

to rove
pattern      draw pattern
pu           move turtle
rt 45        and rotate
fd 60         invisibly
pd
pattern      draw pattern
pu           move turtle
lt 120       and rotate
fd 60         invisibly
pd
pattern      draw pattern
ht           hide turtle
end

```

Run this in the usual way by clearing the screen and typing ‘rove’.

When a graphics procedure is being carried out, it can take a fair time to complete. This time is increased when the turtle has to be re-drawn whenever it moves or changes direction. We can help the computer save time by using ‘hideturtle’ while a procedure is being carried out. It is usual to save as much time as possible using ‘hideturtle’ as the first command in any procedure. If you wish to test this out, time the procedure ‘rove’ and make a note of how long it took the turtle to complete the whole operation. Next type ‘ht’ and then time the procedure again. Not only does it run faster but the visual effect is quite different too!

Project

Design a procedure to draw a picture like the one shown below in Figure 3.3

A possible solution is given in the solutions chapter.

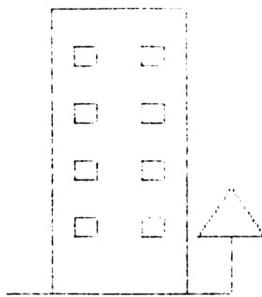


FIGURE 3.3

Let's now have a look at the method provided in DR Logo by which we can insert new commands in previously defined procedures and also erase and/or replace commands. In other words, the facility to EDIT any procedure already stored in the computer.

EDIT (ed)

This is a Logo command which puts the user in touch with the Logo editor. In this mode the user will be able to change any previously defined procedure.

Editing Procedures

Let's say that we want to make a couple of changes to a particular procedure. Suppose, for example, we wish to change the shape of the box in the 'box' procedure. If this procedure is no longer in your computer's memory, and this might well be the case, you should re-define it. If, however, you have not turned off your computer, or done Project 1 of Chapter 2, or hit <SHIFT-CTRL-ESC> since you typed in 'box' at the beginning of Chapter 2 you should be able to list the text of this procedure by typing

```
ed "box
```

On pressing ENTER the screen will change and a listing of the procedure will be given.

In the event that this particular procedure is not still in your computer all it will say is 'to box' and 'end' on the top two lines of the screen. If so, hit <ESC> and then type it in as below:

```
to box
fd 50
lt 90
fd 25
lt 90
fd 50
lt 90
fd 25
lt 90
end
```

Now type 'box' and run the procedure. Having done this, next type:

```
ed "box
```

and hit <ENTER>.

Note that the name of the procedure is preceded by double inverted commas (""). It is important that you use double and not single inverted commas, otherwise the computer will not recognise the entry as a procedure name.

The screen changes, ready to make alterations to the procedure. At the moment the screen looks as shown in Figure 3.4.

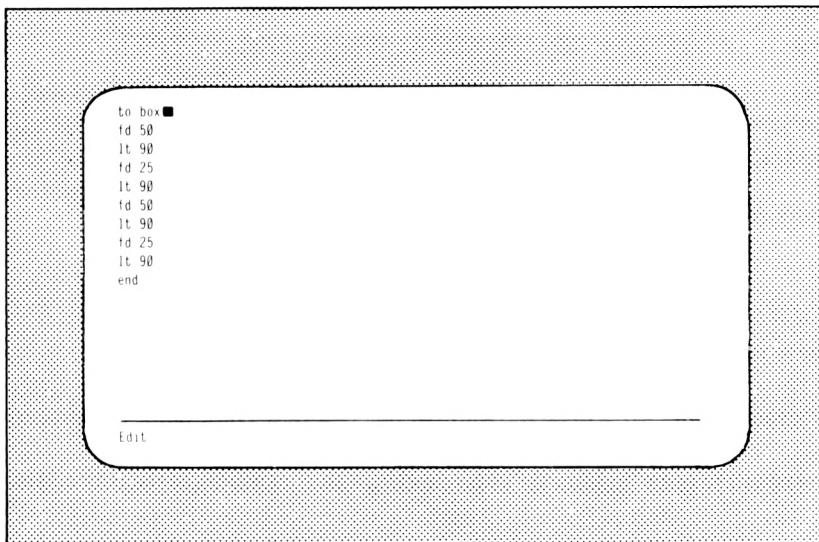


FIGURE 3.4

Let's assume that you want to change the distances involved in each of the 'forward 25' commands. The first thing to do is move the editing cursor to the first line you wish to change. To move the editing cursor up and down you use the grey arrow keys located on the right-hand side of the keyboard. Press the down-arrow key three times and this will move the cursor down (don't hit <ENTER> during this operation). Having selected the third command (fd 25) for modification, you can now delete the 25 by hitting the DELete key twice and type in say 50.

DON'T HIT <ENTER> YET

You can now bring the cursor down to the end of the next 'fd 25' command by use of the down-arrow key. Then move it to the left one place by hitting the left-arrow key once. You are now in a position to delete the '2' and type in a '5'. The cursor is now over the '5' of the 'fd 25' command. Now move the editing cursor one place to the right by hitting the right-arrow key once. You can now delete the '5' and type in a '0'. Having changed the necessary commands, your screen looks like the one shown below in Figure 3.5.

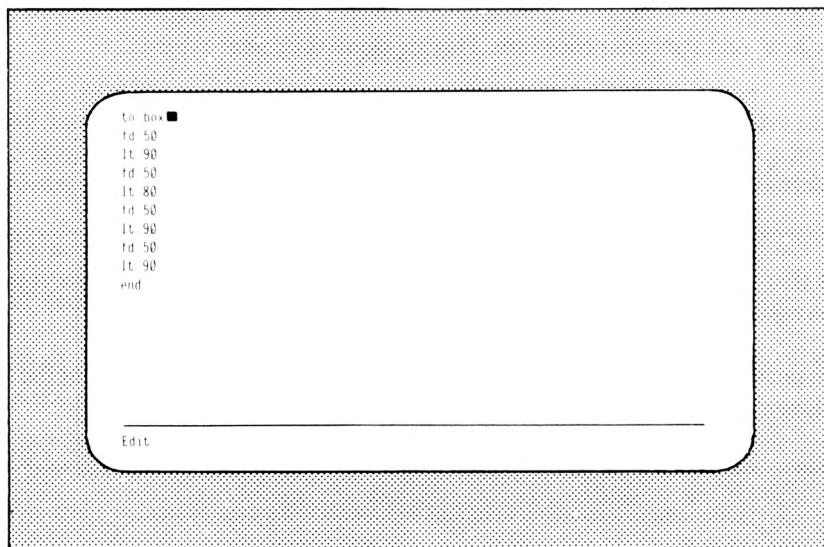


FIGURE 3.5

You are now ready to leave the Logo Editor and to do this you must hit



Will the new version of 'box' work? Try it by typing 'box'.

Let's now examine an edit where we want to insert a new command in a procedure.

Imagine you want to put a 'hideturtle' command in at the beginning of the procedure. Type:

```
ed "box
```

You should now be in edit mode with the procedure listed. You will notice the cursor in its usual position at the end of the first line of the procedure. It is now intended to insert the command 'ht' as the new first line. Now to open up a space for a new first line you can hit <ENTER>.

Try it a few times and see the effect: your screen will look a bit of a mess, so all you have to do is hit the <ESC> key.

You will now receive a message saying:

Stopped!

Now type

ed "box

and you will see that your original procedure is safe and sound. Now to put the 'ht' command in as your new first line, hit

ENTER

Now type 'ht' and your new first line will appear on screen. Now hit

COPY

Suppose we wanted to rub out a command? The first thing we must do is to get back into edit mode. Type:

ed "box

Now there is the procedure; let's imagine that the 'ht' command is no longer necessary. DR. Logo uses various control characters to control the screen display and cursor movement and these can be used to supplement the use of the keys already described. Try experimenting with a few of them before proceeding.

CTRL - A will locate the cursor at the beginning of the line at which it is currently situated.

CTRL - E will move the cursor to the end of its current line.

CTRL - H will act like the delete key and remove the character to the left of the cursor.

CTRL - D deletes the character covered by the cursor.

CTRL - O will open up a space on the screen (try this with the cursor at the beginning of a line).

CTRL - N moves the cursor down a line.

CTRL - P moves the cursor up a line.

CTRL - C ends the editing and re-defines the procedure just like the 'COPY' key.

Well you might like to stay with the arrow keys until you become a little more experienced and so in that case, hit the down-arrow key once and use the DELete key to take out 'ht'. Now you can use the DELete key a third time to finally delete the line, then hit <CTRL-C> or <COPY> to re-define the procedure.

Review Of Progress

By now you are in a position to create almost any pattern you wish, to move the turtle to any required screen position, to draw, store and recall procedures, and then call all these moves and procedures up by nesting them into a final procedure. All this can be achieved using combinations of these commands:

Drawing Mode

(fd)	FORWARD number	Positive or negative.
(bk)	BACK number	Positive or negative.
(rt)	RIGHT number	(Degrees of turn 0 to 360 + or -).
(lt)	LEFT number	(Degrees of turn 0 to 360 + or -).
(cs)	CLEARSCREEN	Wipes clear the screen.
(to)	TO PROCEDURE	Defines a procedure name.
(repeat)	REPEAT number [list]	Enables repetition of whatever is in the square brackets.
(pu)	PENUP	Enables turtle to lift its pen off the paper and move it to a given position on the screen without drawing a line.
(pd)	PENDOWN	Places the pen (turtle) in drawing mode.
(st)	SHOWTURTLE	Makes the turtle visible.
(ht)	HIDETURTLE	Hides the turtle and reduces the time taken for a procedure to be carried out.
(end)	END	Used to finish off all procedures

Edit Mode

ed "name" Lists the procedure to be altered in the edit mode.

arrow keys Keys used to move cursor while in edit mode.

CTRL – A moves to line start.

CTRL – E moves to line end.

CTRL – H removes the character and moves to the left.

CTRL – D deletes the character below the cursor.

CTRL – O opens up a space in the procedure.

CTRL – N moves the cursor down a line.

CTRL – P moves the cursor up a line.

CTRL – C ends editing session.

COPY ends editing session as well.

ESC Used to leave edit mode.

Let's now put most of these commands together to form one large procedure.

The pattern chosen is based on combined straight lines. It is shown below in Figure 3.6 but it would be a good idea for you to re-draw it on squared paper.

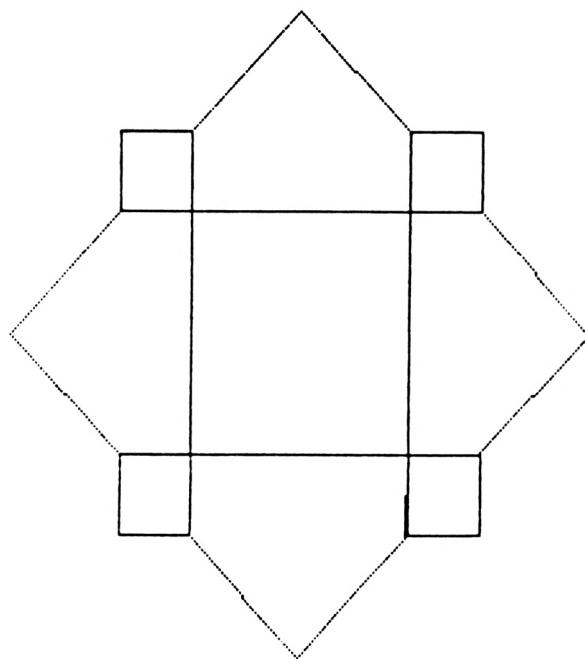


FIGURE 3.6

Let's start off with the large box in the middle of the pattern.

```
to bigbox
pu
fd 21
rt 90
fd 21
pd
repeat 4 [rt 90 fd 42]
end
```

Once you have run this you will notice that the turtle is facing in the right direction to start drawing one of the small boxes. Next try a procedure for the small box. (You might like to type 'ts' (textscreen) first.)

```
to smallbox
fd 14
lt 90
fd 14
lt 90
fd 14
lt 90
fd 14
rt 90
end
```

Now, clear the screen and run ‘bigbox’.

Now run ‘smallbox’.

Now all we have to do is to bring the turtle along the line it is facing, turn it in the right direction and re-run ‘smallbox’. What we need is to create a procedure to draw the next three small boxes. Call this procedure ‘lastbox’. Again, textscreen might be useful.

```
to lastbox
fd 42
rt 90
smallbox
fd 42
rt 90
smallbox
fd 42
rt 90
smallbox
end
```

If you now run ‘lastbox’, the three small boxes will be created but in the wrong position to make the required shape, so you must clear the screen and in direct mode type:

```
fs
bigbox
smallbox
lastbox
```

Now let's 'nest' those into one procedure called 'squares'

```
to squares
fs
bigbox
smallbox
lastbox
end
```

Now in direct mode enter:

```
cs
ht
squares
```

Bring the turtle back by typing 'st'.

Now let's fill in the triangle shapes between the small squares.

```
rt 90
fd 14
lt 45
fd 30
lt 90
fd 30
lt 45
fd 14
```

Type in these commands in direct mode. You will see that we have drawn what we wanted to and the turtle is in a similar position to when it started. Therefore we can repeat this procedure 4 times to complete the outline of our pattern, by using the REPEAT command, as shown below. An exclamation mark '!' will appear when your typing reaches the end of a horizontal screen line – just ignore it and carry on typing.

```
repeat 4 [rt 90 fd 14 lt 45 fd 30 lt 90
fd 30 lt 45 fd 14]
```

Type in the above command in direct mode and hit <ENTER>.

Now the easiest way to draw the whole thing is to define a new command and call it, say, 'picture' so type:

```
to picture
ht
squares
repeat 4 [rt 90 fd 14 lt 45 fd 30 lt 90
fd 30 lt 45 fd 14]
lt 45
pu
fd 30
pd
end
```

The last four commands, ‘left 45’, ‘penup’, ‘forward 30’ and ‘pendown’, have been included to bring the turtle back into the middle of the screen. When ‘picture’ is completed, run the procedure by typing: ‘picture’.

Not the fastest method for drawing that particular outline, but one that illustrates one approach to this particular problem.

Why not try:

```
cs repeat 8 [picture rt 45]
```

What is happening is that the turtle is running over the same ground. This is because the pattern is made up of symmetrical 90 and 45 degree angles.

Multistatement Lines

If you look back through this chapter you will notice that most procedures are written like this:-

```
to smallbox
fd 14
lt 90
fd 14
lt 90
fd 14
lt 90
fd 14
rt 90
end
```

In these procedures each command is situated on its own separate line. This makes the procedure easy to read and follow but if it was to be much longer it would not fit on the screen.

Let's now have a look at another procedure previously described:

```
to picture
ht
squares
repeat 4 [rt 90 fd 14 lt 45 fd 30 lt 90
fd 30 lt 45 fd 14]
lt 45
pu
fd 30
pd
end
```

You will notice that in the third line there are a lot of commands one after the other, separated by spaces. Well, the commands inside the square brackets make up what is known as an 'instruction list'.

The third line of this procedure is simply saying:

REPEAT EVERYTHING WITHIN THE
SQUARE BRACKETS FOUR TIMES

What's interesting however is that each command within the instruction list directly follows the other and is separated only by a space. This in fact is what is known as a 'multistatement line'.

With Logo any line in a procedure can include any number of commands as long as the line does not exceed 77 characters in length. By 'characters' is meant spaces, colons, quotes, signs, full stops etc. as well as letters and numbers.

Let's change the 'smallbox' procedure to look like this (an exclamation mark '!' will appear when you reach the end of a screen line – ignore this and just keep typing):

```
to sb
fd 14 lt 90 fd 14 lt 90 fd 14 lt 90 fd
14 rt 90
end
```

Now type 'sb' and as you can see the procedure works very well. You will notice that there are 40 characters to a line on the screen and that the first line of our 'sb' procedure is 47 characters in length. The exclamation mark is just Logo telling itself that there is more to follow.

You might now have in your computer's memory two procedures that do exactly the same thing; 'smallbox' and 'sb' are procedures that draw the same shape on the screen. Let's get rid of one of them with a new Logo command.

Do you know how many procedures you've defined and what their names are? Well, there's a Logo command to help you out. First of all however, type:

ts

Followed by:

pots

PRINT OUT TITLES (pots)

This command will display the names of all the procedures in the workspace.

Having done this you will no doubt see that your list includes a procedure defined as 'smallbox' and another defined as 'sb'.

Let's imagine that we wanted to dispense with the services of 'smallbox'. To do this, type:

er "smallbox

ERASE (er)

This command will erase a specified procedure from the workspace.

and just to prove that it has gone, type 'smallbox' and you will get a message saying:

I don't know how to smallbox

The programs so far developed have been silent, but DR Logo has a considerable sound capability.

SOUND

This command will output a note frequency and duration defined by the instruction list. For example

```
sound [1 20 50]
```

The first digit in the instruction list selects the channel. (This will be explained further in chapter 6.) The second number defines the tone of the note in a range 0 to 4095. The last number gives the duration of the note in hundredths of a second.

To test out the command type:

```
cs  
repeat 4 [fd 40 rt 90 sound [1 50 10]]
```

This is a group of commands that draws a rectangle of some description, but can you work out when the 'pip' is activated? Try it and see.

Now try in direct mode

```
sound [1 2000 10]
```

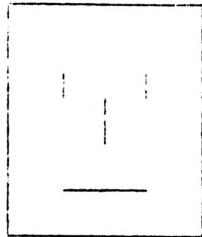
You could even try

```
repeat 40 [fd 3.93 rt 9 sound [1 100 2]]
```

It adds a certain elegance to the procedure but can easily become wearing if used to excess. Perhaps just one or two noises for instance at the end of a procedure is enough! We shall be looking at the musical capabilities of DR Logo in Chapter 6.

Projects

PROJECT 1 Try drawing a face like this one:



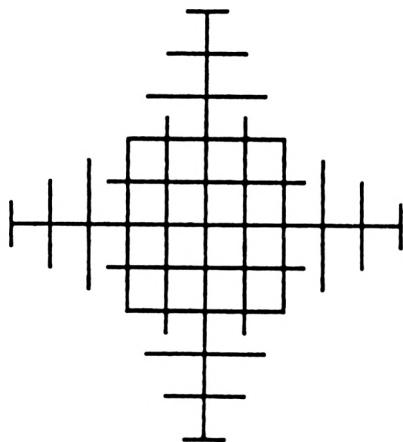
PROJECT 2 Write your name on the screen, make a note of the commands and put them into a procedure. Here's a possibility:

ANDREW

PROJECT 3 Try drawing this tree:



PROJECT 4 Make your Christmas tree instructions into a procedure and define a new procedure to draw this snowflake:



PROJECT 5 Type in this line of commands in direct mode and then join up the dots.

```
pu fd 50 pd repeat 6 [fd 2 rt 60 pu fd  
20 pd]
```

Chapter 4

This chapter is mainly concerned with the creation of circles and curves but before looking at ‘circular’ procedures we will consider words and numbers and the ways in which these might be used to advantage. DR Logo’s command for putting text on the screen is:

PRINT (pr)

This command causes Logo to print any following information onto the screen. It is a very versatile command and has many varieties which will be introduced as needed by the programs.

To try this out, type in:

```
cs  
pr [this is a box]
```

the text inside the square brackets is printed immediately below the command.

Now try this use the space bar!

```
pr [       box]
```

You will note that the spaces inside the square brackets have been ignored and the word ‘box’ appears below the command in the text area of the screen.

One other thing to notice here is that none of the required text is being printed on the graphics screen. It’s all happening in the four text lines below the graphics screen. Try typing these commands in direct mode:

```
fs  
pr [abc]
```

Nothing! Better type splitscreen (ss) to get the text back.

Now this,

```
pr []
```

You will notice that a line has been left before the prompt is printed.

Now try this in direct mode.

```
repeat 4 [pr []]
```

and you have successfully cleared the screen of all text. Are you beginning to see how the print cursor operates? Well, after obeying each print command it moves down a line before obeying the next command so if you say ‘print nothing’ by typing in

```
pr []
```

it does just that.

Therefore, ‘repeat 4 [pr []]’ will give you four clear lines.

The same can be done with ‘pr “ ”’ which works in exactly the same way. Remember the correct spacing is vitally important.

Try

```
pr "MAN
```

Now try

```
pr "
```

To get capital letters, you must hold down <SHIFT> whilst typing or use <CAPS LOCK> – i.e. tap <CAPS LOCK> once to put the computer in ‘capitals’ mode before typing ‘MAN’ and tap it again to go back to lower-case mode.

How about

```
repeat 3 [pr "]
```

If you use <CAPS LOCK>, remember to press it again when you’ve finished because Logo commands must be typed in small letters.

Try a multistatement ‘print’ line:

```
pr "hello pr [my name] pr "is pr "dollop
```

You will find...

```
hello  
my name  
is  
dollop
```

is printed on the screen – very interesting, but does it have to be printed on separate lines? Can’t it be printed on one line? Your DR Logo has a command that will help you out of this difficulty:

SENTENCE (se)

This command outputs a list made up of the input objects.

Try typing:

```
pr se [HELLO MY] [NAME IS DOLLOP]
```

which gives you ‘HELLO MY NAME IS DOLLOP’. Alternatively, you could try:

```
pr se "HELLO [MY NAME IS DOLLOP]
```

which will give you the same output; or even

```
pr (se "HELLO "MY "NAME "IS "DOLLOP)
```

The round brackets ensure that Logo knows where the sentence begins and ends. Try it without them!

You will have noticed that your computer will respond to the command:

```
pr [THE CAT IS SAD]
```

with:

```
THE CAT IS SAD
```

But try ‘pr “THE MAN IS BIG” ’ and it will only print the first string ‘THE’ and then give you an error message.

Drawing Circles

By now you are quite used to the simple command

```
fd 60
```

Another way of writing this command might be

```
repeat 1 [fd 60 rt 0]
```

Try it and see. You will notice that the result of this command is the same as the simple ‘fd 60’ command. We are asking the turtle to move forward 60 units with no change of direction.

Another way of asking for the same thing would be to say

```
repeat 3 [fd 20 rt 0]
```

Try it and you will see that the turtle

- 1) moves forward 20 units
- 2) reads the turn 'rt' command and obeys by not turning
- 3) moves forward
- 4) reads 'rt 0'
- 5) moves forward
- 6) reads 'rt 0'
- 7) stops

If you wanted to bend the straight line how would you do it? The answer of course is to tell the turtle to do a little turn to the right or left as it draws the line. The new command sequence now looks like this:

```
repeat 3 [fd 20 rt 10]
```



FIGURE 4.1

Clear the screen and try it.

Your screen now looks like that shown above in Figure 4.1; the turtle created a curve from three straight lines!

Let's bend it a bit more, but extending the overall line length to 90 units. Try:

```
repeat 10 [fd 9 rt 10]
```

Degree of change of direction

Well, you've certainly bent that line. Let's try and bend it over to create half a circle like the one shown below in Figure 4.2:

```
repeat 18 [fd 5 rt 10]
```



FIGURE 4.2

What we have to do is increase the number of little lines and decrease their length. Let's try:

```
cs  
repeat 30 [fd 3 rt 10]
```

You can watch the turtle go forward 3, turn right 10 degrees, go forward 3, right 10 and so on. Lo and behold the circle is nearly complete.

Now if we look at the last command we can see that the turtle has turned right through 10 degrees 30 times.

That's $30 \times 10 = 300$ Degrees

If you remember you found out in Chapter Two that there are 360 degrees (4×90 degrees) in a complete turn, so if we make our turtle turn through 12 degrees 30 times ($30 \times 12 = 360$) it should turn through a complete circle!

Try:

```
cs  
repeat 30 [fd 3 rt 12]
```

Figure 4.3 below shows a number of closed shapes made up from a number of straight lines – a mathematician would call them 'polygons'. As the number of sides increases, the shape tends to look progressively more round.

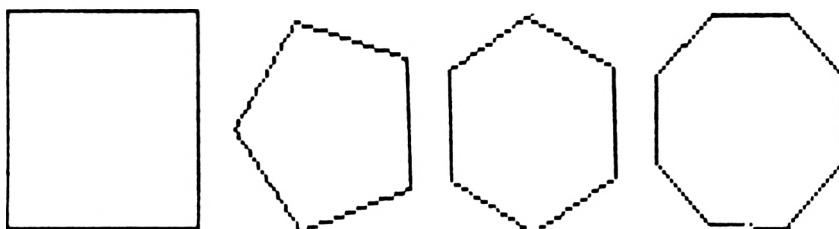


FIGURE 4.3

If we want the turtle to draw one of these, say the hexagon, we have to ask it to: move forward a number of screen units; do a small turn to the right; go forward again; do another small turn to the right and so on until the ‘circle’ is complete. Altogether, the turtle will do six turns.

But, HOW FAR DO WE ASK THE TURTLE TO TURN?

The answer to that is a total of 360 degrees, the number of degrees in a whole turn. However, for a hexagon, at each corner the turtle will turn one sixth of this, that is 360 divided by 6 (the number of sides). Thus at each turn we can tell the turtle to turn 60 degrees ($360/6$) or, better still, we can leave the computer to work out the sum and simply say ‘turn right an angle of 360 degrees divided by 6’ i.e. ‘rt $360/6$ ’.

This character, ‘/’, is 3rd from the right on the bottom row of the keyboard and means ‘divided by’.

So we want the turtle to move forward say 15 screen units then do a right turn of $360/6$ degrees and to do this 6 times.

To see if it works, clear the screen and try this in direct mode:

```
repeat 6 [fd 15 rt 360/6]
```

So far so good, this has drawn a hexagon, but how can we make this into a circle? The answer is to increase the number of sides, thus reducing the size of the turns. To do this it is only really necessary to decrease the angle each time round and then increase the number of times the repeat is made. Try:

```
cs  
repeat 20 [fd 15 rt 360/20]
```

That’s more like a circle but has grown considerably. This is because we have kept the ‘forward’ command at 15 screen units.

Let’s reduce this length and see what happens. Try:

```
repeat 20 [fd 7 rt 360/20]
```

Your screen should now look as shown below in Figure 4.4.

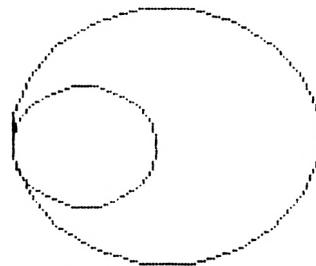


FIGURE 4.4

This little exercise demonstrates how the size and accuracy of a circle drawn in this manner will depend upon:

- 1) the number of sides
- 2) the length of the sides

```
repeat 60 [fd 3 rt 360/60]
```

Try out the above to see what a 60-sided ‘circle’ looks like.

Then try this:

```
repeat 120 [fd 2 rt 360/120]
```

Yet again quite a nice circle, but, so far we’ve no real control over its actual size.

Just to see really clearly what the problem is, try this command sequence:

```
cs  
repeat 360 [fd 1 rt 360/360]
```

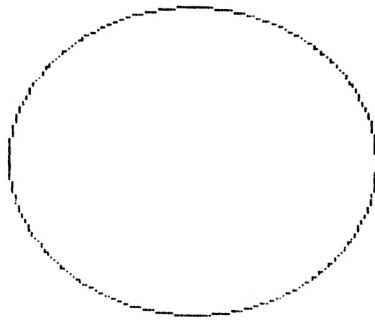


FIGURE 4.5

The turtle is creating a 360 sided polygon, each of the sides being 1 screen unit long.

The total distance around the circle will be 360×1 screen units, 360 in all, the distance being the ‘circumference’ of the circle.

Theory

Don’t worry too much if you can’t follow the algebra below, but do have a go! The results will be tabulated for future reference anyway.

If we wanted to work out the circumference of any circle we would use the formula

$$\text{Circumference} = 3.142 \times \text{Diameter}$$

$$\text{or } C = 3.142 \times D$$

You can see that the diameter of a circle is twice its radius and so we can re-write:

$$C = 3.142 \times D$$

as $C = 3.142 \times 2 \times \text{Radius } (R)$

Now, if $C = 3.142 \times 2 \times R$

then $R = \frac{C}{2 \times 3.142}$

Now, we know that the circumference of the large circle is 360 units, because the turtle took 360 steps of length 1 to draw it.

Therefore $R = \frac{360}{2 \times 3.142}$

$$R = 57.28$$

It looks as though the circle just completed is about 115 screen units in diameter!

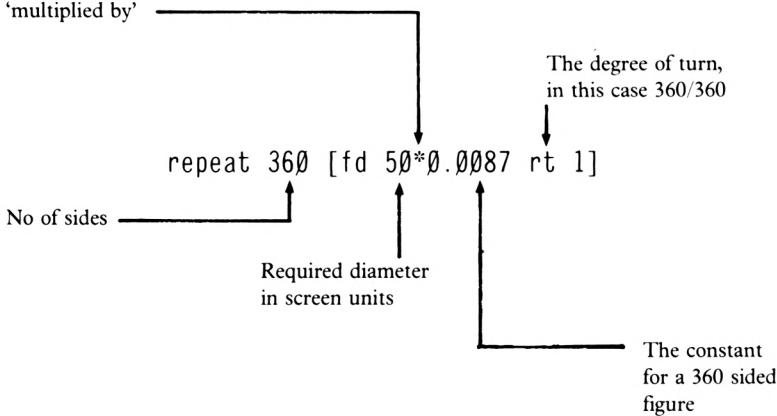
Now as has already been said, the length of each of the sides of a circle decides its diameter so let's look at the relationship between this length and the diameter we calculated.

Side length $\frac{1}{115} = 0.0087$
Circle diameter 115

Now the figure of 0.0087 is a constant for any circle with 360 sides and, therefore, we can use it in our description of any circle of any diameter with 360 sides.

Clear the screen and try the following for a circle 50 screen units in diameter.

Note "*" means
'multiplied by'



It's taken a long time, but there is a circle of 50 screen units diameter. You can test it by typing in direct mode:

```
rt 90  
fd 50
```

These commands should cause the turtle to travel from one side of the circle to a point on the other side if it really does have a diameter of 50 screen units.

Now 0.0087 is fine for a 360 sided polygon but these take a long time to draw and the value of the constant will depend upon the number of sides required.

If we want to draw similarly sized circles but with fewer sides we will have to re-create the constant.

For example

```
repeat 60 [fd 50*.052 rt 360/60]
```

↑	↑	↑	↑
No of sides	Required diameter in screen units	Constant	Size of turn (6 degrees)

All that we have to do each time is to calculate the size of the constant.

Let's calculate the constant for the last example.

Now we know we want it 50 units diameter, so we say.

$$3.142 \times 50 = \text{Circumference } C$$

$$= 157.1$$

Divide the circumference by the number of sides and we will have the length of each side.d

$$\frac{157.1}{60} = 2.618 = \text{Side length}$$

dividing by the diameter:

$$\text{constant} = \frac{2.618}{50}$$

$$= 0.052$$

We can now write our command:

```
repeat 60 [fd 2.618 rt 6]
```

Which is the same as

```
repeat 60 [fd 50*0.052 rt 360/60]
```

Now to try and calculate the constant for a 30 sided 50 unit diameter circle, for example:

circumference = $3.142 * 50 = 157.1$ units

$$\frac{157.1}{30} = 5.237 \text{ units (length of each side)}$$

Therefore

```
repeat 30 [fd 5.237 rt 12]
```

should once again give a circle of 50 screen units diameter.

In terms of visual quality there is very little to choose between the three 50 screen unit diameter circles, but they do become progressively faster to complete and a 30 sided polygon as a circle is a very good approximation to a circle.

So to draw a circle of exactly the diameter required, follow these rules:

- 1) Decide its diameter (say 80 screen units)
- 2) Decide how many 'sides' it is going to have.

e.g.

```
repeat 30
```

- 3) Then calculate the constant:

$$\frac{3.142}{30} \times 80 = 8.38$$

This is the side length

- 4) Write out the procedure

e.g.

```
to circle
repeat 30 [fd 8.38 rt 360/30]
end
```

The really useful feature of this form of procedure is that it enables you to draw bits of circles anywhere on the screen and any size, just by

- 1) placing the turtle where you want it to start,
and by
- 2) limiting the number of repeat cycles to give you the shape you want.

Thus, to get a third of a circle of radius 40 units:

```
repeat 10 [fd 8.38 rt 360/30]
```

Just in case you don't feel like working out the necessary figures to give you circles of precisely the size you want, here is a table for the number of sides giving line length for a range of different circles of different sides and diameters.

Diameter in screen units (I)
and circumference (II)

I	10	20	30	40	50	70	80	100	157
II	31.42	62.84	94.26	125.68	157.1	219.94	251.36	314.2	471.3
No of sides									
10	3.142	6.284	9.426	12.568	15.71	21.99	25.136	31.42	47.13
20	1.571	3.142	4.713	6.284	7.855	10.997	12.568	15.71	23.565
30	1.05	2.095	3.142	4.19	5.234	7.331	8.378	10.473	15.71
40	0.785	1.571	2.37	3.142	3.93	5.498	6.284	7.855	11.782
60	0.524	1.05	1.571	2.095	2.618	3.665	4.189	5.237	7.855
100	0.314	0.6284	0.9426	1.257	1.57	2.199	2.514	3.142	4.713
200	0.16	0.314	0.4713	0.628	0.785	1.099	1.257	1.571	2.3565
300	0.09	0.175	0.26	0.35	0.436	0.61	0.698	0.872	1.309

If for example you wanted to draw a 30 diameter (15 radius) circle with 40 sides, just look at the line length in the table.

2.37

```
repeat 40 [fd 2.37 rt 360/40]
```

Remember that you can always draw circles in the opposite direction by substituting a left command instead of a right.

Let's devise a procedure to draw the shape shown below in Figure 4.6.

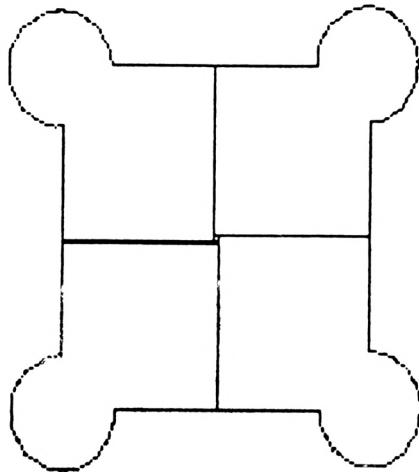


FIGURE 4.6

To produce the circles we shall use 'repeat 60 [fd 1.05 rt 360/60]'. If you look closely at the shape you will see that it is symmetrical, that is, it doesn't matter if you turn it upside down or left to right, the shape looks the same – that is, you can't tell which way up it is. This being the case, all we have to do is define a procedure to draw Figure 4.7 shown below:

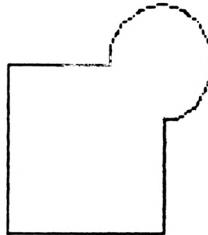


FIGURE 4.7

To get the final desired result, we can repeat it four times. The turtle's starting position is indicated in the above Figure.

The command sequence will be:

```
fd 60
rt 90
fd 40
lt 90
Draw the three quarters of a circle
lt 90
fd 40
rt 90
fd 60
rt 90
end
```

The circle order requires a 20 unit diameter circle and one has been chosen with 60 sides and a line length of 1.05 units.

The procedure 'circle' would be

```
repeat 60 [fd 1.05 rt 360/60]
```

However we only want three quarters of a circle and so only three quarters of the 60 sides are needed. Therefore the procedure will be as follows:

```
to circle
repeat 45 [fd 1.05 rt 360/60]
end
```

↑ ↑

This will
give 3/4 of
a circle

Remember to put the no of
sides for the completed
circle in here.

Now we can write the procedure – let's call it 'shape'

```
to shape
fd 30 rt 90 fd 20 lt 90 circle lt 90
fd 20 rt 90 fd 30 rt 90
end
```

Run this procedure, just to check it out so far.

It works! Now to nest this into a final procedure to get the final shape.

```
to finish
repeat 4 [shape rt 90]
end
```

This tells the turtle to draw the shape, turn through 90 degrees and draw the shape again, etc. Type ‘fs’ and then ‘finish’ to see the full drawing.

The thing to do now of course, if you have the time, is to rotate this procedure. Try

```
to rotate
repeat 3 [finish rt 60]
end
```

You now have the power to mix curves, semi-circles and straight lines in almost any combination. Let’s consider the shape shown below in Figure 4.8.

This shape is quite complex and will make use of all the skills acquired so far. It also happens to be part of a pattern on the wallpaper in the room in which I am currently working.

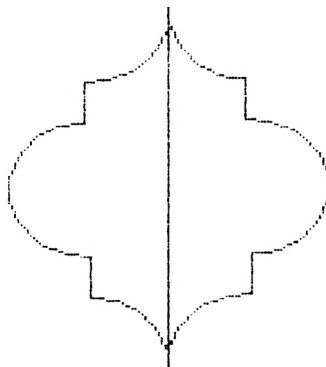


FIGURE 4.8

As you can see, the shape is symmetrical about the vertical axis. This means that all we have to do is to draw half of it in such a way that it can be repeated. The route chosen is outlined below.

Now, using this plan the turtle finishes up in the centre of the graphics screen and if you turn the page upside down you’ll see that if exactly the same route is followed the figure will be completed by following exactly the procedure as before.

The shape and dimensions have been carefully chosen to facilitate accurate procedure commands. Once again it is worth stressing the point that accurate planning and careful thought will result in first-time success.

to shape1	
fd 80	This moves the turtle from “HOME”
repeat 15 [bk 2.618 lt 6]	This draws a quarter of a circle 50 units diameter. Remember we are drawing backwards, so we are turning to the left at each turn.
lt 90	Turns to the left ready for the small straight line.
fd 15	Draws a line 15 units long.
lt 90	Turns left ready to start the semi-circle.
repeat 30 [fd 2.618 rt 6]	This line draws the semi-circle. (You can look up the figure provided in the table.)
lt 90	Turns left ready to draw a 15 unit line.
fd 15	Draws line.
rt 90	Ready to start last curve
repeat 15 [fd 2.618 lt 6]	Draws a quarter of a circle.
bk 76	Returns the turtle home ready to start second half.
end	end

To finish the shape you could say:

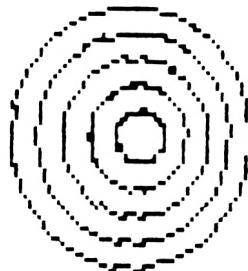
```
to finish1
repeat 2 [shape1]
end
```

And you could go one further and rotate the shape by saying

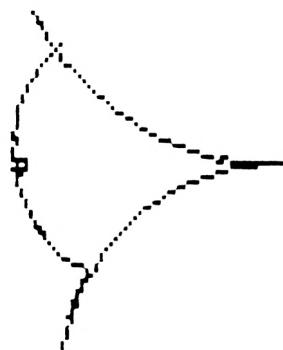
```
to rotate1
repeat 4 [finish1 rt 45]
end
```

Projects

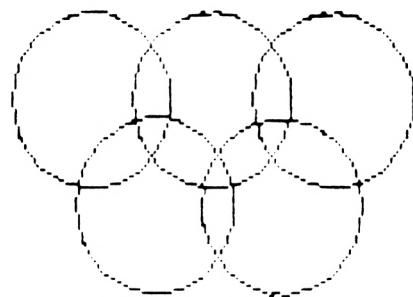
PROJECT 1 Define a procedure to draw a series of concentric circles.



PROJECT 2 Create a procedure to sketch this eye:



PROJECT 3 Try these linked circles:



Chapter 5

If the computer is not already in split-screen mode, put it into split screen by typing 'ss'. As you may be aware, the split screen measures approximately 320 x 640 screen units. Have you found out how long a 'screen unit' is? If not, try these direct commands:

```
cs  
ht  
fd 1
```

And there it is; one screen unit. Now try this:

```
pu  
fd 10  
pd  
fd 0.1
```

And there is a small dot representing a tenth of a screen unit.

Can you think of a procedure to produce a broken line like the one shown below in Figure 5.1?



FIGURE 5.1

It might look like this:

```
to chain
pd
fd 10
pu
fd 6
pd
fd 10
end
```

You might then try this in direct mode:

```
repeat 12 [chain]
```

It's quite a nice effect and one that we could well make use of.

Let's try another version:

```
to dot1
repeat 3 [pd fd 6 pu fd 6]
end
```

There is also a Logo command which will put a dot at a specific screen location without moving the Turtle.

The command is:

DOT (dot)

This command will place a dot in the current pencolour at the coordinate position specified by the instruction list. e.g.

```
dot [30 50]
```

Project

Devise a procedure for a rotated triangle using dotted lines. If you wish you could follow the procedure for a rotated square outlined below.

Once again, planning is important and you must remember that the ‘dot1’ procedure is about 10 screen units long and use it accordingly.

```
to box
repeat 4 [dot1]
rt 90
end
```

and now

```
to link
repeat 12 [box rt 30]
end
```

The procedure is slow because your computer is having to think ‘pen up, forward 6, pen down, forward 6’ etc. for every little line and gap in the procedure. However, the result is quite pleasing and you might like to make use of this technique in other procedures.

It’s a bit of a long-winded process when you wish to move the turtle to a particular screen location using the techniques developed so far. It is necessary to turn the turtle, command it to move a certain number of screen units after first having calculated the forward, back or angular requirements for a particular move. To help us in this process, Logo contains a powerful command called ‘setpos’ which enables the turtle to be moved from wherever it happens to be to any other position on the screen with very little calculation. So that you can make use of the ‘setpos’ command we have re-drawn the screen in the form of a grid as shown below. Notice that the height of the screen is called Y and the length of the screen is called X. This is a mathematical convention and you will soon get used to it.

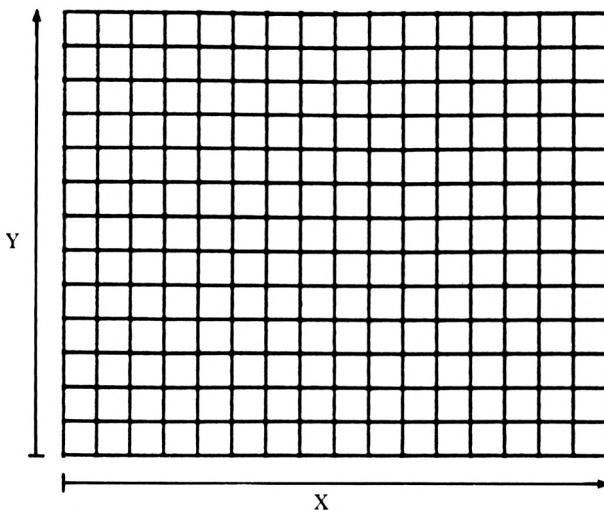


FIGURE 5.2

SETPOS (setpos)

This is a command, short for setposition, that moves the turtle to a specific point on the graphics screen defined by the instruction.

As you know, after you have cleared the screen, the turtle takes up a central position. That is, halfway along the X axis and halfway up the Y axis. This is a point where, in terms of screen units, $Y=0$ and $X=0$. This technique is making use of an imaginary grid. In the grid drawn above, each square represents 50 by 50 screen units.

Suppose that we wanted to locate the turtle 150 units further towards the top of the screen. You could say

```
st fd 150
```

Now the current position of the turtle in terms of X and Y is where $X=0$ and $Y=150$. Now try the command

```
setpos [0 150]
```

The turtle does not move, and so it shouldn't because it is already where it has just been told to go. The 'setpos' command differs from 'forward' and 'back' as it is an ABSOLUTE command and specifies ABSOLUTELY where the turtle should move to. 'Forward' and 'back' are RELATIVE commands as they move the turtle to a position that is RELATIVE to its previous position.

Project

Starting from home move to

- (i) the top left-hand corner firstly by means of relative and then by means of absolute commands.
- (ii) the bottom right firstly by means of relative commands and then by means of absolute commands.

Now clear the screen and type:

```
setpos [50 0]
```

As one would expect, the turtle moves to the left.

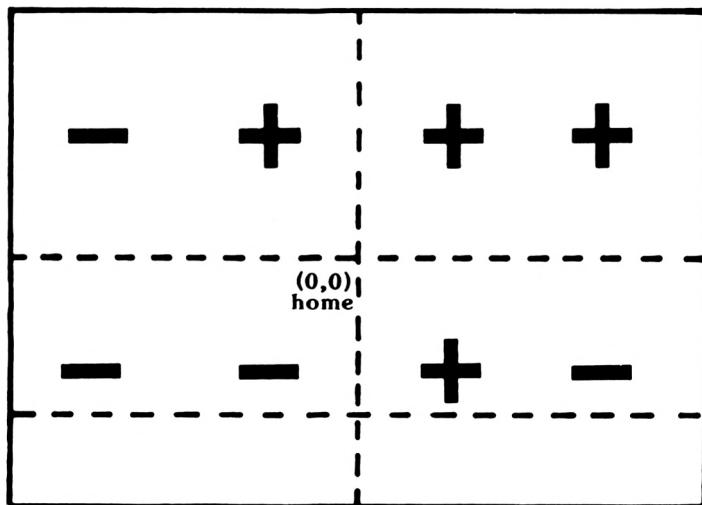


FIGURE 5.3

You will notice that the turtle's field of movement is divided into four areas, as shown by Figure 5.3. In the top right hand corner both X and Y coordinates have positive values, whereas in the bottom left hand corner they both have negative values.

Try some more commands, for example:

```
setpos [0 0]
```

This will put the turtle into the middle of the vertical Y axis and the horizontal X axis.

`setpos [0 -115]`

This will take the turtle to the bottom limit of the split graphics screen along the Y axis.

`setpos [0 200]`

This will take the turtle close to the top limit of the graphics screen along the Y axis in the 'positive' area.

Now let's move the turtle in an ABSOLUTE fashion along the X axis. Try these commands (after clearing the screen):

`setpos [320 0]`

This will take the turtle to the far right of the graphics screen along the X axis in the positive area.

`setpos [-320 0]`

Places the turtle on the left of the screen.

Note also that when using 'setpos', the turtle looks to see whether to move with 'penup' or 'pendown'.

Another feature of 'setpos' is that the turtle will take the shortest route to the position defined. This saves having to calculate precise angles through which to turn the turtle before telling it to go forward or back.

Suppose we wanted to move the turtle from its current location to the position where X=60 and Y=50 as indicated in Figure 5.4 below.

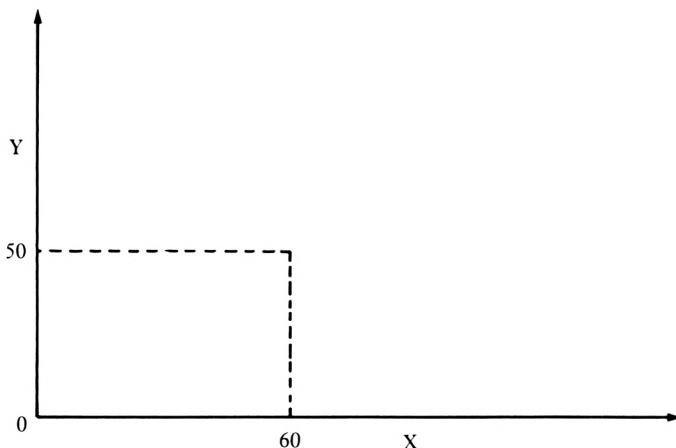


FIGURE 5.4

The command to give is

```
setpos [60 50]
```

When both column (X) and row (Y) coordinates are being defined, the descriptive numbers are separated only by a space within the square brackets. Notice also that this particular position is within the +VE, +VE region of the screen.

Try typing in the above command and notice how the turtle moves directly to that point, drawing a line as it moves. Notice also that it maintains the same heading throughout the 'setpos' move. It can thus point in one direction while moving in any other, unlike with the 'forward' and 'back' commands.

Try these commands under the definition 'trace1':

```
to trace1
  setpos [0 50]
  setpos [50 50]
  setpos [50 0]
  setpos [0 0]
end
```

Have a look at this procedure and try to draw it to find out what shape will be created before you type it in and run it.

It is generally a slightly longer typing job when we enter 'setpos' commands but if you want to draw a line or move the turtle between any points on the screen by means of 'setpos', it isn't necessary to calculate how far you want the turtle to move. Equally you don't have to work out how far the turtle has to turn before telling it to move.

Talking about 'turning' the turtle, you can use 'setpos' in conjunction with 'setheading' (below) to turn the turtle on the spot to face in any particular direction. Now you might argue that this can already be done with the 'right' and 'left' commands followed by a number and of course you would be quite correct. However, right and left commands are 'relative' commands – i.e. when the command is obeyed the turtle moves relative to its last position.

SETHEADING (seth)

This command turns the turtle in absolute terms by a specific number of degrees in a clockwise direction.

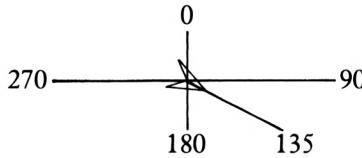
For example

```
seth 50
```

Now:

```
cs rt 90 lt 45 rt 90
```

leaves the turtle facing 'South East' as shown below.



The heading in Absolute terms is +135. This means that starting at 'Due North' which is the turtle's 'home' position, the turtle counts 135 degrees to the right and points in that direction. Clear your screen and type:

```
seth 135
```

now try

```
seth 90
```

and

```
seth 270
```

and

```
seth 360
```

and

```
seth 0
```

You will of course notice that 'seth 0' and 'seth 360' give you the same heading.

When you use positive numbers, as we have above, the turtle turns to the right in a clockwise direction and sets its heading in the required direction. If you wanted to you could use negative numbers in conjunction with setheading, in which case the turtle will move in an anticlockwise direction before setting in the required direction.

Thus

```
seth 270
```

will give exactly the same heading direction as

```
seth -90
```

SETSPLIT (setsplit)

This command sets the number of lines in the splitscreen's text window.

e.g.

```
setsplit 15
```

When the above command is used and the associated number varied, the relative size of the graphics and text windows (screens) may be adjusted. When you have finished experimenting type

```
setsplit 5
```

and normal service will be resumed.

Now that you have gained a lot more freedom of movement and control of the turtle, we can have a look at some 'procedural tools' which might be of use to you as you work with Logo.

Let's imagine that you wanted to display the contents of a particular procedure. All you have to do is type:

```
cs  
po "box
```

PRINT OUT (po)

This will display the contents of a procedure or of the given procedure list.

e.g.

```
po "trace1
```

Now try:

```
ts po [trace1 link]
```

and your computer does exactly as it's told, assuming you have the procedures 'trace1' and 'link' in memory.

Disk Formatting

Well, as you may well already know you can save your Logo procedures on a disc. Now in order to get a new disc ready to receive your procedures you will have to FORMAT it. When a disc is FORMATTED, it is divided up into original tracks and sectors ready to receive data. If your blank disc has not been FORMATTED, just follow the following procedures.

- 1) Reset the computer by pressing <SHIFT>, <CTRL> and <ESC> simultaneously. Then insert your system/utilities disc into the drive and type:

```
| cpm
```

- 2) When you see the message 'A>', type:

```
format
```

and this message will be displayed after a short pause:

Please insert disc to be formatted into drive A then
press any key

- 3) Now assuming you are using a single disc drive take the system/utilities disc out of the drive and replace it with the side of the disc to be formatted uppermost. Make sure that this disc is not write protected – i.e. make sure that the plastic tab covers the little hole in the top left corner of the disc.

- 4) Hit any key and the formatting process will begin. During the formatting process, the 'track numbers' are displayed on the screen as it does it.
- 5) You will then get the message:

Do you want to format another disc (Y/N):

to which you answer by typing Y and then turn your disc over to side 'B' and follow instructions by hitting any key, again making sure that the disc is not write-protected so the computer can 'write' on it.

- 6) You are now ready to answer 'N' to the currently displayed question.

Now replace your formatted disc with your Logo disc and type

logo

If you prefer the original colour scheme, then the easiest thing for now is to do a complete reset of the computer (<SHIFT>-<CTRL>-<ESC>) and re-load Logo in the normal way.

Having formatted your disc you are now ready to save your procedures. There is however a problem: since you switched off your computer you will have to type them in again! This should not take too long and after a short while, your procedure list might look like this:

```
to chain  
to dot1  
to box  
to link  
to trace1
```

With your formatted disc in the drive you are now ready to save your procedures. Now the easiest way to do this is to say

save "lot1

any name will do here

SAVE (save)

This will save the specified procedure or procedures onto disc. It will save the entire contents of the workspace onto the named file.

If you now type:

```
dir
```

DIRECTORY (dir)

This outputs a list of the Logo file names on the disc currently in the disc drive.

The disc drive will hum momentarily and you should then see:

```
[LOT1]
```

which indicates that file LOT1 has been saved on disc.

Now erase your procedures by saying:

```
er "chain  
er "dot1  
er "box  
er "link  
er "trace1
```

You are now ready to reload your procedures from disc by saying:

LOAD (load)

This reads the named file from disc into your computer's workspace.

```
load "lot1
```

and as you do this you will receive the message:

```
chain defined  
dotl defined  
link defined  
tracel defined
```

Now if you wanted to make a change in one of the procedures you can use the 'edit' facility as usual, so type:

```
ed "mat
```

and then change 'tracel' so that it looks like this:

```
to tracel  
setpos [0 50]  
setpos [50 50]  
setpos [0 0]  
end
```

If you now want to put this and all the other procedures back onto disc, you might try

```
save "lot1
```

You will receive a message saying:

```
File lot1 already exists  
?
```

Which on the face of it means that you can't alter any procedures saved on disc. However, all you have to do is create a new file name.

Now if you wanted to make sure that nothing could overwrite your procedures you could move the 'write protect' tab on the disc to uncover the small hole.

If at any time you want to stop using Logo, instead of switching the computer off, you can just type:

```
bye
```

BYE (bye)

This command enables the user to exit the current version of DR Logo.

Having learned how to save, edit and reload procedures, let's now write a procedure for a clock face using turtle graphics.

First of all we need to draw a circle, say 100 units in diameter. A 60-sided circle should suit our purposes and so the first part of the procedure will be 'repeat 60'.

The circumference of the circle will be $3.142 \times 100 = 314.2$ units. Dividing the circumference by the number of sides gives us the side length:

$$\frac{314.2}{60} = 5.24$$

The procedure for the inner circle will be:

```
to inner
pu
setpos [-3 0]
fd 50
pd
rt 90
ht
repeat 60 [fd 5.24 rt 360/60]
end
```

Similarly we can calculate the line length for the other circle of 140 units in diameter:

$$140 \times 3.142 = 440 \text{ Circumference}$$

$440 = 7.33$ Side length of a 140 units
diameter circle with 60 sides.

Our procedure for the outer circle therefore could be:

```
to outer
pu
setpos [-3 0]
fd 70
pd
rt 90
ht
repeat 60 [fd 7.33 rt 360/60]
end
```

Now we can nest these circles into a procedure called, say, ‘circles’.

```
to circles
inner
outer
end
```

We can now put in the small radial lines. Follow the procedure below. If you want to make sure that you really understand it, work out what it should do on some squared paper.

```
to radial
pu setpos [0 0]
repeat 12 [pu fd 70 pd fd 10 pu bk 80
rt 30]
end
```

If you type ‘fs’ and run ‘circles’ and ‘radial’, the clock face should now look like the one shown below in Figure 5.5.

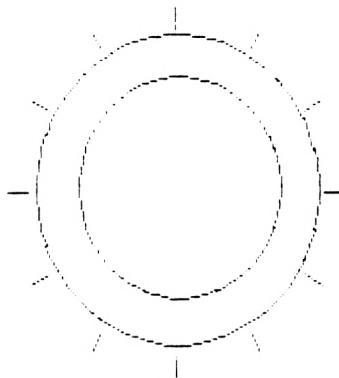


FIGURE 5.5

Having completed that rather elegant clock face, we must turn it into a real, working clock. Other new Logo commands are needed first, however, and the whole operation will be completed in Chapter 9.

Let's start off by outlining a problem. Suppose we wanted to create a series of similar shapes, but of different dimensions like the rectangles one inside the other shown below.

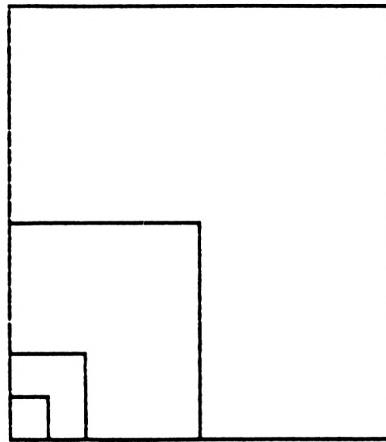


FIGURE 5.6

The procedure would be:

```
to box
pu
ht
fd 10
rt 90
fd 20      etc.
```

You would expect to have to describe a procedure for each of the rectangles and nest these in a final procedure. To help overcome these sometimes long-winded procedures, a process called 'passing parameters' can be used.

All this means is that we can design a procedure to draw a shape, but instead of deciding the size of the shape we can use a word, for example ':scale'. Whenever the procedure is called, the procedure name is typed in followed by a space and then a number. The procedure will then be executed using that number to determine the size of the shape drawn. By this means, we can scale standard procedures up or down.

If you want to use this facility, you must define the procedure in the usual way, followed by '<SPACE>:scale'.

For example type:

```
to box :scale
```

Now, you can type

```
repeat 4 [fd :scale rt 90]  
end
```

Now when you want to run your ‘box’ procedure, you simply type in:

```
box any number
```

So to test this type:

```
box 10
```

next try

```
box 20
```

then

```
box 50
```

and

```
box 80
```

Your screen now looks like the above Figure 5.6.

Let’s try combining some set procedures with the ‘box :scale’ procedure:

```
to draw  
pu  
setpos [30 30]  
pd  
box 20  
pu  
setpos [-30 -30]  
pd  
rt 45  
box 50  
end
```

Any shape, any size, any screen location, any direction. The choice of visual options is beginning to widen rapidly and even accelerate.

What happens if we use a scale factor twice in a procedure? Type in:

```
to rectangle :scale
repeat 4 [fd :scale rt :scale]
end
```

Having stored that procedure type:

```
rectangle 30
```

and you will end up with a drawing that looks like the screen shown below in Figure 5.7.

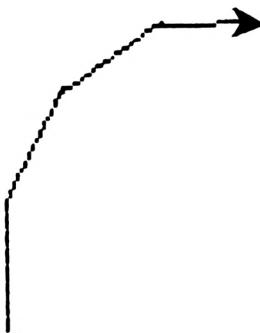


FIGURE 5.7

That is, 'fd 30, rt 30' 4 times. This will be needed in due course.

Let's try to 'scale' a circular procedure: an example of a typical command to draw a circle is:

```
repeat 60 [fd 2.618 rt 6]
```

Clear the screen and run it in direct mode to see how it works.

It should draw a circle 50 units in diameter.

Let's try to scale this command. Type:

```
to circle :scale
repeat 60 [fd :scale rt 6]
end
```

Now type 'circle 2'

Now try

```
ht  
circle 3
```

and

```
circle 4
```

and

```
circle 5
```

Your screen should now look like that shown below in Figure 5.8.

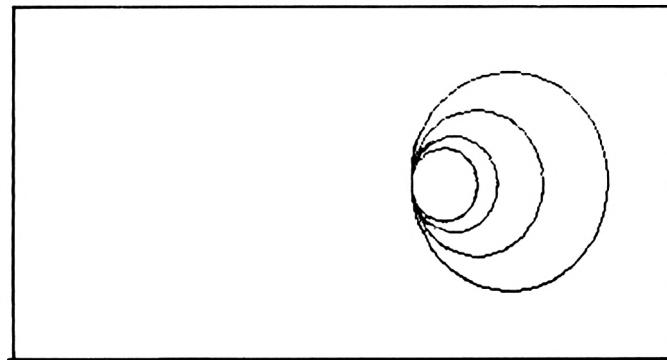


FIGURE 5.8

Now try

```
to shape  
ht  
circle 2  
circle 3  
circle 4  
circle 5  
end
```

and

```
to shape2
ht
repeat 4 [shape rt 90]
end
```

Experiment a bit more. If you like you can edit the procedure ‘circle :scale’ or you can create new ones, it’s up to you.

A new procedure will now be created to draw a short curve:

```
to a :scale
repeat 40 [fd :scale lt :scale]
end
```

Then type in

a 3

What has happened is that the turtle has moved forward 3 and turned to the left 3 degrees after each move and in all has turned through 120 degrees (40×3).

As the curve appears quite smooth let’s make it more of a curve by reducing the repeat commands, maintaining the angle of turn, but increasing the movement forward.

Thus, we might have

```
to b :scale
repeat 10 [fd :scale lt 6]
end
```

Now, clear the screen and type

b 5

Now which way is the turtle pointing? Type ‘st’ to find out.

On its travels it has gone through 60 degrees of turn (10×6). Therefore, if we want it to point down we can give it the order:

lt 120

Then try this again:

b 5

You have drawn a leaf. Imagine having a procedure to draw a leaf that you could repeat, or rotate to create a flower, or even scale up or down.

Try this:

```
to flower
ht
repeat 3 [b 5 lt 120 b 5]
end
```

And then

```
to rose
ht
repeat 4 [flower rt 30]
end
```

Then type 'rose'. This should produce the drawing of a flower as shown below in Figure 5.9.

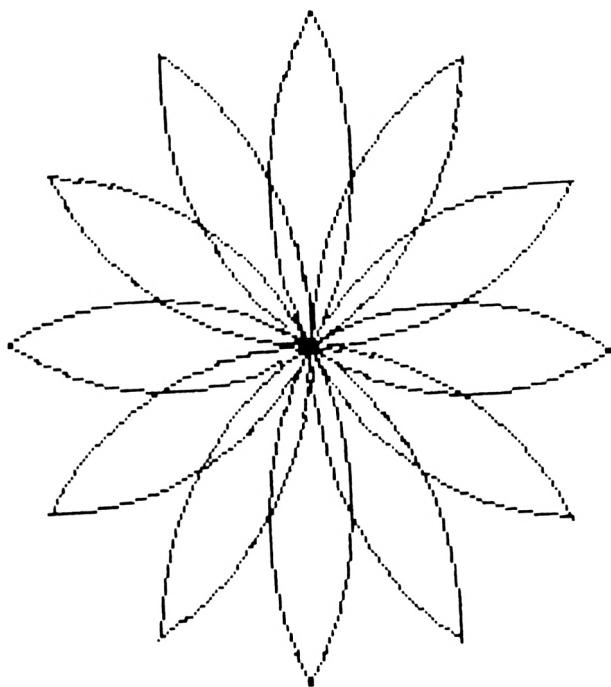


FIGURE 5.9

Why don't you design one with fatter petals?

Whilst you can't scale the size of the flower up or down, you can make adjustments in the 'b :scale' procedure which decides the length and degree of bend of your curve, which in turn will define the leaf shape.

If you wanted to you might like to try 'passing a parameter' like this:

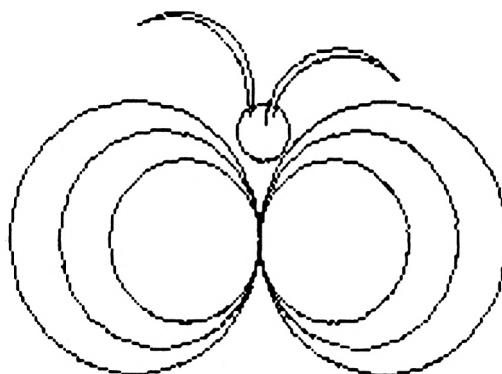
```
to rise :step
repeat :step [pd fd 10 rt 90 fd 10 lt
90]
end
```

and then say:

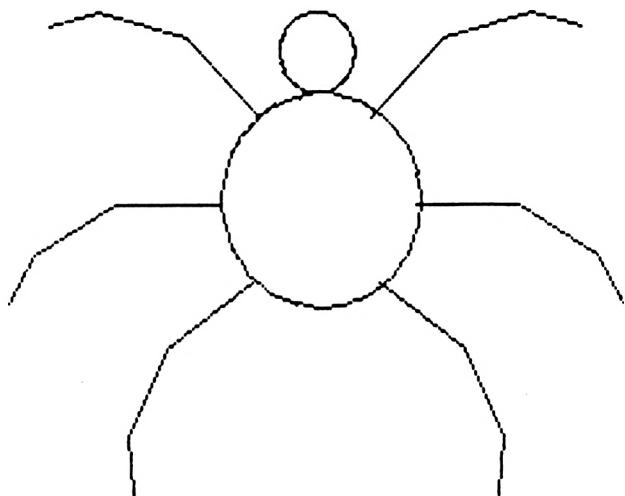
rise 6

Projects

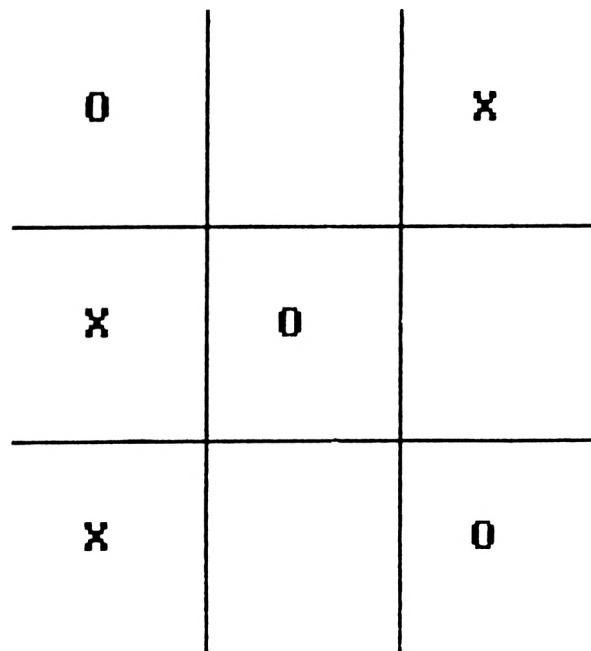
PROJECT 1 Draw this butterfly:
(remember to pass a parameter when drawing the wings.)



PROJECT 2 Draw this six-legged beast.



PROJECT 3 Reproduce this game format using text and graphics.



Possible programs to create these project shapes are given in the solutions chapter.

Chapter 6

Special Effects

Let's think of a procedure to draw an equilateral triangle, that is a triangle where all the sides are the same length and all the internal angles are 60 degrees. See Figure 6.1 below:

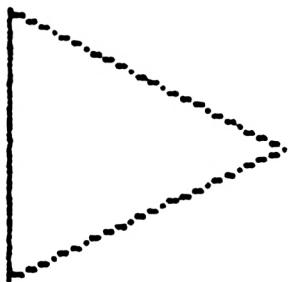


FIGURE 6.1

Here's an equilateral triangle with the turtle shown in its start position the centre of the screen. Let's define a procedure 't':

```
to t
repeat 3 [fd 60 rt 120]
end
```

Now to create a procedure to repeat this triangle six times to form the hexagon in Figure 6.2 as shown below:

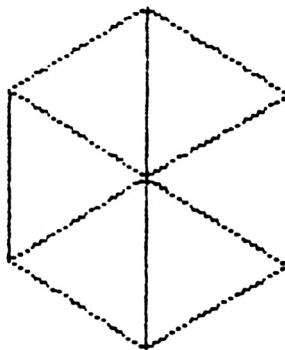


FIGURE 6.2

```
to hex
ht
repeat 6 [rt 60 t]
end
```

Clear the screen and type ‘hex’.

Now if you look closely at the hexagon it could be a transparent cube. With a new command we can make our pattern look like a square block:

PENERASE (pe)

This is a Logo command that paints over lines and in effect appears to rub them out.

In direct mode type:

pe

Now type

fd 60

and the turtle has taken out the line in front of it for the distance requested. Now to rub out the other two lines to make the cube look more like the real thing. The lines to remove are, in fact, the lines at the ‘rear’ of the cube: i.e. the ‘hidden lines’. Once the erasing has been done, the ‘penerase’ command can be cancelled by means of a ‘penup’ or ‘pendown’ command. Type:

```
bk 60
rt 120
fd 60
bk 60
rt 120
fd 60
bk 60
pd
ht
```

This leaves Figure 6.3 as shown below:

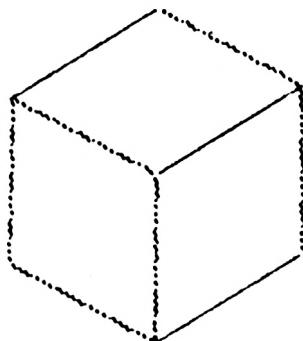


FIGURE 6.3

So to define the cube, we could say

```
to cube
hex
pe
repeat 3 [fd 60 bk 60 rt 120]
pd
end
```

Up until now, most of the patterns have been based on regular polygons. Regular polygons are those where the number of sides is divisible by 2, e.g. 4, 6, 8, 20 and 32 sided figures. These have been rotated and scaled up and down, but what about 5 or 7 or 9 or 13 sided figures? Clear the screen and try the following command in direct mode.

```
repeat 5 [fd 30 rt 360/5]
```

Interesting shape! Rotate it and see the effect; try

```
to shape
repeat 5 [fd 30 rt 360/5]
end

to rotate
repeat 30 [shape rt 360/30]
end
```

Can you predict what will happen if the pentagons are scaled? Try it out with:

```
to try :scale
repeat 5 [fd :scale rt 360/5]
end
```

Now, in direct mode, type

```
fs  
cs  
try 40  
try 20  
try 10
```

And the turtle will have drawn the shape as shown below in Figure 6.4.

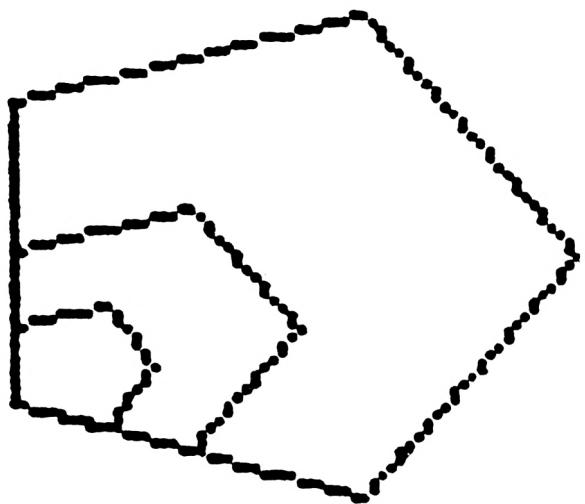


FIGURE 6.4

Now in

```
rt 108  
try 10  
try 20  
try 40
```

and again by typing:

```
rt 108 try 10 try 20 try 40
```

The resultant shape could be anything that your imagination allows. What about that triangle at the top where the pattern does not meet? Perhaps you could work out why that has happened.

Have a go at another odd-sided figure and see if we can develop a super-procedure in which the patterns meet when rotated.

Clear the screen and try:

```
to b :scale
repeat 7 [fd :scale lt 360/7]
end

b 30
```

Bit of an odd shape: it is called a heptagon. Try a nine sided figure (a nonagon or enneagon):

```
to c :scale
repeat 9 [fd :scale rt 360/9]
end

cs
c 15
rt 140
c 15
rt 140
c 15
```

This time there's no gap between the patterns, in fact there is an overlap: can you see why this is?

Exercises

- 1) How many 'rt' commands, followed by 'forward' commands, do you need to draw a rectangle?

- 2) How many 'rt 45' commands, followed by 'forward' commands, do you need to draw an octagon?

- 3) How many 'rt 60' commands, followed by 'forward' commands, do you need to draw a hexagon?

- 4) How many 'rt 1' commands, each followed by a 'fd' command, do you need to draw a 360 sided polygon, i.e. a circle?

When you have cleared the screen, try this procedure:

```
to tri  
fd 50 rt 140 fd 76 rt 140 fd 50 rt 80  
end
```

and type ‘tri’. You will end up with a closed shape that looks like that shown in Figure 6.5.

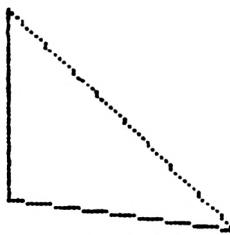


FIGURE 6.5

Now try this in direct mode:

```
cs repeat 4 [tri rt 100]
```

and the resultant patterns will overlap. This is because the total turtle trip in this case has been

$$4 \times 100 = 400 \text{ degrees}$$

instead of 360.

Well, if you look closely at the procedure which draws various ‘closed’ shapes, you will see that the sum of the internal angles always adds up to 360 degrees.

If the turtle has travelled around the shape and has ended up with the same heading it must have turned through 360 degrees.

This is the case with, for example

repeat 4 [fd 50 rt 90]	$(4 \times 90^\circ = 360^\circ =$	)
or			
repeat 8 [fd 40 rt 45]	$(8 \times 45^\circ = 360^\circ =$	)
or			
repeat 6 [fd 45 rt 60]	$(6 \times 60^\circ = 360^\circ =$	)
or			
repeat 360 [fd 0.5 rt 1]	$(360^\circ \times 1 = 360^\circ =$	)

Reduce the repeat number by one in each of the above commands and see what happens. You will see that the figure drawn isn't a closed shape. This is because the turtle has not completed a total trip of 360 degrees.

The total trip of the earlier patterns that didn't meet added up to 324 degrees, i.e. less than 360 and the the total trip of the patterns that overlapped was 420 degrees.

What if we try an odd shape? For example the one shown below in Figure 6.6.

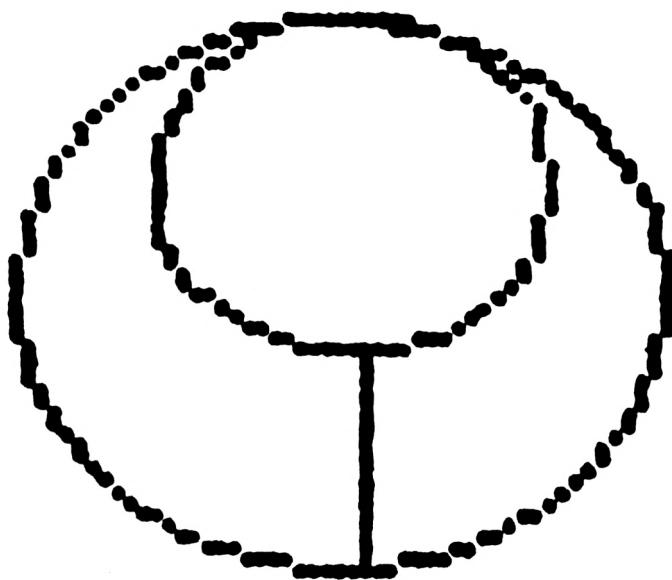


FIGURE 6.6

Follow this procedure through and make sure that you understand it, and then try it out.

```
to d
  ht fd 20 lt 90 repeat 40 [fd 2.36 rt 9]
  lt 90 fd 20 rt 90 repeat 40 [fd 3.93 rt
    9]
end
```

And now try:

```
to r
  fs
  repeat 8 [d rt 45]
end
```

What will happen when you rotate this shape? Have a look at Figure 6.7 shown below.

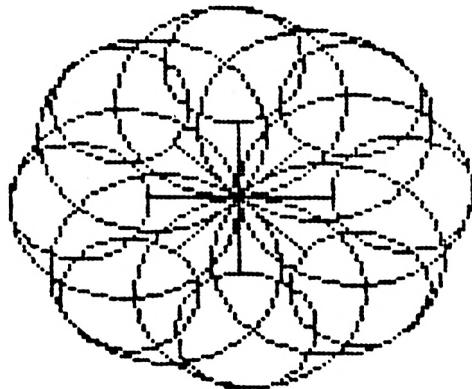


FIGURE 6.7

Now to put the ‘scale’ feature to use – to draw a ROBOT. One that looks like Figure 6.8 below:

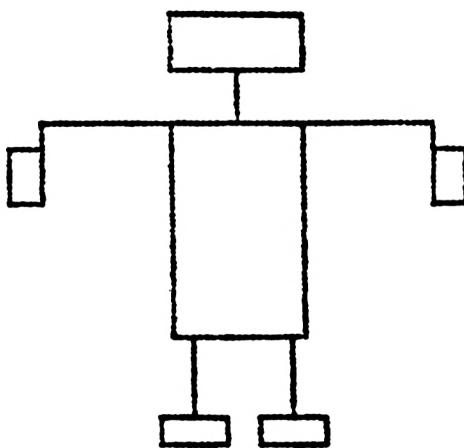


FIGURE 6.8

Assuming the head, hands, and feet are similar-shaped rectangles, the basic ‘box’ procedure could read as follows:

```
to box :scale
  rt 90 fd :scale rt 90 fd :scale rt 90
  fd :scale*2
  rt 90 fd :scale rt 90 fd :scale lt 90
end
```

You can now draw any sized rectangle of these proportions anywhere on the screen.

For example make the head ‘box 20’. Having done that, you can make up your own procedure to draw a similar robot, or follow the procedure outlined below:

```
to robot
  pu fd 80 pd box 20 pu setpos [0 0]
  fd 60 pd bk 20
  rt 90 fd 60 rt 90 fd 20 rt 90 box 10
  pu setpos [0 0] seth 0 fd 40 pd lt 90
  fd 60 lt 90 fd 20 lt 90
  box 10
  pu setpos [0 0] lt 90 fd 20 pd box 40
  pu setpos [0 0] seth 0
  bk 40 rt 90 fd 15 pd rt 90 fd 30 pu
  fd 10 pd box 10
  pu setpos [0 0] seth 0 bk 40 lt 90
  fd 15 pd lt 90 fd 30 pu
  fd 10 pd box 10 pu setpos [0 0] seth 0
end
```

Before going on to look at colour control, consider a new Logo command.

CLEAN (clean)

This command will erase the graphics screen without altering the position or heading of the turtle.

You will have noticed that when you use ‘clearscreen’, the turtle always returns to the centre point of the screen. Well if you type

```
clean
```

the graphics will disappear leaving the turtle where it is.

Colour Control

So far, all the programs developed have used the screen colours provided by DR Logo itself. However, you don’t have to be satisfied with this colour combination; DR Logo provides a command to put control back in your hands: the command ‘setpc’ sets the pen colour in which the turtle draws.

SET PEN COLOUR (setpc)

This is a Logo operation which controls the colour of the ink used by the turtle.

Keeping a pattern on the screen, try this command in direct mode:

```
setpc 1
```

This SETs PenColour to colour 1, i.e. the normal colour that you have been used to – so you will see no change.

```
setpc 2
```

This time the pen colour is SET to 2 and the turtle changes to colour 2, blue.

Now try the rest of the ‘setpc’ numbers.

setpc 0	Dark blue
setpc 1	Yellow
setpc 2	Light blue
setpc 3	Red

Sound Control

Now for something really different. Not visual this time, but audible. Do you remember at the end of Chapter 3 when programs were enlivened using the ‘sound’ command? As a reminder, try:

```
sound [1 20 100]
```

As promised in Chapter 3, we can now look into some of the musical capabilities of DR Logo.

Now the sound command is followed by an instruction list which includes three digits:

```
[1 20 50]
```

Now the first number, in the above example the number 1, defines the channel status. The channel status tells the machine how to play the note. The Amstrad has three sound channels, A, B and C. A channel value of 1 tells the Amstrad to play the note through its first sound channel, A. The values required to use the sound channels are:

- A 1
- B 2
- C 4

In order to achieve harmonised notes by playing sounds simultaneously through more than one channel, the reader should consult the user manual: Logo's sound command is very similar to the Amstrad's BASIC sound command. Throughout the rest of this book, channel A (1) will be used.

As mentioned in chapter 3, the second number refers to the tone of the note in a range of 0 to 4095. The higher the number, the lower the tone. Try these two examples:

```
sound [1 100 100]
sound [1 4000 100]
```

The last number in the instruction list decides the note duration measured in hundredths of a second. Thus the above two notes will last for 1 second.

Now you will see that there is a massive choice within this range and here is a suggestion. Type:

```
to play
sound [1 478 50] sound [1 426 50]
sound [1 379 50] sound [1 358 50]
sound [1 319 50] sound [1 284 50]
sound [1 253 50] sound [1 239 50]
end
```

The 'tone numbers' can be found in the appendices of your user manual.

Now if you wanted to you could define these procedures:

to c1	to f
sound [1 478 50]	sound [1 358 50]
end	end
to d	to g
sound [1 426 50]	sound [1 319 50]
end	end
to e	
sound [1 379 50]	
end	

```
to a
sound [1 284 50]
end
```

```
to b
sound [1 253 50]
end
```

```
to c2
sound [1 239 50]
end
```

Now to be adventurous! Combine sound and graphics:

```
to noise
ht fd 40 d rt 51.6 fd 40 e
rt 51.6 fd 40 d rt 51.6 fd 40
cl rt 51.6 fd 40 sound [1 506 50]
rt 51.6 fd 40 cl rt 51.6 fd 40
d
end
```

Musical mathematics; a polyphonous polygon! Try:

```
to bad
repeat 9 [noise rt 90]
end
```

Now back to visual matters. Imagine that you wanted the computer to draw anything it felt like drawing. To let the computer's imagination, of which it has none, and creative ability, of which it has even less, run wild, we have an operation that allows this.

RANDOM (**random**)

This is a Logo operation which will output a whole number.

When we use this operation we tell the computer to think of a whole number, an integer, and to do something with it.

Let's ask the computer to think of a number and print it onto the screen. How about any number from 0 to 9 inclusive?

```
cs  
pr random 10
```

And there is a 'random' number chosen by the computer, printed just where you have have come to expect it. Now try

```
pr random
```

Note what happens with this next line.

```
ts  
repeat 20 [pr random 20]
```

You will end up with a screen which looks something like the one in Figure 6.9 below.

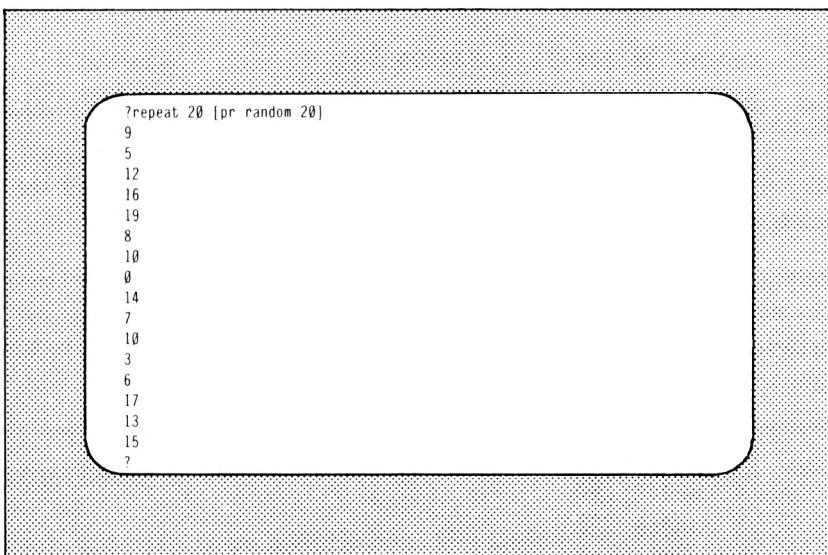


FIGURE 6.9

Now, let's enlarge the computer's possible choice of numbers; clear the screen and try:

```
repeat 20 [pr random 10000]
```

And there we have 20 random numbers between 0 and 9999.

This all is very good, but suppose you wanted a number within a range other than from 0 up to the number required.

For example, imagine you wanted 20 numbers between 15 and 40. Now there are methods for doing this which will be covered in later chapters, but for the time being we can use the following method:

We will ask the computer to think of any number up to 25 and then add 15. This will give us a number between 15 and 40.

Random number chosen by the computer	Constant to be added	Resultant random numbers
1	15	16
2	15	17
3	15	18
24	15	39
25	15	40

So clear the screen and try:

```
repeat 20 [pr 15 + random 25]
```

And there are 20 numbers between 15 and 40.

This ability to choose random numbers will be used later in the book in the games programming chapters.

Now, as promised, we will get the computer to draw a random picture.

What you can do is to move the turtle a random number of screen units, then turn it to the left or right a random number of degrees. Consider the following:

```
to draw
fd random 100
rt random 360
bk random 100
lt random 360
end
```

Now type ‘draw’ and a small, randomly produced picture will be created. All that is left to do now is to say:

```
to scribble
repeat 100 [draw]
end
```

If the turtle moves off screen, don't worry, it will probably come back. If it does not come back just hit escape or wait until the procedure is completed off screen. If you wanted all the turtle movements on screen you could make use of the DR Logo command

wrap

WRAP (wrap)

This makes the turtle reappear on the opposite side of the screen when it moves past the normal screen boundary.

Try typing

```
cs wrap scribble
```

and you will see how this type of screen operates. The type of screen that you are used to is called

window

WINDOW (window)

This type of screen allows the turtle to operate outside the visible graphics screen.

Now if you wanted to stop the turtle moving off the screen you could make use of

fence

FENCE (fence)

Now this will establish an invisible boundary around the graphics screen over which the turtle is not allowed to pass.

Try typing:

```
cs fence scribble
```

and you will notice that the procedure stops with the message:

```
Turtle out of bounds in draw: fd random 100
```

Now to get back to the ‘old’ screen condition, just type

```
window
```

Random commands, of course, can be used to move shapes around the screen.

Let’s define a rotated procedure and draw it in random positions on the screen.

```
to box
repeat 4 [fd 20 rt 90]
end

to rotate
repeat 12 [box rt 30]
end

to pattern
ht
repeat 6 [rotate setpos [0 0] seth ran
dom 360 fd random 200]
end
```

Now clear the screen and run ‘pattern’

The turtle will be hidden, the ‘rotate’ procedure carried out wherever the turtle happens to be (‘random’ positions picked out within the range defined) and ‘rotate’ repeated.

If you want to eliminate the lines connecting your randomly positioned patterns, edit your ‘pattern’ procedure to look like this:

```
to pattern
ht
repeat 6 [rotate pu setpos [0 0] seth
random 360 fd random 200 pd]
end
```

As you can see it is possible to set the turtle to a ‘random’ heading. Clear the screen, show the turtle (st) and type:

```
seth random 360
```

The turtle will set itself in any direction it chooses. You can probably see that the turtle has moved, but what is its new heading? Well you can find out by typing:

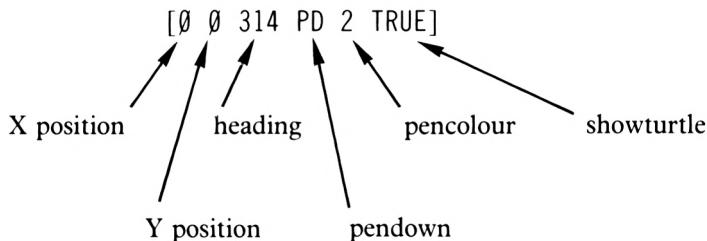
```
setpos [Ø Ø]  
tf
```

TURTLE FACTS (tf)

This command outputs information regarding

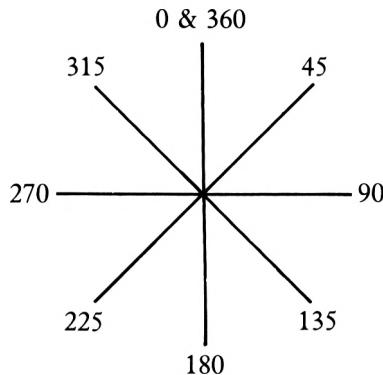
- 1) the turtle's position along the x axis
- 2) the turtle's position along the y axis
- 3) the turtle's current heading
- 4) the turtle's current pen colour
- 5) penup and pendown
- 6) hideturtle or show turtle.

Having entered 'tf' you will now see a set of figures and messages that look similar to this:



Try changing these variables and typing 'tf' again and note the difference.

Remember that in absolute terms your screen headings look like this:



Imagine we wanted to limit its choice range to between 0 and 90 degrees. In this case we would say,

```
seth random 90
```

Try this procedure:

```
to leg
repeat 2 [fd 50 rt 90]
end
```

Type 'leg' and it will draw the simple shape shown below in Figure 6.10.

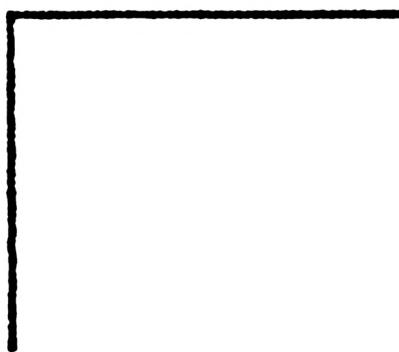


FIGURE 6.10

Then

```
to object
repeat 100 [leg rt random 360]
end
```

Hit 'object' and help! The turtle has wandered off the screen and it is still going. Let it finish, and if you are lucky the turtle may make a momentary guest appearance before going off again.

Then edit the procedure to read

```
repeat 100 [leg setpos [0 0] pd rt rand
om 360]
end
```

Now enter 'object' again and we have constructed a circle, a wheel, a firework or whatever you see it as.

Now enter these procedures:

```
to pat
repeat 15 [fd 10.473 lt 12]
end

to squirt
repeat 50 [pat pu setpos [0 0] pd rt random 360]
end
```

Then hide the turtle, clear the screen and type ‘squirt’ and the resulting drawing looks like a spiral galaxy.

Using the ‘heading’ operation, try the following procedures.

```
to blob
ht
repeat 10 [fd 9.426 rt 36]
end

to patt
repeat 10 [fd random 60 blob seth random 360]
end
```

And try ‘cs patt’.

Now let’s control the pattern by editing ‘patt’ to read:

```
repeat 10 [fd random 60 blob pu setpos [0 0] pd seth random 360]
end
```

Recursion

Recursion is a term which, although it sounds complicated, is very simple.

TO DO AND TO CONTINUE DOING
UNTIL TOLD TO STOP

For example, a procedure to draw a circle could be viewed as a recursive activity in which the turtle moves forward and turns through an angle repeatedly until a given number of ‘forward/turn’ cycles have been completed.

Another recursive ‘talent’ available to you in Logo is the ability to ask any procedure to recall itself simply by putting the procedure’s name within the procedure itself. Have a look at the following example:

```
to box
repeat 4 [fd 30 rt 90]
pu setpos [0 0] rt random 360 fd random
m 200
pd
box
end
```

This procedure will draw a box in the middle of the screen, lift the pen, relocate randomly, put the pen down and start the procedure all over again. Type in the above procedure and watch it work. You will have to hit <ESC> to stop it because there is no counting mechanism in the procedure to stop it; this will be covered later.

If you wanted precise control over a procedure, especially a recursive one like the above ‘box’ procedure, a new Logo command will be of use.

MAKE (make)

This command is very important because it ‘makes’ the computer remember things by assigning a value to a variable.

As an example, enter this line:

```
make "xx 1
↑
Quote marks, these are very
important
```

When you hit <ENTER> nothing appears to happen, but something has. The computer has given the name ‘xx’ to an area of its memory and put a value of 1 in it. To prove this you just have to type:

```
pr :xx
↑
remember the colon
```

and the value of the ‘variable’ as it is known, is displayed on screen.

Having given the variable ‘xx’ a value does not mean that it has to keep that value. It can always be changed by using the ‘make’ command to assign a new value to it, hence the use of the word ‘variable’: you can ‘vary’ the contents of the memory location.

For example type:

```
make "xx 6 pr :xx
```

and there’s the new value. What has happened is that the computer has seen the ‘make’ command, looked at xx to see if the variable (or memory location) already exists and, since it does, simply reassigned a new value to the variable xx.

It is very important to use the quote marks and the colon when ‘making’ or recalling a variable.

Try this command:

```
make "xx :xx + 1
```

When you hit <ENTER> as you might expect not much happens, but what does this command mean? Well, follow it through carefully. It says:

```
make "xx :xx + 1
```

‘Make the variable (memory location) called xx equal its current value, +1’

Now enter ‘pr :xx’ and you will find that its value has increased by one. If a loop is set up which continuously increments a variable, the variable just gets bigger and bigger.

Let’s increase the value of xx using recursion. Remember you must

- 1) set the value of the variable
- 2) print its value
- 3) increase its value
- 4) recall the procedure

```
to grow
pr :xx
make "xx :xx + 1
grow
end
```

If you now type

```
grow
```

the value of xx will be printed, and incremented by one before the procedure is recalled. This process will continue until you hit <ESC>.

Clear the screen then let's make it count backwards in steps of three by typing these direct commands:

```
make "zz 33  
repeat 10 [pr :zz make "zz :zz - 3]
```

Now that you are able to increment and decrement numbers, try to define a procedure to draw a spiral like the one shown below in Figure 6.11.

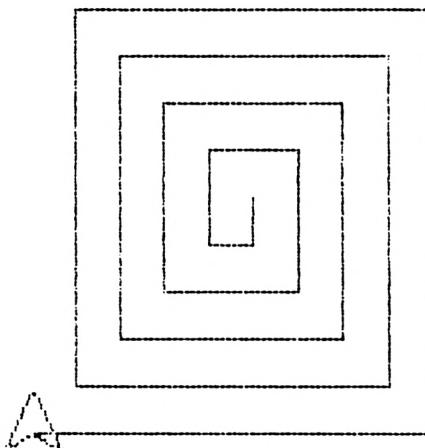


FIGURE 6.11

Without the ability to set a variable and increment it, each line would have to be typed in separately – which would make it a fairly long-winded task. However, using recursion the technique becomes much simpler as we can say:

```
to maze  
make "zz 15 rt 180
```

This first line sets the variable zz equal to 15 screen units and turns the turtle to face the bottom of the screen ready to start drawing.

```

repeat 9 [fd :zz rt 90 fd :zz make "zz
          zz + 15 rt 90]
end

```

This turns the turtle before drawing the next two lines

This command will add 15 units to zz before drawing the next two lines

This will draw 2 lines zz screen units in length at right angles to one another

You will have noticed that when a procedure was made to call itself it carried on and on until it was stopped by hitting <ESC>. We then controlled the recursion by use of the repeat command. However it is quite often required to test a set of conditions and stop a procedure when particular conditions have been met and since this can't be done by use of <ESC> or 'repeat', use of 'if' and 'stop' can be made.

IF (if)

This is an operation that runs an instruction list if the condition set is true.

STOP (stop)

This stops the execution of the current procedure.

Now for something visually exciting. Edit 'maze' to look like this:

```

to maze
  fd :zz rt 90 fd :zz make "zz :zz + 5 r
  t 90
  if :zz=150 [stop]
  maze
end

```

Let's reduce the size of the gap between the lines to make it look as though you are looking down a long corridor or down on a tall pyramid. Try:

```

to try :zz
  fd :zz rt 90 fd :zz rt 90
  make "zz :zz + 2
  if :zz = 99 [stop]
  try :zz
end

```

Now what we have done here is to pass a parameter into a recursive procedure. We know that it will stop when the line length equals 99. So now try:

```
try 2
```

Now the procedure works beautifully except that it won't stop, so you will have to hit <ESC>. The reason this recursive procedure didn't stop was because zz never equalled 99. It started at 2 and was incremented by steps of 2, so it was 98, then 100. The best thing therefore is to edit the third line of 'try' to read

```
if :zz > 99 [stop]
```

If zz is greater than 99 stop the procedure.

What will happen if you rotate this procedure? The following procedure does just that:

```
to look
ht fs
repeat 3 [try 2 pu setpos [0 0] pd rt
30]
end
```

Type

```
look
```

The visual effect is good but could be improved.

The answer might be to increase the space between the lines so edit 'try' to look like this

```
to try :zz
fd :zz rt 90 fd :zz rt 90
make "zz :zz + 10 ← change
if :zz > 99 [stop]
end
```

Why not try defining a procedure to draw the pattern shown below in Figure 6.12?

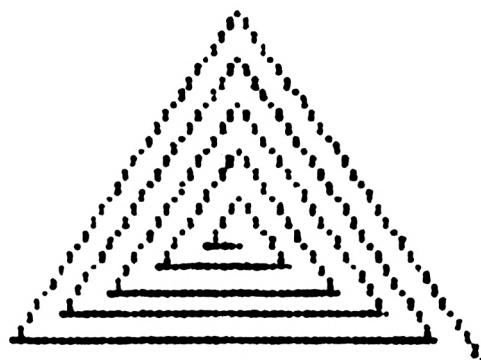


FIGURE 6.12

Using recursive techniques this is now a relatively simple matter. In the pattern shown above each line is longer than the previous one. A procedure to draw this pattern might be:

```
to triangle :aa
  fd :aa rt 120
  make "aa :aa + 5
  if :aa > 60 [stop]
  triangle :aa
end
```

Now type

```
lt 90 triangle 4
```

You might then say, in direct mode

```
lt 90 repeat 3 [triangle 4]
```

Both the square and the triangle are quite interesting but how about drawing a spiral shape with a curved nature about it? Like the one shown below in Figure 6.12(a).

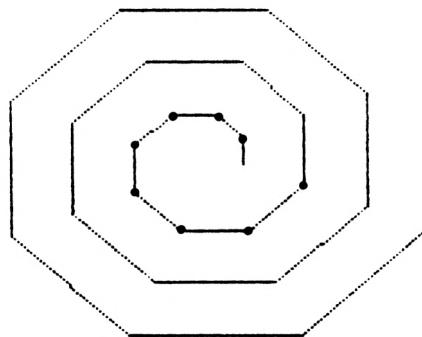


FIGURE 6.12(a)

If you consider the number of turns for the total trip of the turtle by counting from the diagram, you will find that the turtle must turn left 8 times. This is $360/8=45$ degrees each turn.

Make the first movement of the turtle 10 screen units in length by making $xx=10$, and add 2 units to each new line:

```
to web :aa
  fd :aa lt 45
  make "aa :aa + 2
  if :aa > 100 [stop]
  web :aa
end
```

Type

```
web 10
```

What will happen if you reduce the forward movements in the ‘web’ procedure? Edit the procedure to look like this:

```
to web :aa
  fd :aa lt 45
  make "aa :aa + 1
  if :aa > 100 [stop]
  web :aa
end
```

Now enter ‘cs web 1’ and your screen will fill up with a cobweb shape like Figure 6.12(b) shown below.

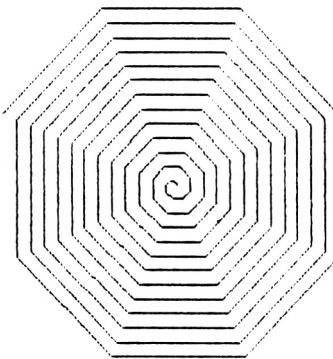


FIGURE 6.12(b)

What will happen if you make the left turn smaller?

Well, try it and see. Edit ‘web’ in the following way:

```
to web :aa
  fd :aa lt 25
  make "aa :aa + 1
  if :aa > 100 [stop]
  web :aa
end
```

Now try ‘web 1’ again and look at the result: a perfect spiral (well, nearly). Your screen should show the spiral of Figure 6.12(c) below:

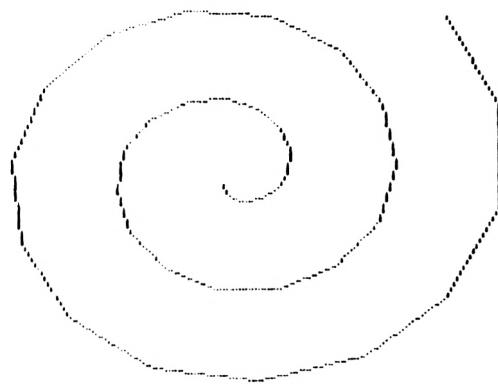


FIGURE 6.12(c)

Let’s tighten the spiral up by reducing the forward movements once again:

```
to web :aa
  fd :aa lt 15
  make "aa :aa + 0.1
  if :aa > 10 [stop]
  web :aa
end
```

Now type ‘cs web 0.1’ and there it is, just like the one shown below in Figure 6.12(d).

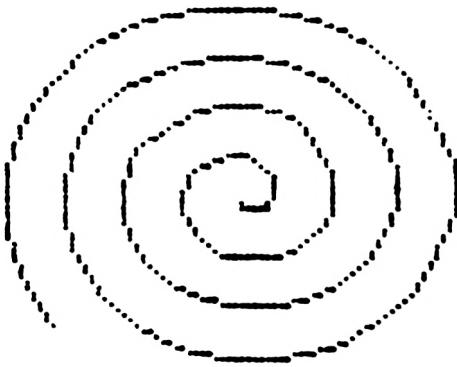


FIGURE 6.12(d)

Clear the screen and say, in direct mode:

```
lt 90 repeat 4 [web 0.1]
```

and wait for the effect!

In the light of the knowledge gained so far it would seem reasonable to limit the subsequent increase in each forward movement of the turtle in order to tighten up the spiral even further. Why not try editing 'web' in the following manner:

```
to web :aa
fd :aa lt 15
make "aa :aa+0.05 ← change
if :aa > 8 [stop] ←
web :aa
end
```

and now, in direct mode, you can try

```
fs repeat 5 [web 0.1 rt random 360]
```

After a little while, you might get a message saying

I'm out of space in...

The '?' prompt will have changed to '!' as well. This means that the computer has run out of memory space. The reason is that whenever a procedure calls another procedure, the computer has to remember which procedure did the calling. Procedure 'web' calls itself so many times that the computer can't cope! To free some of the computer's memory for future use, type in a new command:

```
recycle
```

RECYCLE (recycle)

This command gets the computer to sort out its memory space and make available for further use as much memory as possible without forgetting any procedures or variables.

Then alter the '0.05' in web to '0.07' – the 0.05 will appear as '5.e-02', so just change the 5 to a 7 – this is the computer's own way of representing small numbers, where the 'e-02' means that the number is 5 with the decimal point moved 2 places to the left. A bit confusing perhaps, but you don't have to bother with that anyway as the computer can accept normal numbers if you choose to type them in.

Right! Now let's try again:

```
fs repeat 5 [web 0.1 rt random 360]
```

and then go to lunch while this is running.

Let's try nesting a procedure inside a recursive procedure and see what happens. Start off with a simple box procedure like this:

```
to box
  fd :xx rt 90 make "xx :xx + 10
  fd :xx rt 90 make "xx :xx - 10
  fd :xx rt 90 make "xx :xx + 10
  fd :xx rt 90
end
```

Don't run this procedure right now: xx hasn't been given a value yet.

Now try this:

```
to pin
  box make "xx :xx + 5
  if :xx > 60 [stop]
  pin
end
```

Now type:

```
make "xx 5
cs
fs
repeat 4 [pin rt 90 make "xx 5]
```

And your screen should look like that shown below in Figure 6.13.

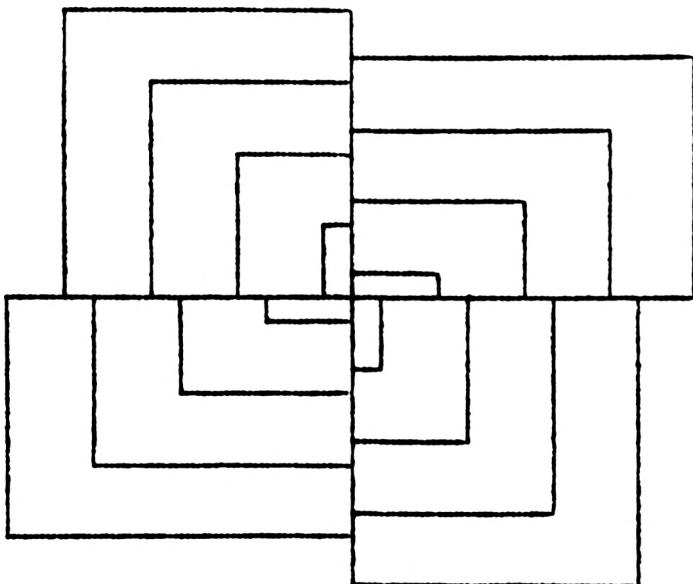
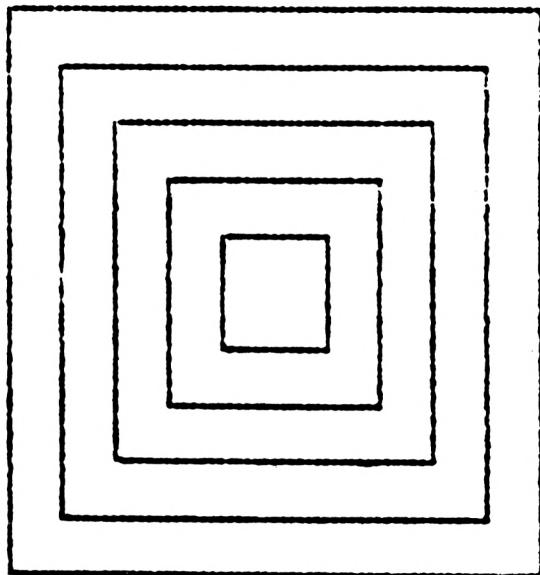


FIGURE 6.13

Why not try some of your own ideas using some recursive rotated procedures, mixing shapes like triangles, squares and spirals? One thing to watch out for, though: no doubt sometimes you will get messages like 'I'm out of space' or 'Out of stack space' or 'Out of stack during garbage collection', or even 'I don't have any nodes left'. These all mean essentially the same thing: your procedures are calling themselves too many times and the poor computer can't keep track of it all! Try altering the procedures a bit so that they can do the job without calling themselves quite so many times. (You'll have to type 'recycle' before doing any editing though.)

Projects

PROJECT 1 Draw some concentric squares like the ones shown below and define a procedure to produce the finished article.



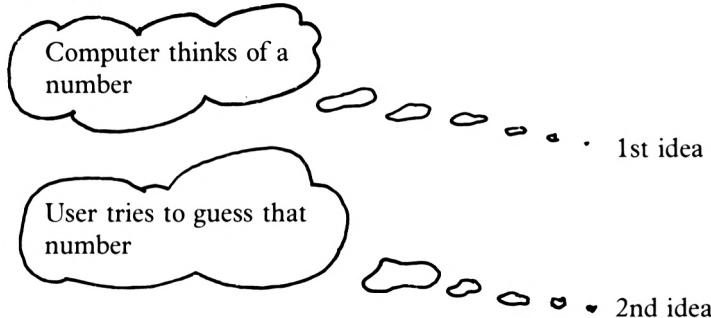
PROJECT 2 Fill the screen with random sized boxes at random screen locations.

Chapter 7

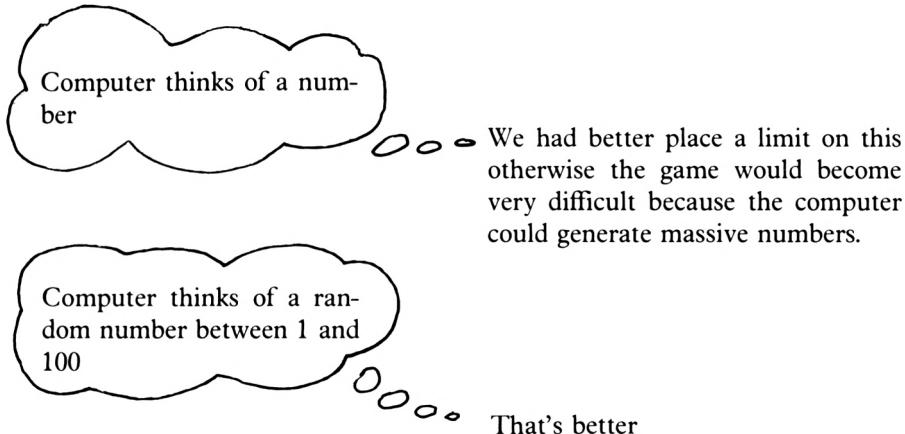
Now to make use of the computer's ability to think of a random number. Up to now we have concentrated on the graphical abilities of Logo. To introduce and demonstrate other abilities, we will make up a number guessing game that anybody can play against the computer.

The approach to the problem will be to plan and design the central core of a procedure that will run the number guessing game. Then this will be embellished with sound and messages, to make it more enjoyable for the user.

First of all, look at a couple of general ideas from which we can begin to create the core procedures:



Now let's look more closely at the 1st idea.



To make the computer generate and remember numbers we can use the command 'make'.

As a reminder, try this quick procedure:

```
to loop
  make "n 2000
  repeat 10 [pr :n]
end
```

Now type 'loop', and 2000 will be printed 10 times. We could have got exactly the same result by saying:

```
to loop2
  repeat 10 [pr [2000]]
end
```

What we did however, was to make 'n' represent 2000, i.e. a fixed number. For the number guessing game 'n', or more clearly 'number', needs to be made equal to a random number:

```
make "number random 100
or
make "gh random 100
```

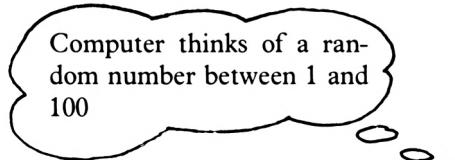
Any string of letters will do for the variable name – the computer will tell you if any name given is unsatisfactory.

Try the first one by putting it into a procedure:

```
to guess
  make "number random 100
  repeat 10 [pr :number]
end
```

Now type 'guess' and the random number chosen between 1 and 100 is printed ten times.

Now to have a look at the first idea translated into Logo:



```
make "number random 100
```

That line is a very useful first step in a number guessing game!

The game so far now looks like this:

to guess

make "number random 100

We try to guess that number

It might be a good idea if 1) the computer tells us if we are right or wrong and 2) whether our guess is too high or too low.

So the procedure will include these ideas.

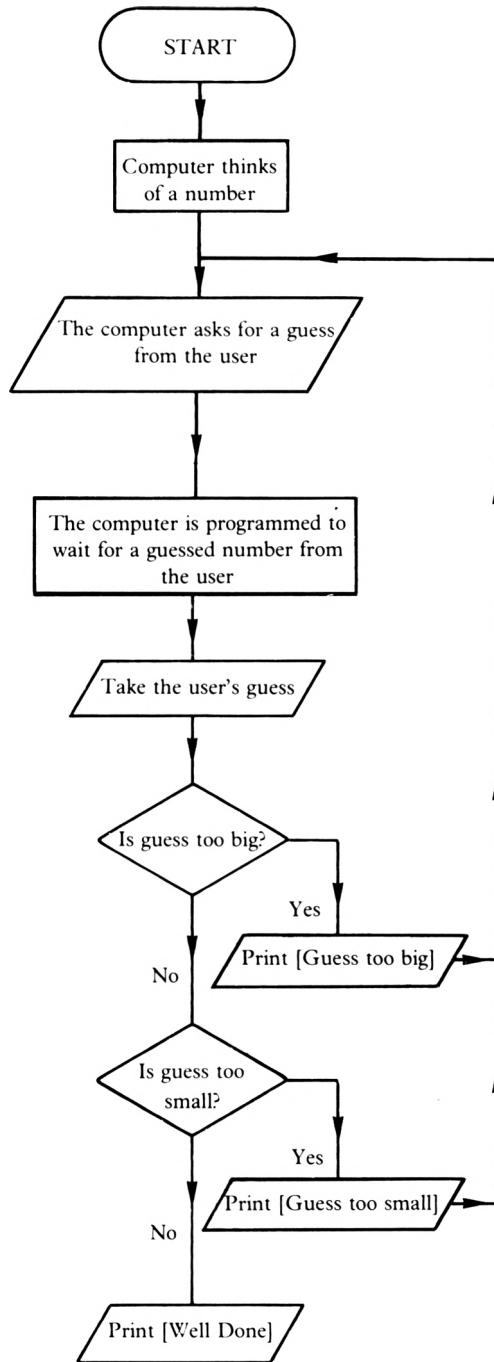
to guess

make "number random 100

A guess from the user. A message from the computer saying “too high” or “too low” or “guess correct”.

It might also be a good idea if we place a limit on the number of guesses allowed. This will make the game more exciting.

Now to formalise these ideas into a plan of action that the computer can follow in a logical manner.



make "number random 100

This can be in the form of a printed message, for example: pr [Put in your guess]

How to do this hasn't been explained yet.

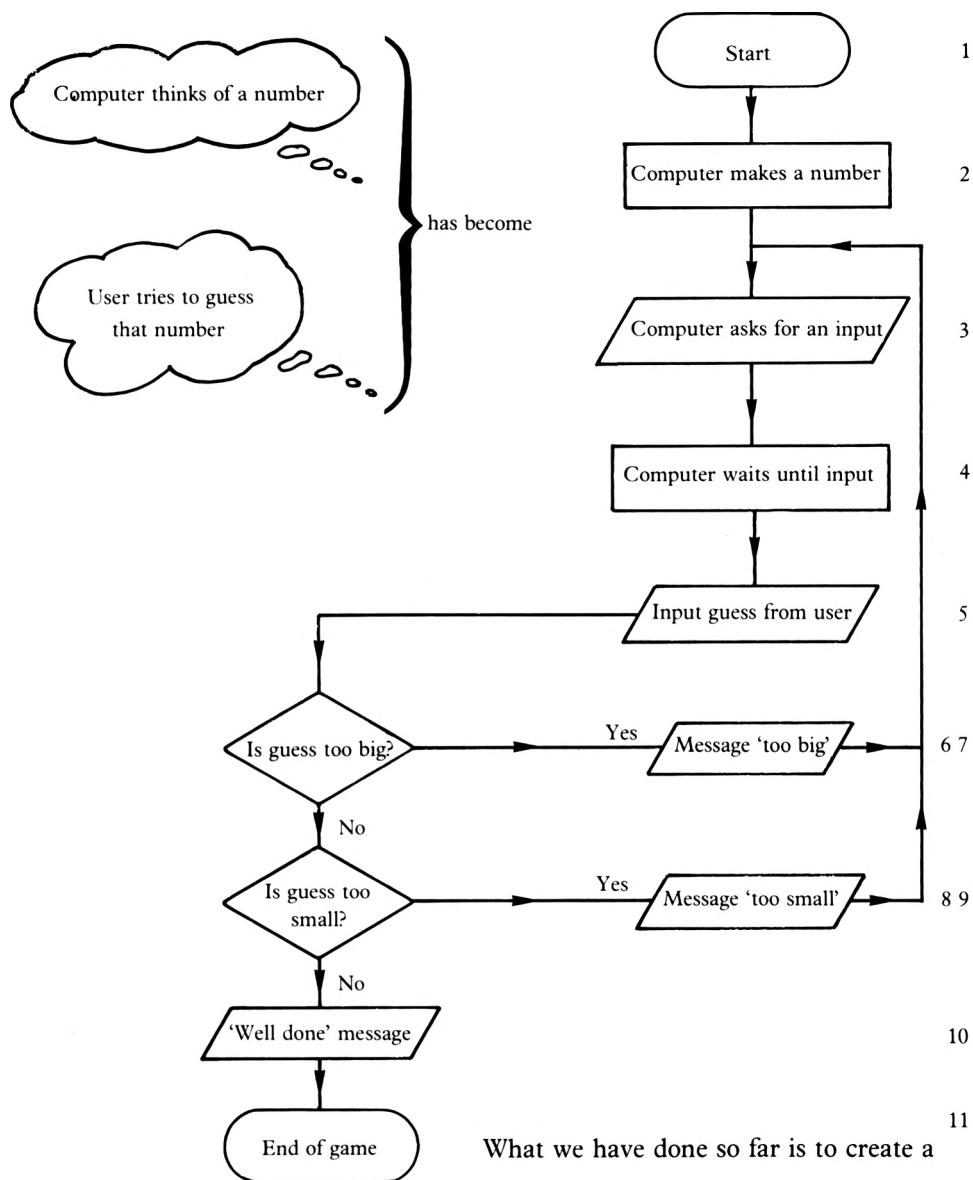
This will be a number typed in by the person using the computer and it will go into a memory location in the computer.

Here, the computer will compare the random number with the input number. If the input number is too big, a message to that effect is printed and the computer asks for another guess from the user.

Having looked to see if the guessed number is too big, if the answer is 'No' then the computer now looks to see if it is too small. If it is, then the user is once again asked to have another guess.

If the guess is neither too big nor too small, it must be just right: that is, equal to the random number. This being the case, the game ends with a 'well done' message.

Now let's compare our starting ideas with our new plan.



What we have done so far is to create a

Flow Diagram

This diagram shows us how the game procedure will work. The procedure will follow the arrows on the diagram until the guessed number is the same as the randomly chosen number. At this point, the game ends. If you like, you can try it with a friend. All you have to do is write a number on a piece of paper and ask a friend to make guesses. All you do is to say "too big" or "too small". Eventually the number will be guessed correctly at which point you can give a "well done" message.

You will notice that the areas on the flow diagram have been numbered. This is just so that they can be referred to easily. Areas 1, 2 and 3 have been dealt with.

1) call up procedure guess and the game starts

2) make "number random 100

3) pr [Please guess a number]

Now we come to area 4.

The computer waits for an input from the user.

This uses a new command:

READ LIST (rl)

'Read list' is a Logo operation which waits for an input from the user. It stores the input in a named memory location (i.e. in a 'variable'). When 'rl' is used in a procedure, a '█' cursor is displayed to show that the computer has stopped and is waiting for an input at the keyboard. In the number-guessing game that input will be a number.

Command number four will therefore be:

```
make "anything rl
```

Let's rephrase that and say

```
make "input rl
```

Both 'anything' and 'input' are examples of names for memory locations. Just about any string of letters will do.

Try this procedure:

```
to ask
  make "input rl
  pr :input
end
```

When you call up ‘ask’ you will notice the ■ cursor waiting for your input. When you have typed in a number and hit <ENTER>, the rest of the procedure is carried out. In this case the number is printed.

Your screen will now look like this

```
?ask ← You called the procedure  
123 ← You answered the request for an input  
123 ← The input was printed  
?
```

We can use

```
make "input rl
```

as the fourth command in the procedure.

The fifth area of the flow diagram is not complicated. The player simply types in a number.

The sixth area looks at the input and compares it with the chosen random number. For this we use the command:

IF (if)

IF is a Logo operation that runs an instruction list when the condition set is true.

In the case of this program, if the random number is smaller than the guess then the program prints a “too big” message.

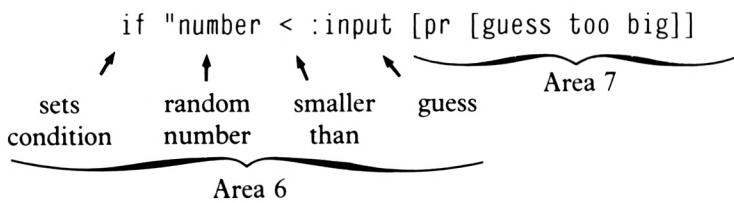
If the random number is larger than the guess, the computer passes this line and goes to the next command down (areas 8 and 9).

In computer languages, ‘>’, SHIFTED full stop, means ‘greater than’ and ‘<’, SHIFTED comma, means ‘less than’.

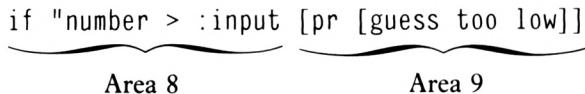
Thus we can replace



in the flow diagram with



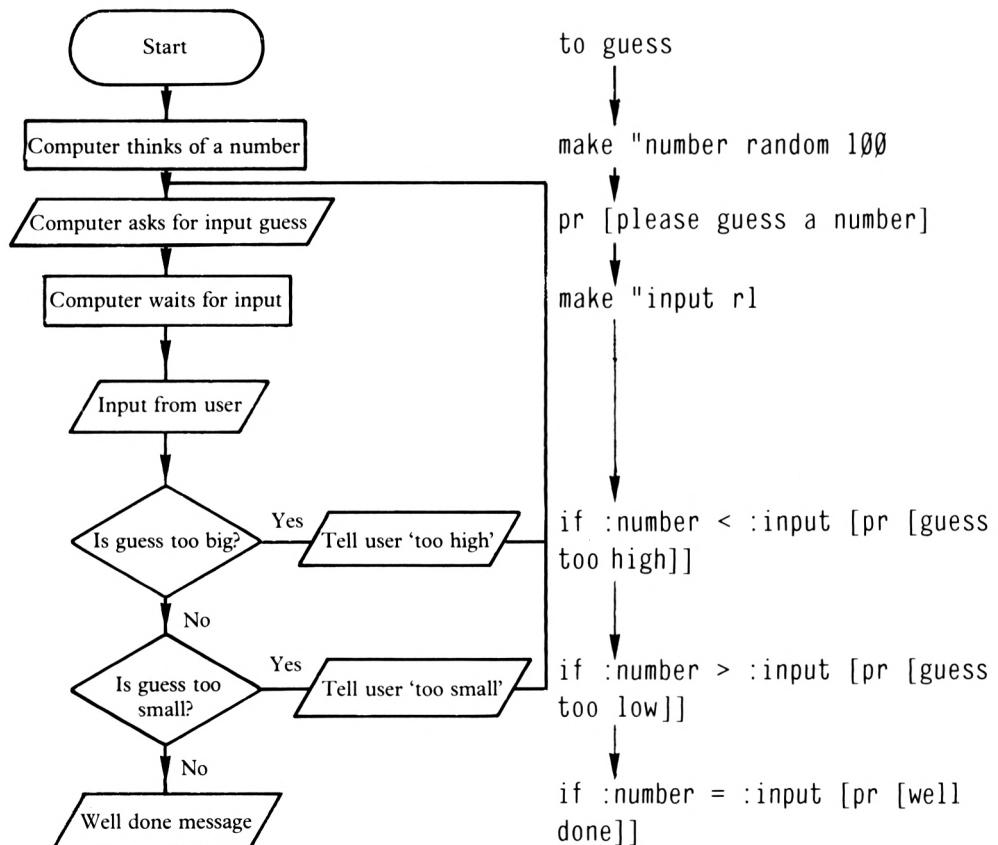
The eighth and ninth steps are now fairly easy. We can simply say:



Similarly for area 10.

```
if "number = :input" [pr [well done]]
```

Now let's have a look at the flow diagram and the corresponding procedure:



If you now enter the procedure as outlined above and run it, you will, after putting in your first guess, be confronted with the message

```
< doesn't like [56] as an input in guess
```

Your first guess will
be printed here.

Now what is happening is that the computer thinks you are trying to compare an ordinary number with a number inside a set of square brackets, so we must try to take the 'rl' number out of the brackets (i.e. out of the memory location called 'input', which the computer thinks contains a list of numbers put in by the 'read list' or 'rl' operation). To do this, use can be made of:

ITEM (item)

This command will output the specified element of a list.

Try this:

```
item 1 [rt 45]
```

What is being said is 'output item one of the following list'.

Now try

```
item 2 [wendy]
```

and you will receive the message

```
Too few items in [wendy]
```

This is because you've only one item in the list.

Now try

```
item 2 [w e n d y]
```

and you will get

```
e
```

Now try

```
item 1 :input
```

and your guess will be printed.

Armed with this knowledge, edit your ‘guess’ procedure to look like this:

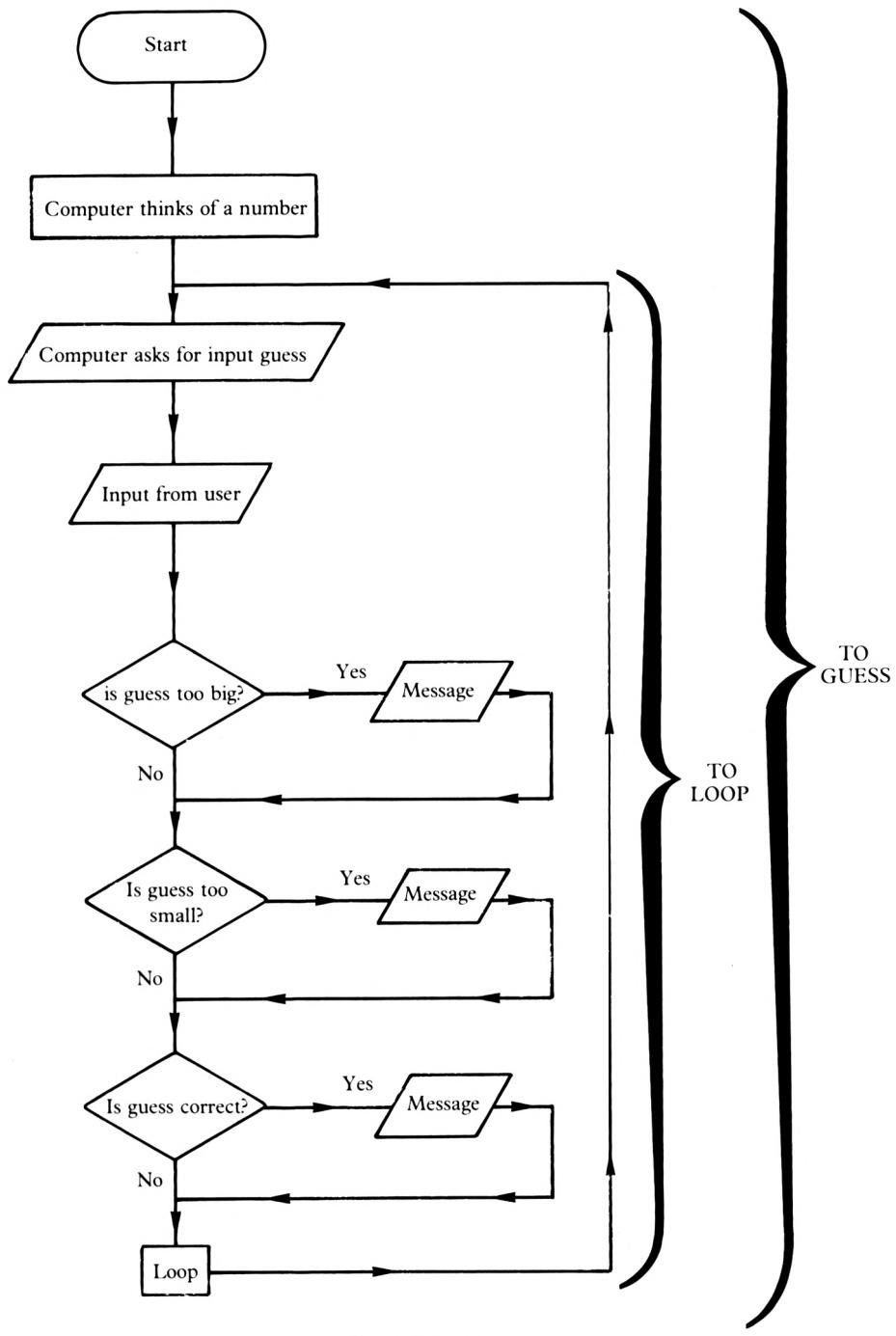
```
to guess
make "number random 100
pr [please guess a number]
make "input rl
if :number < item 1 :input [pr [guess
too big]]
if :number > item 1 :input [pr [guess
too low]]
if :number = item 1 :input [pr [well d
one]]
end
```

At the moment the following will happen when the program is run:

- 1) Computer thinks of a number
- 2) Computer asks user to make a guess
- 3) User inputs guess
- 4) Computer looks at number and says whether or not it is high, low or correct
- 5) End

Well, that’s not much of a game. One guess and it’s all over. What we have got to do is decide how to get back along the loop on the flow diagram below so that guessing can continue until the user guesses correctly.

The best way of doing this is to define the above procedures within the loop as a nested procedure and then call the procedure again. Thus:



Flow Diagram

FIGURE 7.1

We now have the basic structure of a number guessing game.

Let's put the necessary procedure into the computer.

First, edit the nested procedure to look like this:

```
to guess
pr [please guess a number]
make "input rl
if :number < item 1 :input [pr [guess
too big]]
if :number > item 1 :input [pr [guess
too low]]
if :number = item 1 :input [pr [well d
one]]
guess
end
```

Now the game procedure

```
to game
make "number random 100
guess
end
```

The important principle here is that we can send the computer around a loop by making a procedure call itself. (Remember recursion in Chapter six?).

The above 'game' procedure will work, but it will not stop once the right answer has been guessed by the user. (Press <ESC> to stop it.) So to make things more interesting, let's put a limit on the number of guesses the user has; say six.

To do this we must make the computer count the number of times it goes round the loop and stop after the sixth time round. Of course, it also needs to stop once the number has been guessed correctly. To count, set a variable to zero and add one to it every time the 'guess' procedure is called.

To get the program to stop, remember you can include 'stop' in an instruction list:

STOP (stop)

A Logo command that stops a procedure running and returns control to the user.

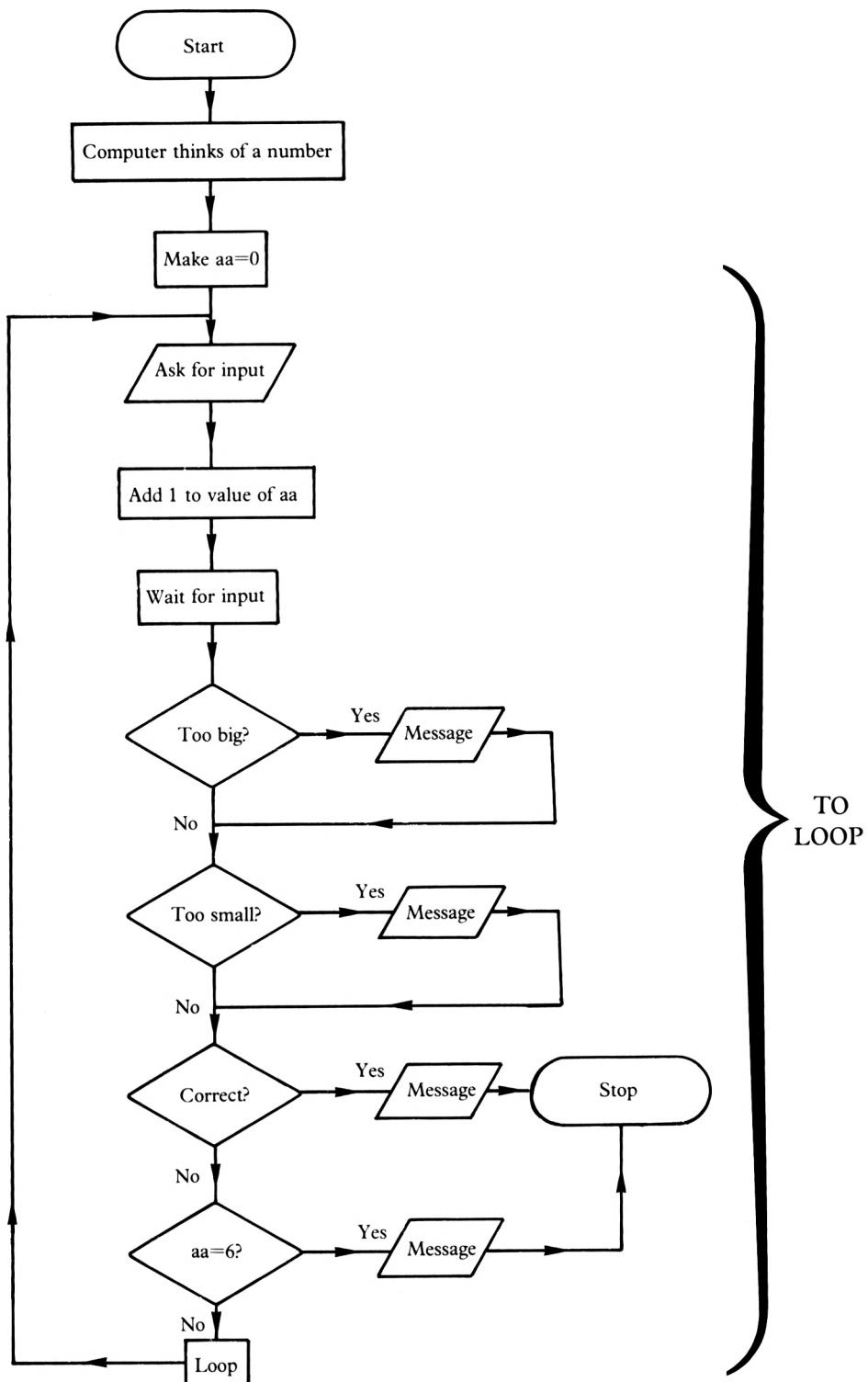


FIGURE 7.2

Edit the procedures to look like this:

```
to game
make "number random 100
make "aa 0 ← Here we make aa=0 thus starting our counting of
guesses
end
```

```
to guess
pr [please guess a number]
make "aa :aa + 1 ← Here we take out aa and add 1 to it. Thus
make "input rl
if :number < item 1 :input [pr [guess
too big]]
if :number > item 1 :input [pr [guess
to low]]
if :number = item 1 :input [pr [well done] stop]
if :aa = 6 [pr [you've run out of guesses] stop]
guess
end
```

The computer reads 'aa' and once its value reaches 6 it gives an appropriate message and stops the game.

Here we take out aa and add 1 to it. Thus every time 'loop' is used, aa increases in value by one.

If you guess correctly within 6 guesses, the game stops.

You will have noticed that as the game progresses, the screen becomes cluttered and after a couple of runs the messages eventually disappear off the top of the screen. This is because the invisible text cursor moves down the screen and so the player no longer has a visible history of the guesses to date. To overcome the first problem, the command 'ts' could be used. This will clear the screen each time. The computer could also print guesses on the screen and then say if that particular guess is too high or too low. Have a look at the following revised procedures incorporating the suggested changes.

```
to game
cs ts ← This has the effect of clearing the screen and
make "number random 100
make "aa 0
guess
end
```

```

to guess
pr [please guess a number]
make "aa :aa + 1
make "input rl
if :number < item 1 :input [pr :input p
r [guess too low]]
if :number > item 1 :input [pr :input p
r [guess to big]]
if :number = item 1 :input [pr :input p
r [well done] stop]
if :aa = 6 [pr [you've run out of guess
es] stop]
guess
end

```

These will print the user's input number each time the 'guess' loop is used.

Edit the old procedures to look like this and run the game. The game runs and works and does all that it should do.

You can also make the computer wait for any given period of time, then carry out the rest of a procedure. Lots of modern clocks and watches nowadays are electronic or 'quartz controlled'. The computer also has an internal electronic clock. This starts counting when the computer is switched on and stops when it is switched off. It counts in quarters of a second and the operation associated with it is:

WAIT (wait)

WAIT is a Logo operation that counts 0.016s of a second and will stop the execution of a procedure for a specified amount of time.

Try this procedure:

```

to time
pr [go]
wait 240
pr [stop]
end

```

When you run this procedure, the word 'go' is printed. Later, 'stop' is printed.

We can now incorporate this feature into the procedure to give:

```
to guess
pr [please guess a number]
wait 250
make "aa :aa + 1
make "input rl
if :number < item 1 :input [pr :input p
r [guess too big]]
if :number > item 1 :input [pr :input p
r [guess too low]]
if :number = item 1 :input [pr :input p
r [well done] stop]
if :aa = 6 [pr [you've run out of guess
es] stop]
guess
end
```

Now how about a welcome message explaining the rules and being generally friendly? Again, let's PLAN the introduction. It can go into the 'game' procedure by nesting another procedure:

Let's say:

```
to message
"HI THERE"
wait
clearscreen
"THIS IS A GUESSING GAME"
"FOR YOU TO PLAY"
wait
"AGAINST ME!"
wait
clearscreen
"I PICK ANY NUMBER
UP TO 100"
wait
"You GET SIX GOES
AT GUESSING IT"
wait
clearscreen
"GOOD LUCK"
wait
clearscreen
```

A procedure to adopt this plan might be:

```
to message
pr [HI THERE]
wait 117
pr [THIS IS A GUESSING GAME]
pr [FOR YOU TO PLAY]
wait 235
pr [AGAINST ME!]
wait 470
pr [I PICK ANY NUMBER]
pr [UP TO 100]
wait 470
pr [YOU GET SIX GOES AT GUESSING IT]
wait 117
pr [GOOD LUCK]
wait 470
end
```

Reading through the ‘message’ procedure, here’s what happens.

- The graphics turtle disappears. The screen is cleared. The message “HI THERE” is printed.
- The computer waits for two seconds.
- A message is printed (the text turtle moves down a line). Another message is printed.
- The computer will now do nothing except wait for 4 seconds from the start of the procedure.
- The message “AGAINST ME!” is printed.
- The computer counts to about 8 seconds.
- The screen is cleared. “I PICK ANY NUMBER” printed, “UP TO 100” printed just below.
- The computer waits 8 seconds.
- Another message is printed.
- Wait 2 seconds.

- The screen is cleared: “GOOD LUCK” appears.
- Wait 8 seconds.
- End

Now let's look at the procedures.

```

to guess
pr [please guess a number]
wait 250
make "aa :aa + 1
make "input rl
if :number < item 1 :input [pr :input p
r [guess too big]]
if :number > item 1 :input [pr :input p
r [guess to low]]
if :number = item 1 :input [pr :input p
r [well done] stop]
if :aa = 6 [pr [you've run out of guess
es] stop]
guess
end

to message
cs ts
pr [HI THERE]
wait 117
pr [THIS IS A GUESSING GAME]
pr [FOR YOU TO PLAY]
wait 235
pr [AGAINST ME!]
wait 470
cs ts
pr [I PICK ANY NUMBER]
pr [UP TO 100]
wait 470
pr [YOU GET SIX GOES AT GUESSING IT]
wait 117
ts
pr [GOOD LUCK]
wait 470
end

```

```
to game
message
make "number random 100
make "aa ""
guess
end
```

The best of luck!

Mind Reading

Having completed a number guessing game procedure, the next task is to consider a way in which the computer can work out the number the user is thinking of. The rules of the game are as follows:

- i) The user thinks of a number between 1 and 60.
- ii) The computer displays six sets of numbers, thirty numbers in each set. The user then answers 'Yes' or 'No' to the question 'Is the number you are thinking of contained here?' as each set is displayed.
- iii) If the user answers truthfully to the six questions the computer will clear the screen and probably display the message:

The number you thought of is X

Sounds almost impossible, but you can be quite sure that you will enable your computer to effectively 'read your mind'.

During the development of the mind reading procedure you must be very careful when typing the sets of numbers in the display procedures.

Let's have a look at the first set of numbers.

The display required is as shown below:

3	5	7	9	11	1
13	15	17	19	21	23
25	27	29	31	33	35
37	39	41	43	45	47
49	51	53	55	57	59

FIGURE 7.3

One way of achieving this is by means of print statements:

```
to aa
cs ts
pr (se " 3 " 5 " 7 " 9 11 " 1)
pr []
pr [13 15 17 19 21 23]
pr []
pr [25 27 29 31 33 35]
pr []
pr [37 39 41 43 45 47]
pr []
pr [49 51 53 55 57 59]
end
```

The empty [] simply moves the text cursor thus leaving an empty line.

All the fuss with 'se' and quotes and spaces in the first print statement is just to make the single-digit numbers line up nicely with the two-digit numbers. A quote mark followed by a space causes the computer, surprisingly, to print a space!

```
to bb
cs ts
pr (se " 3 " 6 " 7 10 11 " 2)
pr []
pr [14 15 18 19 22 23]
pr []
pr [26 27 30 31 34 35]
pr []
pr [38 39 42 43 46 47]
pr []
pr [50 51 54 55 58 59]
end

to cc
cs ts
pr (se " 5 " 6 " 7 13 12 " 4)
pr []
pr [14 15 20 21 22 23]
pr []
pr [28 29 30 31 36 37]
pr []
pr [52 38 39 46 45 44]
pr []
pr [47 54 53 55 60 23]
end
```

```

to dd
cs ts
pr (se " 9 10 11 12 13 " 8)
pr []
pr [15 14 24 25 26 27]
pr []
pr [28 29 30 31 40 41]
pr []
pr [42 43 44 46 45 47]
pr []
pr [56 57 58 59 60 13]
end

to ee
pr [17 18 19 21 20 16]
pr []
pr [22 23 24 25 26 27]
pr []
pr [28 29 30 31 49 48]
pr []
pr [50 51 52 53 54 55]
pr []
pr [24 56 57 58 59 60]
end

to ff
ts
pr [33 35 34 37 36 32]
pr []
pr [39 38 41 40 42 43]
pr []
pr [44 45 46 47 48 49]
pr []
pr [50 51 52 53 54 55]
pr []
pr [56 57 58 59 60 37]
end

```

Once all these are typed in, the first – rather tedious – job is to check them!

Now a procedure to display the necessary message:

```
to message
pr []
pr []
pr [IS THE NUMBER YOU ARE THINKING OF]
pr [CONTAINED HERE? 1 FOR YES, 2 FOR N
0]
end
```

The time has come to explain how the computer's 'mind reading' actually works.

First look at the top right hand number in each of the procedures aa,bb,cc,dd,ee and ff.

```
aa - 1
bb - 2
cc - 4
dd - 8
ee - 16
ff - 32
```

For the whole procedure to work properly all the computer has to do is to add the above numbers to each other whenever the user answers yes (1) to the question 'Is the number you are thinking of contained here?'.

Let's try it! Try 37 and you will get

1 + 4 + 32 = 37 Easy!

```
aa cc ff

to mind
make "tt 0
pr [THINK OF A NUMBER]
wait 32
aa message make "zz rl if 1 = item 1 :
zz [make "tt :tt + 1]
bb message make "xx rl if 1 = item 1 :
xx [make "tt :tt + 2]
cc message make "cc rl if 1 = item 1 :
cc [make "tt :tt + 4]
dd message make "uu rl if 1 = item 1 :
uu [make "tt :tt + 8]
```

```
ee message make "bb rl if 1 = item 1 :  
bb [make "tt :tt + 16]  
ff message make "nn rl if 1 = item 1 :  
nn [make "tt :tt + 32]  
pr (se "YOUR "NUMBER "IS :tt)  
end
```

Now type ‘mind’ and experience the mind–blowing powers of the mind–reading computer.

Projects

- PROJECT 1 Devise a coin–tossing game where 1=heads and 2=tails. The computer chooses 1 or 2, users have to call heads or tails by inputting 1 or 2 and then be told that they have won if they guessed correctly or lost if they don’t. You could also include a scoring procedure.
- PROJECT 2 Devise a procedure for calculating how many carpet tiles measuring 0.5m square are needed to cover any input floor area.

Chapter 8

Arithmetic Operations

It will be no surprise to you by now that you can use Logo to do maths operations. If for example you wanted the answer to the problem $6+7+9+154$ you might care to do the adding up for yourself. However, why not check the answer on your computer? Just type:

```
pr 6+7+9+154
```

The answer is printed on your screen. Why not try something a bit more complicated?

For example $6+3-7+54\times 6$

Now you won't be able to find the multiplication sign (\times) on your keyboard; don't be tempted to use the letter X. In all computer languages the multiplication sign is '*' (shifted ':'), so to work out the above example you should type:

```
pr 6+3-7+54*6
```

Your computer is capable of carrying out very large and complicated calculations, but you must be careful about the way in which you set them out. Take the last one as an example.

$6+3-7+54*6 = 326$

What the computer has done is to

- 1) multiply 54 by 6
- 2) then add 6
- 3) then add 3
- 4) then subtract 7

But suppose we wanted to know the answer to $6+3-7+54$ all multiplied by 6?

Now the answer to this is 336, not 326.

But how do we get the computer to understand this?

The answer is to use brackets; try:

```
pr (6+3-7+54)*6
```

What the computer does is to look inside the brackets first, do that calculation and then do the rest of the calculation.

You must always use brackets in an arithmetic expression if the meaning would otherwise be unclear. The contents are evaluated starting with the innermost pair of brackets, working outwards.

Try typing in direct mode:

```
pr (37*15)+(5/3)+3-7+(6*6)
```

The '/' means 'divided by'. Now try the same expression without the brackets.

```
pr 37*15+5/3+3-7+6*6
```

and the answer is the same. The reason for this is to do with rules of mathematical precedence which will be explained soon.

The arithmetic operators are:

- * multiplication
- / division
- + addition
- subtraction

Suppose you wanted to divide $3+5$ by $6+2$. It would be written $(3+5)/(6+2)$ and if you want it printed on the screen you should type:

```
pr (3+5)/(6+2)
```

If you left out the brackets, the computer would test the expression as

$$\frac{3+5+2}{6}$$

The computer obeys certain rules of precedence when working out expressions. When it sees an expression, it takes the * signs first, then / signs, and last of all + and - working from left to right, working out anything inside brackets first.

Just to make sure you understand these arithmetic rules, work through the following list of expressions and test them to see if you were correct.

It might be a good idea to use a pencil so that your answers can be erased for the next reader of this book.

	Your answer	Actual answer
pr 6 + 3 - 7 + 4		6
pr 7 * 3 + 6		27
pr 4 + 6 / 3 + 8 - 2		12
pr 7 - 8 / 4 - 6		-1
pr 7 - (8/4) - 6		-1
pr 7 - (8/4) - (6/3)		3

Why don't you try typing:

fd 2+3-3*7

or

repeat 40 [fd 3 rt 3*3]

or even

repeat 6*2+4 [fd 37-34 rt 3*3]

These commands are simply indications of the way in which arithmetic expressions can be used in Logo.

Sometimes people have difficulty in remembering their multiplication tables. So we can use Logo to devise a program that you will be able to keep and use whenever you wish to consult it.

First of all the program must be planned. Then we shall use the necessary key words (syntax) to put the program into operation as a procedure.

An introduction might be the best way to start the program. It might read:

Hello, this program helps you with
multiplication tables. Do you wish
to proceed?

Having asked the question, it requires an answer and so the introductory message should continue:

Please answer 1 for Yes or 2 for No.
Hit RETURN after your answer.

The computer will keep this message on the screen until it receives an answer. If it receives the answer '1' the program will continue. If however it receives the answer '2' it will display a message to the effect

OK
GOODBYE
END

Assuming that the answer is '1' the program should perhaps clear the screen and ask another question:

Which multiplication table would
you like to see? Enter a number and
hit return.

The computer will now wait for a number and when it receives it (say 6) it will clear the screen and print:

1 times 6 = 6
2 times 6 = 12
3 times 6 = 18
4 times 6 = 24
5 times 6 = 30
6 times 6 = 36
7 times 6 = 42
8 times 6 = 48
9 times 6 = 54
10 times 6 = 60
11 times 6 = 66
12 times 6 = 72

After a little while it might be useful to ask

Have you seen enough? Answer 1
or 2 and hit ENTER.

If the answer is yes (1) the program can ask say ten questions about the particular multiplication chosen and give a score out of ten. If the answer is no the screen is cleared and the table re-printed. When the program has run its course it will ask the question

Do you want more?
1 or 2

If the answer is '1' the program is re-run, if it is '2' the goodbye message is printed.

Now to start writing the program.

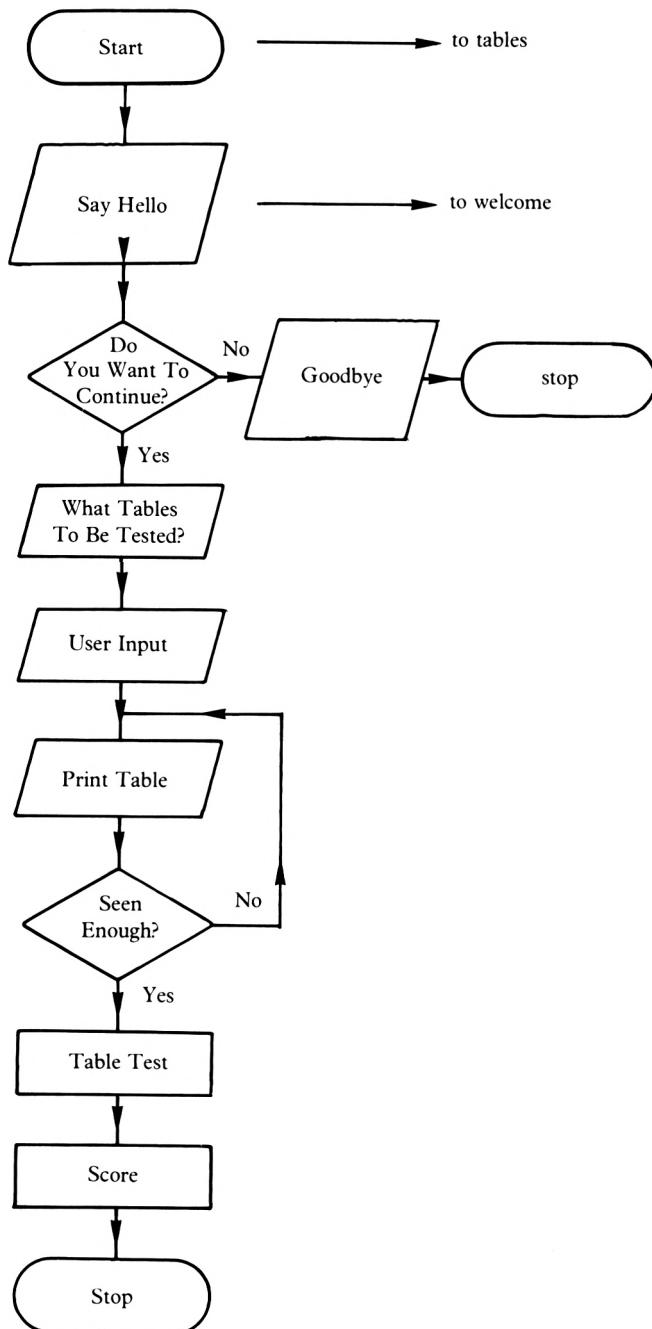
It will start off with a welcome message.

```
to welcome
cs ts
pr [Hello, this program]
pr [helps you with your]
pr [MULTIPLICATION TABLES.]
wait 100
pr []
pr [DO YOU WISH TO GO ON?]
pr []
pr[Pleas answer 1 for yes]
wait 60
pr [or 2 for no.]
wait 100
end
```

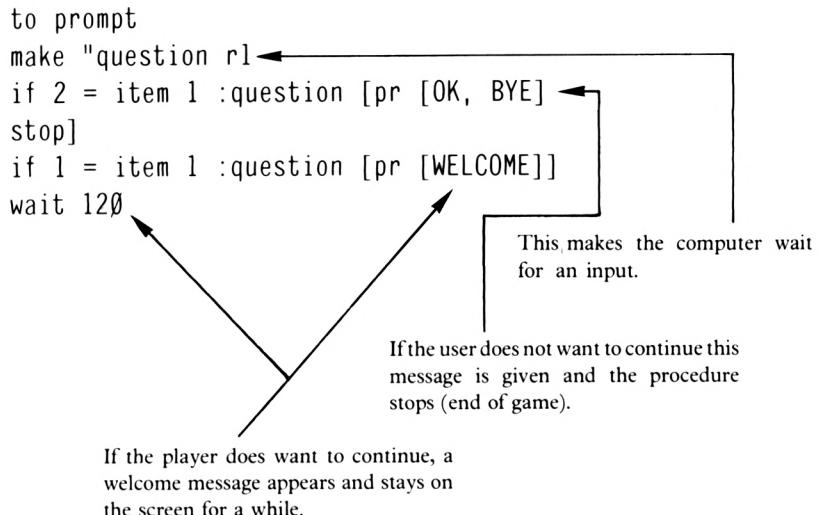
The 'tables' procedure so far now looks like this:

```
to tables
welcome
end
```

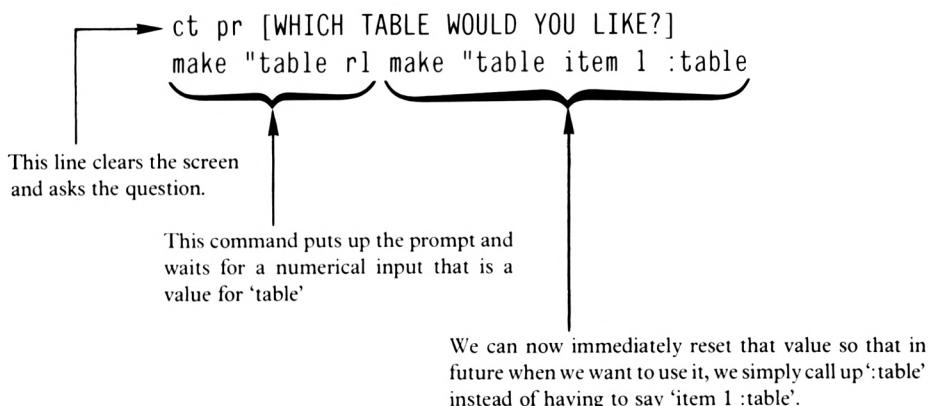
Let's have a look at a flow diagram which will help in the planning of the procedure



After running the 'welcome' procedure, the computer will wait before continuing with the next part of the 'tables' procedure. If you examine the flow chart you will see that what is needed is an input from the user in the form of '1' for 'yes' or a '2' for 'no'. The 'make readlist' command can be used to do this as shown below. Let's call the next procedure 'prompt'. It will start like this:



Looking at the flow diagram you can see that another question which requires an answer from the user is asked. So we can say:



You will notice a strange command in the fifth line of 'prompt'. That command is:

ct

CLEARTEXT (ct)

This erases all text currently on the screen and places the text cursor in the upper left hand corner of the text screen.

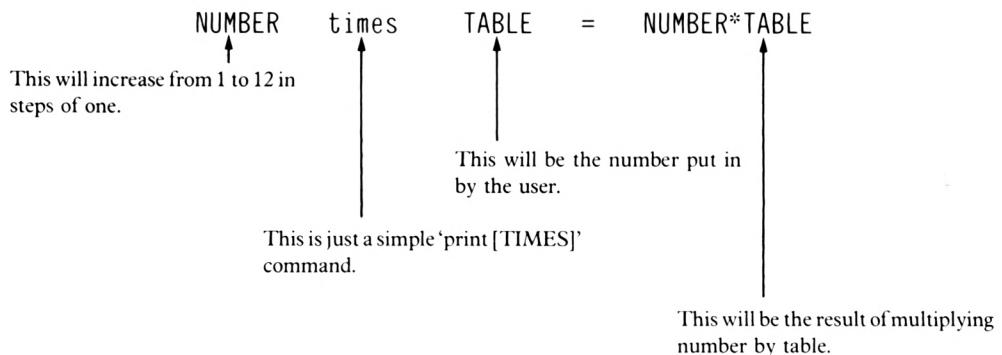
Having received an input, the computer should then print out the table requested. Let's say that we want the print out in this form:

1 times 6 = 6
2 times 6 = 12
3 times 6 = 18
4 times 6 = 24
5 times 6 = 30
6 times 6 = 36
7 times 6 = 42
8 times 6 = 48
9 times 6 = 54
10 times 6 = 60
11 times 6 = 66
12 times 6 = 72

'6' has been used here as a possible value for 'table' which would have been previously put into the computer following the question 'which table would you like?'.

As something is to be repeated twelve times it is as well to create a separate procedure to print a line and then repeat this procedure twelve times.

Let's look at what we need:



Another thing that needs to be done is to set a variable to 1. It can then be increased by 1 every time a new line is printed. If you remember, something similar was done in Chapter 7 when a count was needed of the number of guesses made. The first thing to be done is to set the variable to '1' or 'initialise' it; for example:

```
make "nn 1
```

which will be the next line of 'prompt'. The procedure 'show1' will use this variable when it is repeated twelve times to display the required table.

Try this in direct mode:

make "pp 45	This sets the variable 'pp' to 45.
pr :pp	This will print 45
pr :pp + 1	This will take the value for 'pp', add one to it and print the result.

Having set the variable 'nn', the next lines of 'prompt' should get a number from the user with which it sets the variable 'table' and then asks for the 'show1' procedure to be run twelve times. Thus the 'prompt' procedure will look like this:

```
to prompt
  make "question rl
  if 2 = item 1 :question [pr [OK, BYE]
    stop]
  if 1 = item 1 :question [pr [WELCOME]]
    wait 120
    ct pr [WHICH TABLE WOULD YOU LIKE?]
    make "table rl make "table item 1 :tab
    le
    → make "nn 1
    ct repeat 12 [show1]
    end
```

This will set the counting variable.

This will repeat the procedure 'show1' twelve times and display the required multiplication table.

Now the first element of the first line of ‘show1’ will print the ‘1’ of ‘1 times 6 = 6’ for example:

```
to show1
pr :nn
    NUMBER times TABLE = NUMBER*TABLE
```

The next two pieces of information needed to continue the procedure have already been introduced.

```
to show1
pr :nn pr [times] pr :table
    NUMBER times TABLE
```

Finally the last command can be added to finish the multistatement line:

```
pr :nn pr [times] pr :table pr [=]
pr :nn * :table
```

Having printed the first line, 1 must be added to the value of ‘nn’ before the procedure is repeated, this being achieved quite simply by:

```
make "nn :nn + 1
end
```

All this does is to take the last value of ‘nn’ and add 1 (or ‘increment’ it), so 1 is incremented to 2 which increments to 3 and then 4 and so on as the procedure ‘show1’ is repeated.

Having defined ‘prompt’ and ‘show1’, in direct mode type

```
prompt show1
```

and on screen you will see the tables scrolling past, ending with:

```
13
times
4
=
52
```

To get this you will have to hit 1 and 4 when ‘prompt’ stops for an input.

Now this is not what we expected, but you must remember that the text cursor will move down a line every time it recognises a ‘print’ command. To overcome this problem we can make use of

```
sentence
```

(mentioned in Chapter 4.)

So edit your ‘show1’ procedure to look like this:

```
to show1
pr (se :nn "times :table "= :nn * :tab
le)
make "nn :nn + 1
end
```

and remember that your ‘prompt’ procedure looks like this:

```
to prompt
make "question r1
if 2 = item 1 :question [pr [OK, BYE]
stop]
if 1 = item 1 :question [pr [WELCOME]]
wait 120
ct pr [WHICH TABLE WOULD YOU LIKE?]
make "table r1 make "table item 1 :tab
le
make "nn 1
ct repeat 12 [show1]
end
```

We can now allow the final procedure to include

```
to tables
ts ct
welcome
prompt
stop
end
```

Going back to the flow diagram you will see that once ‘tables’ has been run the procedure should ask the question ‘Have you seen enough?’. To achieve this, we can print the question as shown below when a period of time, say 30 seconds, has elapsed after printing the requested multiplication table. Thus let’s write down a sequence for the testing part of the final procedure which will be something along the following lines:

After 20 seconds ask ‘seen enough?’ Ask the question a second time, or if the answer is yes, carry on with the test procedure and scoring.

seen enough?
Answer 1 or 2

1 x 2 = 2
2 x 2 = 4
3 x 2 = 6
4 x 2 = 8
5 x 2 = 10
6 x 2 = 12

Having printed the required multiplication table on the screen, a new procedure is required to find out if the user has studied it for long enough and wishes to carry on with the next part of the procedure in which they will be tested on their chosen table. In the procedure outlined below, ‘ask’, the user has two thirty second periods in which to study the table, after which time the program carries on with the testing element. Under a new procedure, say, ‘ask’, the computer can start to count. Meanwhile, edit the ‘tables’ procedure to look like this:

to tables	
ts ct	Procedures so far defined
welcome	
make "ss Ø	
make "zz Ø	Commands to be explained later on
make "qq Ø	
prompt	Calls ‘show’
ask	This procedure to be defined next
stop	
end	

Now consider the ‘ask’ procedure:

```
to ask
wait 1764
```

Under the procedure ‘show’ the table will have been printed on the screen. Then ‘ss’, ‘zz’ and ‘qq’ will have been set to zero for use later on and with the first command in the ‘ask’ procedure you are counting 30 seconds while the user studies the multiplication table on the screen.

```
make "zz :zz + 1
```

After a while, ‘zz’, which has previously been set to zero, is taken out of memory and has 1 added to it before being put back into memory. Thus, the variable ‘zz’ is being used to count the number of study periods allowed to the user for studying the table.

```
pr [SEEN ENOUGH?] pr [ANSWER 1 OR 2]
```

These lines set the text cursor and ask ‘Seen enough?’.

```
if :zz = 2 [gol]
```

Calls ‘gol’, which is yet to be written.

You can come back to the above two commands a little later on. The latter is only obeyed if the user has had the allotted two looks at the table to be tested.

```
make "input rl make "input item 1 :inp  
ut
```

The computer will wait for an input of 1 or 2 from the user. If it is not forthcoming then the user can sit and look at the table for as long as desired. This is something you could change as an exercise at the end of this chapter, if you like.

```
if :input = 2 [ask]
```

This line of the ‘ask’ procedure calls itself. Remember that at this point the user has had one look at the multiplication table to be tested. If the user hits 2 in response to the question ‘Seen enough?’, the ‘ask’ procedure is run again. But this time through zz=2 so the user does not get another chance to study the table but is instead transferred to procedure ‘gol’ if a third look is requested.

```
if :input = 1 [test]  
end
```

This says that if the input is ‘1’ (for Yes) then move to a procedure called ‘test’ which says repeat ten times a procedure called ‘try’, followed by ‘score’, after which the program stops.

When we have finished describing the present ‘ask’ procedure we will have a look at ‘try’ and ‘score’.

Line five of the procedure is:

```
if :zz = 2 [gol]
```

Once the computer has had a look at ‘zz’, seen that zz=2, it dutifully moves to ‘gol’.

```
to gol  
ct ts pr [IT'S TEST TIME!]
```

This command moves the text turtle and prints the message “IT'S TEST TIME!”.

```
wait 140
```

The message remains on the screen for a while.

```
test  
end
```

Now this line is very similar to the last line of ‘ask’.

Both the ‘ask’ and ‘gol’ procedures call up ‘test’ which repeats the ‘try’ and ‘score’ procedures ten times as shown below:

```
to test  
repeat 10 [try score]  
end
```

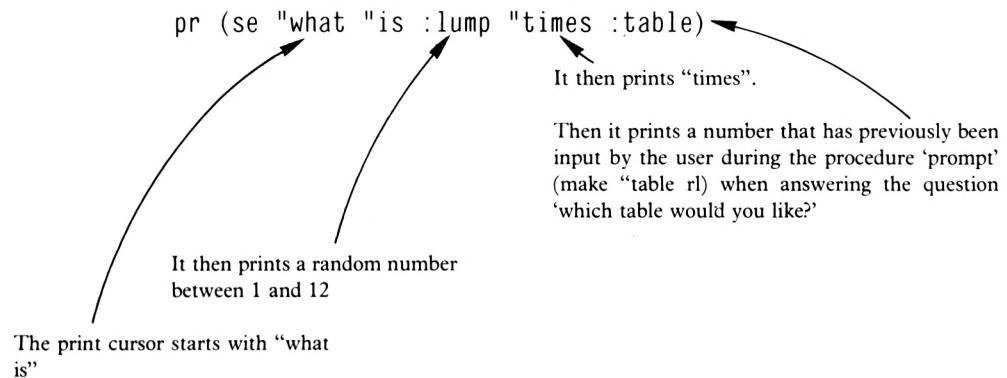
Thus both routes have brought the user to the same testing procedure. Let’s have a look at it:

```
to try  
ct  
setcursor [10 10]
```

The first command clears the screen and calls up the textscreen.

```
make "lump random 13
```

Now to ask ten questions by running the procedure ‘try’ ten times by means of a REPEAT. However, each pattern of questions needs to be different so Logo gets the computer to pick any number between 0 and 12 inclusive as the first digit in each of our questions. Each time the procedure is run, it picks a number.



This results in a question posed in the middle of the screen such as:

what is 8 times 7?

The next task is to input the user's answer and store it in the variable 'answer'.

```
make "answer rl make "answer item 1 :a  
nswer
```

When the user has responded with a numeric answer Logo will add one to the value of 'ss' (remember this variable set to zero in 'tables'?), which is used to count the number of questions that have been answered by the user.

```
make "ss :ss + 1
```

If the user's input (answer) equals the computer's calculated answer to the test question, the message "correct" will printed and 1 will be added to the value of 'qq'. This variable stores the count of the number of correct responses to the ten test questions.

```
if :answer = :lump * :table [pr [correct] make "qq :qq + 1 pr []]
```

If the user's input (answer) is higher or lower than the computer's calculated answer to the previous test questions, the message "incorrect" will be printed.

```
if :answer < :lump * :table [pr [INCORRECT] pr []]
if :answer > :lump * :table [pr [INCORRECT] pr []]
```

The correct answer is then printed by means of:

```
pr (se :lump "times :table "= :lump *)
  :table)
    Random no chosen in question
      times
        table chosen to be printed and tested by user
          = answer
```

For example:

$$6 \text{ times } 7 = 42$$

The last line of the procedure 'try' counts 3 seconds which allows the user time to read the answer. Even if the user is correct in response to the test question, the answer is displayed:

```
wait 176
end
```

The procedure 'try' is repeated 10 times with a procedure provided to display the user's score:

```
to score
```

The last correct answer from 'try' is cleared.

```
ct
pr (se "YOU "GOT :qq "OUT "OF :ss)
wait 100
end
```

End of game

The game is now complete: just to remind you, here's how it works:

```
to tables
  ts ct
```

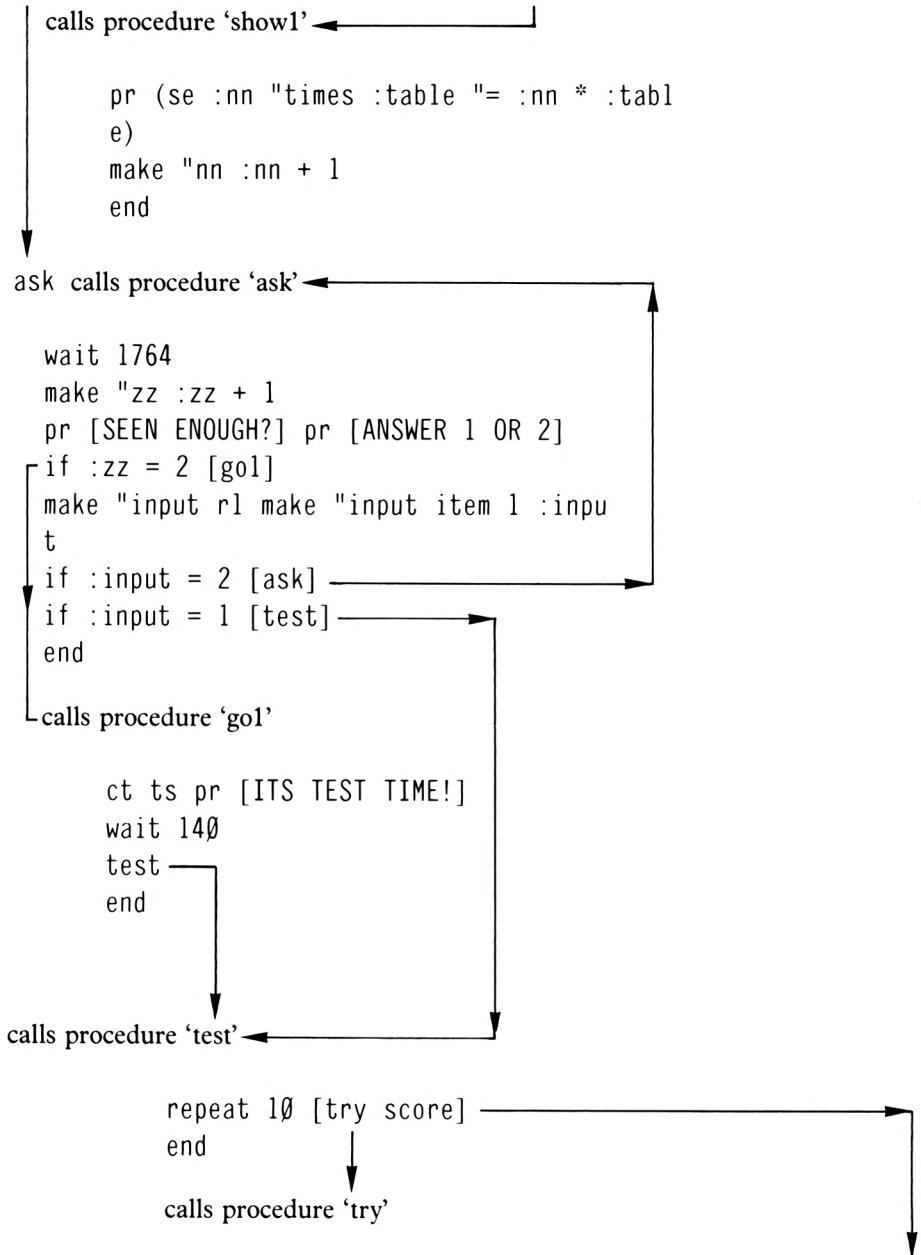
welcome calls procedure 'welcome'

```
to welcome
pr [hello, this program]
pr [helps you with your]
pr [MULTIPLICATION TABLES]
wait 100
pr []
pr [DO YOU WISH TO GO ON?]
pr []
pr [please answer 1 for yes]
wait 60
pr [or 2 for no.]
wait 100
end
```

```
make "ss Ø
make "zz Ø
make "qq Ø
```

prompt calls procedure 'prompt'

```
make "question rl
if 2 = item 1 :question [pr [OK, BYE] stop]
if 1 = item 1 :question [pr [WELCOME]]
wait 120
ct pr [WHICH TABLE WOULD YOU LIKE?]
make "table rl make "table item 1 :table
make "nn 1
ct repeat 12 [show1]
```



```

ct
setcursor [10 10]
make "lump random 13
pr (se "what "is :lump "times :table)
make "answer rl make "answer item 1 :an
swer
make "ss :ss + 1
if :answer = :lump * :table [pr [correc
t] make "qq :qq + 1 pr []]
if :answer < :lump * :table [pr [incorr
ect] pr []]
if :answer > :lump * :table [pr [incorr
ect] pr []]
pr (se :lump "times :table "= :lump * :
table)
wait 176
end

```

calls procedure 'score' ←

```

ct
pr (se "YOU "GOT :qq "OUT "OF :ss)
wait 100
end

```

Projects

- PROJECT 1. Devise a procedure for asking 10 random questions on all 12 multiplication tables with scoring, sound and colour changes.
 (No solution given for this one.)

Chapter 9

In earlier chapters procedures were developed to draw a clock face. Now, several chapters later, enough new commands have been introduced to turn the clock face into a real working clock. For the time being, a simplified version of the face can be used to explain the basic principles of the working clock. Then, finally, the face developed in Chapter five can be incorporated into this final procedure.

Obviously, a very useful feature of any computer is its ability to count. Remember the rate at which it counts? Well it's around 60 times per second, which means that the clock will count $60 \times 60 = 3600$ beats in a full minute.

Try this procedure

```
to blob
  fd 40 bk 40
  wait 3600
  rt 30 fd 40 bk 40
end
```

As expected the procedure will draw a line, wait one minute, turn to the right thirty degrees, and draw another line. It will be quite easy to turn this into a procedure to produce a working clock driven by the computer and displayed on the TV screen. First of all though, it is necessary to plan the essential procedures to do this. So let's start off with a couple of broad ideas within which to develop the overall schema.

Set up clock face → to face
repeat 12 [pu fd 80 pd fd 40
pu bk 120 rt 30]
end

Ask the user to set the clock → Here we could ask the user to set the heading of the hour hand and the minute hand of the clock; once this has been done, the clock will immediately start working.

Allow the clock to work. → Let's have a look in some detail at what we want to happen here!

Print the minute hand in the position specified

Print the hour hand in the position specified

After a minute, erase the minute hand and re-draw it in a new position

Every minute erase the hour hand and re-draw it in a new position.

The first broad idea suggests that a conventional clock face is necessary and so the procedure 'face', shown below, will be suitable until you feel like designing something a little more elaborate with perhaps circles and numbers. The next idea involves the user setting the hands of the clock so that it will tell the correct time.

```

to face
ht
repeat 12 [pu fd 80 pd fd 40 pu bk 120
    rt 30]
end

```

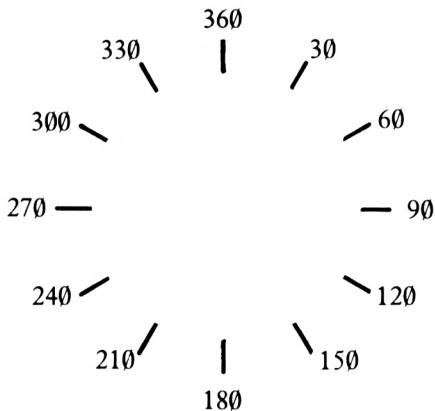
The user's input, in this case the direction in which to point the minute hand, is handled by means of 'make "variable rl'. Before you use this command however, it would be a good idea to print a screen message that tells the user what to do; for example under the procedure 'hands':

```

to hands
pr [ENTER MINUTE HAND HEADING]
pr [THEN HIT ENTER]

```

This will require an input number from the user.



The exact location of the minute and hour hands in terms of their heading will be easy to work out as shown above, as 1 minute on a clock face is equal to 6 degrees of angular rotation.

If the variable 'mh' is used for Minute Hand then the setting can be specified by:

```
make "mh rl make "mh item 1 :mh
```

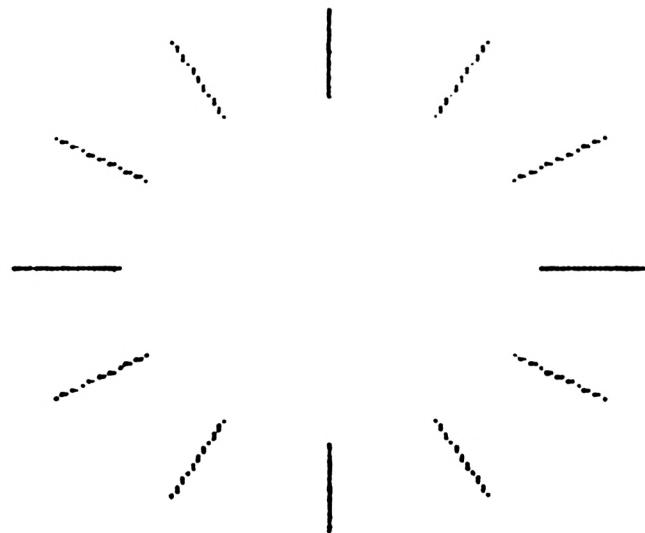
Next to ask for a setting for the hour hand, this time using the variable name 'hh' for Hour Hand:

```
pr [NOW THE HOUR HAND HEADING]  
make "hh rl make "hh item 1 :hh  
end
```

Let's list it so far:

```
to hands  
pr [ENTER MINUTE HAND HEADING]  
pr [THEN HIT RETURN]  
make "mh rl make "mh item 1 :mh  
pr [NOW THE HOUR HAND SETTING]  
make "hh rl make "hh item 1 :hh  
end
```

Run 'face' and 'hands' and make sure that the procedures in fact do as you expect. If you have run the procedures 'face' and 'hands' your screen will look something like Figure 9.1 shown below.



THEN HIT ENTER

30

NOW THE HOUR HAND HEADING

60

?■

FIGURE 9.1

Well, so far we've got a clock face defined as the procedure 'face' and we have a procedure called 'hands' which doesn't do a lot!

Let's add to the 'hands' procedure and make it draw the minute and hour hands in the angular positions specified by the user.

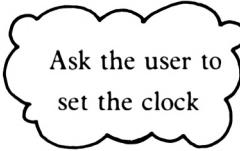
To do this, the turtle is first turned to the correct angle for the minute hand ('rt mh') and then drawn ('fd 35'). Then in preparation for the next operation it is returned home.

```
seth :mh pd fd 70 pu setpos [0 0]
```

Exactly the same procedure is then followed for the hour hand!

```
seth :hh pd fd 40 pu setpos [0 0]
```

The procedure to fulfil the contents of the “broad idea”



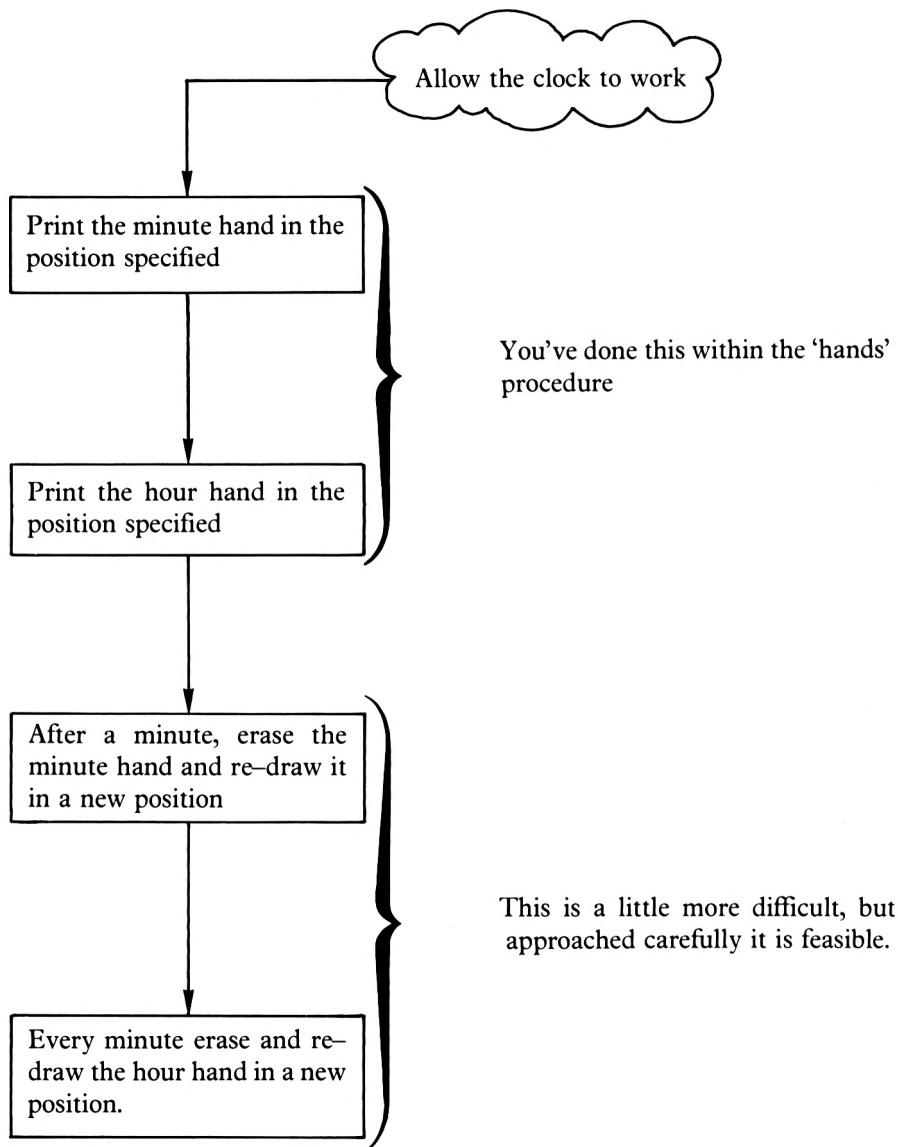
Ask the user to
set the clock

is now complete.

```
to hands
pr [ENTER MINUTE HAND HEADING]
pr [THEN HIT RETURN]
make "mh rl make "mh item 1 :mh
pr [NOW THE HOUR SETTING]
make "hh rl make "hh item 1 :hh
seth :mh pd fd 70 pu setpos [0 0]
seth :hh pd fd 40 pu setpos [0 0]
end
```

Next comes the tricky task of working out a procedure to accurately move the hands of the clock and make them maintain the correct time.

Let's take another look at the expanded version of the idea.



At the beginning of this chapter you were reminded of the way in which the computer keeps time: by counting in 60ths of a second.

The two broad ideas suggest that having set the clock, the computer should wait for 1 minute and then

- 1) rub out the minute hand
 - 2) re-draw it in a new position
 - 3) rub out the hour hand and
 - 4) re-draw it in a new position

Well, the first question to ask is: "How do I get the computer to wait for 1 minute?"

The answer is to use 'wait'. Thus in a 'clock' procedure

```
to clock  
wait 3600
```

The above procedure should make the computer wait while counting for 1 minute, but what next? Well, let's extend the clock procedure to:

- 1) Rub out the minute hand and re-draw it in a new position

and

- 2) Do the same to the hour hand.

This immediately raises the question:

"How can I rub out the minute hand?"

As you know the heading (mh) of the minute hand, the 'rubout' command from chapter 6 can be used in the form:

The diagram shows a sequence of commands: `seth :mh pe fd 70 pu setpos [0 0]`. Three arrows point to specific parts of the command:

- An arrow points to `:mh` with the label "This sets the turtle's heading".
- An arrow points to the number `70` with the label "The input number".
- An arrow points to the closing brackets `[0 0]` with the label "These commands rub out the minute hand and 'home' the turtle".

The next question is:

"How far do I move the heading of the minute hand before it is re-drawn?"

You know that there are 360 degrees in a circle and 60 minute divisions in a circular clock face. Therefore, each one minute step equals 6 degrees. Therefore, to step forward one minute it is only necessary to change the heading from 'mh' to 'mh+6', i.e. by means of:

```
make "mh :mh + 6
```

This takes the last value or number given to 'mh' and adds 6 to it before putting it back in memory. In the first instance 'mh' is of course set by the user but this value will be changed by adding 6 degrees to it every 60 seconds.

Once the heading is set, the new minute hand can be drawn by means of:

```
pd seth :mh fd 70 pu setpos [0 0]
```

This will point the turtle in the new direction

This will re-draw the minute hand before homing the turtle

Similar commands are used to erase and re-draw the hour hand.

The way in which this is worked out is as follows:

1 minute = 6 degrees; the hour hand will move the equivalent distance to 5 minutes in 1 hour which is $6 \times 5 = 30$ degrees. The hour hand will therefore need to move $30/60$ degrees every minute, i.e. 0.5 degrees.

Once the new features are added, the 'clock' procedure will look like this:

```
to clock
wait 3600
seth :mh pe fd 70 pu setpos [0 0]
make "mh :mh + 6
pd seth :mh fd 70 pu setpos [0 0]
seth :hh pe fd 40 pu setpos [0 0]
make "hh :hh + 0.5
pd seth :hh fd 40 pu setpos [0 0]
end
```

The working clock is now displayed by means of:

```
to finish
pu setpos [0 0] seth 0 pd
face
hands
repeat 720 [clock]
end
```

This will run the 'clock' procedure for 12 hours!

If you leave this procedure running for say ten minutes you will find that it is running slow. This is because it is counting for 1 minute then going through the hand-changing procedure before starting to count for another minute. So in real time each minute of the procedural clock takes 1 minute (3600 counts) PLUS the time it took to erase and re-position the clock hands.

Save your procedures on disc, clear the screen and take the 'wait 3600' line out of 'clock' and then type 'finish'.

There is the clock face; input the minute and hour hand positions and the clock will race away. Apart from it being visually interesting these procedures serve to show that erasing and re-drawing the hands of the clock takes time and, therefore, if your clock is to keep real time the procedure to start repositioning the hands should start before one minute has elapsed.

Clear the screen and re-run the fast version of 'finish'. Set the hands to say 6 o'clock, 'mh=360', 'hh=180', and then find out how long it takes for the clock to move to 6-15.

Having discovered that it takes time for your computer to operate, reload the 'face', 'hands', 'clock' and 'finish' procedures.

Now edit the timing in the 'clock' procedure to read say:

```
wait 3490
```

This will start the hand-changing routine just before a minute of 'real time' is up and the clock will keep better time.

Now to really polish up the project, let's add some 'chimes'.

Do you remember Chapter 6 when we defined some procedures to play a musical scale? Well, now we can make use of them by calling them up within the 'clock' procedure. If you haven't got these procedures in memory or on disc, you can redefine them in the following way:

```
to c1
sound [1 478 50]
end
```

```
to d
sound [1 426 50]
end
```

```

to e
sound [1 379 50]
end

to f
sound [1 358 50]
end

to g
sound [1 319 50]
end

to a
sound [1 284 50]
end

to b
sound [1 253 50]
end

to c2
sound [1 239 50]
end

```

First of all try this in direct mode:

```
a f g c1 c1 g a f
```

This produces a ‘Westminster’ chime. What is needed is for the clock to perform this every fifteen minutes. It is necessary, therefore, for the program to detect just when the clock’s hands are in a position that calls for chiming. As these positions occur at 90, 180, 270 and 360 degrees, this would seem to be the best criterion to use. A program line can be used that says something like:

```
if :m = 90 [PLAY THE CHIMES]
```

To incorporate this, edit the ‘clock’ procedure to include chimes every hour, half hour and quarter hour so that it looks like this:

```

to clock
wait 3490
seth :mh pe fd 70 pu setpos [0 0]
make "mh :mh + 6
pd seth :mh fd 70 pu setpos [0 0]
seth :hh pe fd 40 pu setpos [0 0]
make "hh :hh + 0.5
pd seth :hh fd 40 pu setpos [0 0]
if :mh = 90 [a f g c1]
if :mh = 180 [a f g c1 c1 g a f]
if :mh = 270 [a f g c1 c1 g a f a f g c
1]
if :mh = 360 [a f g c1 c1 g a f a f g c
1 c1 g a f]
end

```

You must remember when you set the ‘mh’ heading to make sure that the heading plus a multiple of 6 will add up to 90.

As a test of your abilities you could now go on to make your clock strike the hour on the hour.

You will also find, given the inclusion of chimes, that your clock needs retiming.

Continuing with the clock theme, let’s turn our attention to a digital clock. This could be elaborated by the addition of an alarm.

This chapter will now consider the bare bones of the procedures upon which a digital clock can be built.

First to input the starting conditions – the hour setting under the name ‘hour’ and the minute setting – ‘minute’:

```

to ask
make "hour rl make "hour item 1 :hour
make "minute rl make "minute item 1 :m
inute
end

```

The ‘ask’ procedure halts execution and expects an input. This it assigns to the variable ‘hour’. It then asks for another number from the user and assigns this to the variable ‘minute’. The next process is to display the input data:

```

to work
ts pr (se :hour " :minute)
This will print the number associated with the
variable 'hour'; in the first instance this will be the
number put in by the user.

This will print a space

wait 200
This will mean that early versions of
the digital clock will run fast but you
can rectify this later on.

ct
You will want the time re-printed in
the same place every minute so you
can use this to 'home' the text
cursor. This also can be modified
later on.

if :hour = 24 [make "hour Ø]

```

In these 'core procedures', a 24-hour clock has been designed. If you wanted to, you could make it into a 12 hour clock with an AM/PM indication. Either way, once the variable 'hour' has reached a certain value it must be re-set and the above command is the way of doing just that in Logo.

```
make "minute :minute + 1
```

This line adds one to the value of the variable 'minute' every time the computer has counted 200.

```
if :minute = 60 [make "minute Ø make "hour :hour + 1]
```

There are only 60 minutes in an hour, and so the moment that 60 minutes have elapsed, 'minute' is revalued at 0 and 'hour' is increased by 1.

```
end
```

The clock procedures should look like this:

```
to ask
make "hour rl make "hour item 1 :hour
make "minute rl make "minute item 1 :mi
nute
end

to work
ts pr (se :hour " :minute)
wait 200
ct
if :hour = 24 [make "hour 0]
make "minute :minute + 1
if :minute = 60 [make "minute 0 make "h
our :hour + 1]
end
```

Then to finish it off:

```
to finish
ask
repeat 720 [work]
end
```

Try out ‘finish’.

You will see that it works. Now hit the ESCape key.

Try the clock again. This time, however, set it at 23 58 and see what happens.

That’s odd! Why has it put up the time as

24 0

when it should read

0 0

at midnight?

Have another look at ‘work’. If you study the procedure closely you will see that the line that reads:

```
if :hour = 24 [make "hour Ø"]
```

is in the wrong position because ‘hour’ won’t change to 24 until the last command has been obeyed. So move the command to the first line of ‘work’ so that the procedure now looks like this:

```
to work
if :hour = 24 [make "hour Ø"]
pr (se :hour " :minute)
wait 22Ø
cs
make "minute :minute + 1
if :minute = 6Ø [make "minute Ø make "h
our :hour + 1]
end
```

The raw working procedures now look like this:

```
to ask
make "hour r1 make "hour item 1 :hour
make "minute r1 make "minute item 1 :mi
nute
end

to work
if :hour = 24 [make "hour Ø"]
pr (se :hour " :minute)
wait 22Ø
cs
make "minute :minute + 1
if :minute = 6Ø [make "minute Ø make "h
our :hour + 1]
end

to finish
ask
repeat 72Ø [work]
end
```

Now you have to design an alarm setting system for your clock.

First of all however, let's tidy up the 'ask' procedure with a few messages and set a couple more variables to represent the hour and minute alarm settings. Take a look at the following:

```
to ask
pr [what hour setting?]
make "hour rl make "hour item 1 :hour
pr [what minute setting?]
make "minute rl make "minute item 1 :mi
nute
pr [what alarm hour?]
make "ah rl make "ah item 1 :ah
pr [what alarm minute?]
make "am rl make "am item 1 :am
end
```

In the enlarged version of 'ask', we've simply added some messages and set two more variables, namely:

'ah' for the alarm hour

and

'am' for the alarm minute.

Now in order for the alarm to be sounded at a given time two conditions must be met.

First of all, the variable 'hour' must be equal to the variable 'ah' AND the variable 'minute' must be equal to the variable 'am'.

To help do this we can make use of a very useful piece of Logo syntax:

AND (and)

This will output 'TRUE' if all of the input expressions are true.

Try this in direct mode:

```
and (6 < 10) (5 > 4) (6 / 3 = 2)
```

and you will receive the message

TRUE
TRUE

because the first and second expressions are true and the first and third expressions are true.

Now try:

and (6 < 10) (1 = 2) (6 / 3 = 2)

Now we can make use of ‘and’ in the ‘alarm’ procedure below.

```
to alarm
if and (:hour = :ah) (:minute = :am) [noise]
end
```

All this short procedure is saying is that ‘if both expressions are true then go to a procedure called ‘noise’.

```
to noise
sound [1 20 50]
pr [TIME TO GET UP]
end
```

You can now include ‘alarm’ in the work procedure as shown below.

```
to work
if :hour = 24 [make "hour 0]
ts pr (se :hour " :minute)
wait 200
ct
make "minute = :minute + 1
if :minute = 60 [make "minute 0 make "
hour :hour + 1]
alarm
end
```

All that is left now is to adjust the ‘wait 200’ command in ‘work’ so that the digital clock keeps the correct time. You might like to start by trying

wait 3490

Chapter 10

In this section of the book we are going to experiment with ‘text handling’. This process is vital to any computer language and Logo is no exception. So far you have dealt mostly with numbers, but for the solution of many problems it is necessary to input, store and recall items other than numbers. It might, for example, be desirable to allow the user to answer ‘Yes’ or ‘No’ to questions rather than using 1 or 2 to indicate the response.

Let’s start off with a text handling command called:

FIRST (first)

This is a Logo operation that outputs the first element of a given list of objects.

In direct mode type:

```
pr first "elephant
```

As you might expect, an ‘e’ has been printed. The letter ‘e’ is the FIRST in the string of text.
Try this:

```
pr first "9pills
```

And there is the first character in the string, which in this case is the nine.

Now try this:

```
pr first "<>
```

and you will get an error message because the computer tries to do some mathematics with these symbols, but finds that it doesn’t have enough numbers to work with. So now try:

```
pr first [<>]
```

It works with groups of characters as well. (You call them words.)

Try:

```
pr first [MEN RUN FAST]
```

and the word 'MEN' is printed on the left of the screen. Note that if you use a series of words, they must be listed within square brackets.

Now try:

```
pr first [AZZLOGOWIG 1]
```

So you can see that 'first' applies to either the first character in a string of characters, or the first string in a series of strings. You can of course include this in other commands, for example

```
repeat 5 [pr first [BIG BOY]]
```

This will print 'BIG' five times one under the other.

You might like to try the following:

```
pr first item 3 [nice small dog]
```

and the 'first' of 'item 3' of the list is printed.

Now try:

```
pr item 2 [AZZLOGOWIG 1]
```

and there as you might expect, the computer has obeyed and printed '1'.

There are obviously many combinations which will allow you to pick out the words or characters of your choice.

Try these:

```
pr first [ONE GIRL] → ONE  
pr item 2 [ONE GIRL] → GIRL  
pr first item 2 [ONE GIRL] → G  
pr first item 1 [ONE GIRL] → O
```

Well that's not bad for a start, but how could you access and print the 'N' of 'ONE' and the 'IR' of 'GIRL'? There are two commands which will help:

BUTFIRST (bf)

This outputs the whole of an object except the FIRST element.

BUTLAST (bl)

BUTLAST outputs the whole of an object except the last element.

Try these commands:

pr bl [ONE GIRL] → ONE

now try:

pr bl first [ONE GIRL] → ON

All but the last of the first string.

now try:

pr bf first [ONE GIRL] → NE

and now:

pr bf bl first [ONE GIRL] → N

and

pr bf bl item 2 [ONE GIRL] → IR

pr bf bf bl item 2 [ONE GIRL] → R

pr bf bf bf bl bl "AZZLOGOWIG → LOGO

You can now see that with these four commands at your disposal, you can strip down text to display anything you like.

Let's consider a series of words such as a sentence.

An example of a simple sentence might be:

'The dog sat in the house
and sniffed his nice bone'.

You can label the various words (or strings) in the following way.

The dog sat in the house and sniffed his nice bone.
↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
article verb article noun conjunction pronoun noun
noun preposition noun verb adjective

Now it might be quite entertaining if you developed lists of articles, nouns, verbs etc. and were able to call them off in a random fashion to create random sentences.

Before you start your lists let's develop a procedure that will pick out a random word in a set of words.

First of all, let's think of a list.

For example:

CLEAN NICE BAD SIMPLE GOOD UGLY

Now it is possible to set a variable to equal this list of names by saying:

```
make "zz [CLEAN NICE BAD SIMPLE GOOD U  
GLY]
```

Try this command in direct mode then enter:

```
pr bl :zz
```

and your string has been printed with the exception of the last in the grouping, i.e. UGLY.

You will of course realise that you can now manipulate 'zz' and display whatever word or letter you wish to.

For example try:

```
pr bl bl bf bf bf bl bl bf bf :zz
```

and there, nothing has been displayed; in fact you will get an error message. This is because the first command to be obeyed is that next to 'zz'. Now starting with that command, work backwards and you will see why nothing is printed.

If you wanted the 'M' of 'SIMPLE' you could say:

```
pr item 3 item 4 :zz
```

Well, it's quite easy to see that you can strip off words by using the 'ITEM', 'FIRST', 'BF' and 'BL' commands, and that you would do this a random number of times to leave a shortened list of words. All one would then have to do is to print the first of what remains. Have a look at the two lines of a procedure to do this, making sure that you can follow what is happening.

```
to pick
make "zz [CLEAN NICE BAD SIMPLE GOOD U
GLY]
```

Now this line will set the variable 'zz' to the words inside the square brackets.

```
repeat random 6 [make "zz bf :zz]
```

This line will strip down the list of words a random number of times; for example if '2' were the random number chosen, 'zz' would become:

```
BAD SIMPLE GOOD UGLY
```

Now all that has to happen is that we can say either:

```
pr item 1 :zz
```

or

```
pr first :zz
```

The latter is probably better, so as the third line in our explanatory 'pick' procedure, we can say:

```
make "zz first :zz
pr :zz
end
```

Now let's have a look at those lines.

```
to pick
  make "zz [CLEAN NICE BAD SIMPLE GOOD U
  GLY]
  repeat random 6 [make "zz bf :zz]
  make "zz first :zz
  pr :zz
  end
```

This will pick a random number
and strip down the list.

This will make 'zz' equal to the first
word in the remaining list of words.

This will print 'zz'

Enter the 'pick' procedure then type 'pick' and a randomly selected string will be printed.
You could do this as many times as you wished, or alternatively you could say:

```
ct repeat 20 [pick]
```

You are now ready to try some random sentence generation. But first another text-handling operation.

WORD (word)

This is a Logo operation that outputs a word made up of its inputs.

Let's look at 'word' by trying it out. Type:

```
pr (word "he "ll "o)
```

You can see that the command 'word' takes all the objects and brings them together to form a single word.

As you know, 'sentence' does a similar job. Try:

```
pr (se "good "old "boy)
```

and there, your sentence has been printed. We will be coming back to take a further look at these operations and their use a little later on.

Let's now make up a list of groups of words for our random sentence generation procedure.

The following are only suggestions and you are obviously free to make up your own sentences in whatever way you like.

Article	Noun	Verb	Preposition	Article	Noun	Conjunction
THE	DOG	SAT	IN	THE	HOUSE	AND
A	MAN	STOOD	ON	A	CAR	THEN
	BOY	LAY	UNDER		VAN	
	CAT	HID	BESIDE		SCHOOL	
	DAD		BEFORE		SHOP	
	TEACHER				BIKE	
	WOMAN				TABLE	
	GIRL				BED	
	MUM					

Verb	Pronoun	Adjective	Noun
SMELT	HIS	NICE	BONE
LIKED	ITS	NASTY	FOOT
HIT	HER	BAD	SOCK
WASHED		BIG	TOE
CUDDLED		SMALL	SPANNER
BROKE		HEAVY	COMPUTER
		DIRTY	
		SMELLY	

From these lists there are 9,953,280 possible sentences that could be generated. Let's get them into procedures!

```
to list1
make "art [The A]
make "noun [dog man woman girl cat boy
    child teacher mum dad]
make "verb [sat stood lay hid]
make "prep [in on under beside before]
make "arrt [the a]
make "nwn [house car van school shop b
    ike table bed]
make "conj [and then]
make "vrob [smelt licked hit washed cu
    ddled broke]
make "pron [his its her]
make "adj [nice nasty bad big small he
    avy dirty smelly]
make "nun [bone foot sock toe spanner
    computer]
end
```

Obviously if we have more than one list of a particular part of speech we must set each one to a different variable. Thus, the lists of nouns will be assigned the names:

‘noun’, ‘nwn’, and ‘nun’.

If you now type

```
list1 pr :vrob
```

the contents of the variable ‘vrob’ are dutifully displayed.

Now to define a procedure to choose the randomly selected words before putting them into a sentence.

```

to choose
repeat random 2 [make "art bf :art] ma
ke "art first :art
repeat random 9 [make "noun bf :noun]
make "noun first :noun
repeat random 4 [make "verb bf :verb]
make "verb first :verb
repeat random 5 [make "prep bf :prep]
make "prep first :prep
repeat random 2 [make "arrt bf :arrt]
make "arrt first :arrt
repeat random 8 [make "nwn bf :nwn] ma
ke "nwn first :nwn
repeat random 2 [make "conj bf :conj]
make "conj first :conj
repeat random 6 [make "vrob bf :vrob]
make "vrob first :vrob
repeat random 3 [make "pron bf :pron]
make "pron first :pron
repeat random 8 [make "adj bf :adj] ma
ke "adj first :adj
repeat random 6 [make "nun bf :nun] ma
ke "nun first :nun
end

```

In your last procedure you can say:

```

to write
list1 choose
pr (se :art :noun :verb :prep :arrt)
pr (se :nwn :conj :vrob :pron)
pr (se :adj :nun)
end

```

Then enter ‘write’!

Before going on to look at some random poetry, it’s worth mentioning a Logo command that has not as yet been used in any of your Logo procedures.

Although we haven’t made use of it, a command that might prove useful to you in the future is:

FIRST PUT (fput)

This will output a new object made by adding the first object to the front of the second object.

Try:

```
pr fput "box "ing
```

or better still, if you still have 'list1' in memory

```
list1 pr fput :vrob :nwn
```

Now, as promised, some random poetry. Have a look at the following procedures and then try them out.

```
to man
make "zz [Frenchmen Russians Scotsmen
Englishmen Americans Irishmen Africans
]
make "xx [red orange green indigo purple puce violet]
make "cc [Charlton Liverpool Exeter Spurs Burnley Glasgow Hatfield Rangers]
end

to choose3
repeat random 7 [make "zz bf :zz] make
"zz first :zz
repeat random 7 [make "xx bf :xx] make
"xx first :xx
repeat random 8 [make "cc bf :cc] make
"cc first :cc
end
```

```
to rhyme
man choose3
pr (se :zz "are :xx)
man choose3
pr (se :zz "are "blue)
pr (se :cc "six)
man choose3
pr (se :cc "two)
end
```

Now type 'rhyme'.

There are 16807 options to this verse. It will take you a long time to come up with the results that you would like to see on a Saturday scoreline.

Chapter 11

In this chapter we are going to make use of most of the commands and operations previously described, and introduce one or two new ones which will extend your knowledge and enable the creation of even more complex procedures. The theme chosen is one of a game in which the computer thinks of a four digit number and the user has to guess the correct digits in the correct order, for example if the computer thinks of:

1 7 4 6

and the user's first guess is:

7 3 9 6

the computer will give the user the message:

"1 cat and 1 dog"
"Not quite right"

Which means that one of the numbers input by the user is correct but is in the wrong place (the 7) and that one of the numbers is correct and in the correct place (the 6).

The user gets an indefinite number of guesses and continues playing until s/he receives a message like:

"4 cats"

after which the user will have "Got it right" and will receive a noisy ovation and a message like

"You did it in 999 goes"

In order to compare each section of a randomly chosen four-digit number with each part of a user's choice, the computer must assess each section separately. To this end, under a procedure called 'finish', four variables (n w t f) plus a counting variable 'zz' have been set.

Type in the following procedure and then enter the separate line at the end in direct mode:

```
to finish
ct ts
make "zz ""
make "n random 10
make "w random 10
make "t random 10
make "f random 10
end

finish pr (se :n :w :t :f)
```

A four digit number will have been printed with the first element set to the variable ‘n’, the second to ‘w’, the third to ‘t’ and the fourth to ‘f’.

The variable ‘zz’ will be used to count the number of guesses by the user and this information will be conveyed in the concluding message:

```
"YOU DID IT IN ZZ GOES"
```

Now to define the central core of the game, procedure ‘try’. Firstly though, we need to think very carefully about what we want to happen.

- 1) We will need four inputs from the user, so to indicate that a four digit number is needed the input is:

```
pr [put in a 4 digit number please]
make "input rl make "input item 1 :input
```

- 2) Now we need to analyse each segment of that string and so must set a separate variable for each part:

```
make "a bl bl bl :input
make "b bf bl bl :input
make "c bf bf bl :input
make "d bf bf bf :input
```

Now these lines will strip down the input so that the variable ‘a’ will be set to the first digit, ‘b’ to the second digit, ‘c’ to the third and ‘d’ to the fourth. So far then, the ‘try’ procedure looks like this:

```

to try
pr [put in a 4 digit number please]
make "input rl make "input item 1 :input
ut
make "a bl bl bl :input
make "b bf bl bl :input
make "c bf bf bl :input
make "d bf bf bf :input
end

```

In the next part of the procedure, the variables have to be compared. For example,

computer's choice: 1 7 4 6

user's guess: 4 7 1 \emptyset

The computer will have assigned its own choice to the variables as shown below:

$n=1, w=7, t=4, f=6$

The procedure 'finish' has carried out the assignment process. Now, if the user's first guess is 4710, the procedure 'try' will have assigned the variables:

$a=4, b=7, c=1, d=\emptyset$

Now some procedures must be devised to examine the two sets of variables and carry out the necessary comparisons. These procedures will report back on the user's performance. This report will tell how many numbers have been guessed correctly but are in the wrong place and how many numbers are correct and in the right place. There are many ways of carrying out this process: the one chosen here is to call each wrongly placed but correct number a dog and each correctly placed correct number a cat.

Thus, some reports:

computer's choice:	1	7	4	6
user's guess:	4	7	1	Ø
	number correct but in wrong place	number correct and in correct place	number correct but in wrong place	number wrong
	DOG	CAT	DOG	

score = 1 CAT and 2 DOGS

and...

computer's choice:	1	7	4	6
user's guess:	1	7	6	4
	number correct and in correct place	number correct and in correct place	number correct but in wrong place	number correct but in wrong place
	CAT	CAT	DOG	DOG

i.e. score = 2 CATS and 2 DOGS

Finally, a totally correct guess scores four cats and then the number of attempts taken is reported.

This process of checking must be done systematically: a series of four procedures is provided to check each character of the user's guess in turn and build up the score. Take the one to check the first character. It will say:

- (i) Does first character of guess (a) equal first character of number (n)?
That is,

IF N=A...

If it does then one CAT is scored i.e.

IF N=A score one CAT

This line is, of course, not a valid Logo statement so it must be translated. One way to do this is to create a procedure called 'c' (for Cat) which increments the value CAT each time it is called. Thus the line becomes

if :n = :a [c]

- (ii) Does second character of guess equal first character of number?

IF W=A...

This time, if the condition is true, a DOG is scored, this being achieved by means of a procedure 'd' for DOG.

if :w = :a [d]

(iii)

Does third character of guess equal first character of number? If so score a DOG i.e.

if :t = :a [d]

(iv)

Does fourth character of guess equal first character of number? If so score a DOG i.e.

if :f = :a [d]

Thus, the whole procedure becomes:

```
to one
if :n = :a [c]
if :w = :a [d]
if :t = :a [d]
if :f = :a [d]
end
```

The two procedures to keep the score are, quite simply

```
to d
make "dog :dog + 1
end
```

```
to c
make "cat :cat + 1
end
```

Each time one of these procedures is called, the relevant variable is incremented. This gives rise to one problem: if the game is played twice the score of CATs or DOGs will keep on rising. So, each time it is started again these variables must be set to zero or 'initialised' i.e.

```
make "cat Ø make "dog Ø
```

Giving the procedure 'one' a dummy run using a guess of 4170 and a computer choice of 1746 gives:

```
a = 4
n = 1      w = 7      t = 4      f = 6
```

When these numbers are used, the resulting steps are as follows

if :n = :a [c]	IF 1=4 [C]	c=Ø
if :w = :a [d]	IF 7=4 [D]	d=Ø
if :t = :a [d]	IF 4=4 [D]	d=1
if :f = :a [d]	IF 6=4 [D]	d=Ø

The other procedures to sort out the second, third and fourth numbers are much the same, i.e.

```
to two
if :n = :b [d]
if :w = :b [c]
if :t = :b [d]
if :f = :b [d]
end
```

```
to three
if :n = :c [d]
if :w = :c [d]
if :t = :c [c]
if :f = :c [d]
end
```

```
to four
if :n = :d [d]
if :w = :d [d]
if :t = :d [d]
if :f = :d [c]
end
```

Edit the 'try' procedure to look like this:

```
to try
make "cat Ø make "dog Ø
pr [put in a 4 digit number please]
make "input rl make "input item 1 :input
make "a bl bl bl :input
make "b bf bl bl :input
make "c bf bf bl :input
make "d bf bf bf :input
one two three four
make "zz :zz + 1
end
```

This line sets the variables 'cat' and 'dog' to zero.

This line adds one to the counting (no. of guesses) variable 'zz'.

This line evaluates the input by calling these procedures.

It is possible to tell when the player has guessed the number as the variable ‘cat’ at that point will equal 4. A more direct test is possible using the one-to-one correspondence of the elements of the guess with elements of the number to be guessed.

This will make use of the ‘and’ conditional. What we can do is to say:

“If the first of the computer’s numbers equals the first number in the user’s guess ‘and’ the second of the computer’s numbers equals the second number in the user’s guess, then carry out the procedure ‘next’.”

In Logo, this will read:

```
if and (:n = :a) (:w = :b) [next]
```

Now if the first two digits of the user’s guess are correct (cats) the procedure ‘next’ is run. This procedure will evaluate the third and fourth digits in exactly the same way as the above line:

```
to next
  if and (:t = :c) (:f = :d) [make "p 9
    game]
  end
```

Now if ‘game’ is called all four digits of the user’s guess must be of the correct value and in the right positions. Therefore a congratulatory message is in order.

```
to game
  ct ts repeat 5 [pr [well done]]
  pr (se "you "did "it "in :zz "goes)
  end
```

You will have noticed that before ‘game’ was called, a variable ‘p’ was set to 9. Now this is going to be used to stop the whole procedure by adding a line to ‘try’ which reads

```
if :p = 9 [stop]
```

Of course this line will only be obeyed once the conditions set in ‘next’ have been fulfilled. It also means that the ‘p’ variable will be set to say ‘0’ at the beginning of ‘try’.

The 'try' procedure will by now look like this:

```
to try
  make "cat Ø make "dog Ø make "p Ø
  pr [put in a 4 digit number please]
  make "input rl make "input item 1 :input
  make "a bl bl :input
  make "b bf bl bl :input
  make "c bf bf bl :input
  make "d bf bf bf :input
  one two three four
  make "zz :zz + 1
  if and (:n = :a) (:w = :b) [next]
  pr (se :c "CATS "and :d "DOGS)
  wait 2ØØ
  if :p = 9 [stop]
  try
end
```

These last few lines of 'try' should need very little explanation. The user sees the score in terms of cats and dogs, views this for about three seconds and is then returned to 'try' for another go. Here are the developed procedures. You may of course alter, adjust or add to them as you like. (Note the addition of 'try' to procedure 'finish'.)

to finish	to d
make "zz Ø	make "dog :dog + 1
make "n random 1Ø	end
make "w random 1Ø	
make "t random 1Ø	to one
make "f random 1Ø	if :n = :a [c]
try	if :w = :a [d]
end	if :t = :a [d]
to c	end
make "cat :cat + 1	
end	

```

to two                      to four
if :n = :b [d]              if :n = :d [d]
if :w = :b [c]              if :w = :d [d]
if :t = :b [d]              if :t = :d [d]
if :f = :b [d]              if :f = :d [c]
end                         end

to three
if :n = :c [d]
if :w = :c [d]
if :t = :c [c]
if :f = :c [d]
end

to try
make "cat Ø make "dog Ø make "p Ø
pr [put in a 4 digit number please]
make "input rl make "input item 1 :input
t
make "a bl bl bl :input
make "b bf bl bl :input
make "c bf bf bl :input
make "d bf bf bf :input
one two three four
make "zz :zz + 1
if and (:n = :a) (:w = :b) [next]
pr (se :c "CATS "and :d "DOGS)
wait 2ØØ
if :p = 9 [stop]
try
end

to next
if and (:t = :c) (:f = :d) [make "p 9 g
ame]
end

```

```

to game
ct ts repeat 5 [pr [well done]]
pr (se "you "did "it "in :zz "goes)
end

```

Now just as you were about to amaze your friends with this marvellous game, you discover that sometimes when you play the game the results in terms of cats and dogs are garbled rubbish and make no sense at all. In fact they are not rubbish and the computer is doing exactly as it is told. The fact is that the procedures are faulty and that messages like:

2 cats and 6 dogs

are very logical and understandable results of the procedures given their present stage of development. These 'garbled' messages occur only when the computer picks the same digit twice. Let's now outline what is happening within the procedures that results in what at first glance seems to be an illogical message. To help explain this, assume that the computer has thought of the number

9 9 1 1

and that the user's input number is say

1 1 1 1

Now if everything was working absolutely correctly one would expect a message saying

2 cats and Ø dogs

instead of which you will get

2 cats and 6 dogs

and here's the reason why:

n	w	t	f	a	b	c	d
9	9	1	1	1	1	1	1

Now the computer's number will have been assigned to the variables 'n', 'w', 't' and 'f' as shown above; each of the input digits will have been assigned to the variables 'a', 'b', 'c' and 'd'. Now look very closely at procedure 'one' and answer all the questions.

```
IF N=A ADD 1 TO CAT results in CAT=∅ (NO CHANGE)
IF W=A ADD 1 TO DOG results in DOG=∅ (NO CHANGE)
IF T=A ADD 1 TO DOG results in DOG=1 (TRUE)
IF F=A ADD 1 TO DOG results in DOG=2 (TRUE)
```

Thus given the computer's chosen number of 9911 and the input guess of 1111, after running through 'one' the value of 'dog' is already 2. Now let's look at 'two':

```
IF N=B ADD 1 TO DOG results in DOG=2 (NO CHANGE)
IF W=B ADD 1 TO CAT results in CAT=∅ (NO CHANGE)
IF T=B ADD 1 TO DOG results in DOG=2+1 (TRUE)
IF F=B ADD 1 TO DOG results in DOG=3+1 (TRUE)
```

'Dog' now has a value of 4 and 'cat' has a value of zero. Now watch their values as 'three' is worked through.

```
IF N=C ADD 1 TO DOG results in DOG=4 (NO CHANGE)
IF W=C ADD 1 TO DOG results in DOG=4 (NO CHANGE)
IF T=C ADD 1 TO CAT results in CAT=∅+1 (TRUE)
IF F=C ADD 1 TO DOG results in DOG=4+1 (TRUE)
```

'Dog' now has a value of 5 and 'cat' a value of 1.

Finally 'four':

```
IF N=D ADD 1 TO DOG results in DOG=5 (NO CHANGE)
IF W=D ADD 1 TO DOG results in DOG=5 (NO CHANGE)
IF T=D ADD 1 TO DOG results in DOG=5+1 (TRUE)
IF F=D ADD 1 TO CAT results in CAT=1+1 (TRUE)
```

Thus 'cat' finishes with a value of 2 and 'dog' finishes with a value of 6.

Which is logical given the procedures, but of no use to the player. We must therefore consider the procedures a little more closely and modify them.

The trouble in the case above is that 't' and 'f' are being counted eight times instead of twice. Now ideally the cats should be counted first and having been counted once they should be given a value that will stop them from being counted again. A convenient value is zero. However, the computer could choose zero as one or more of the random numbers when it is asked to 'make "f random 10', so we will have to fix this. First of all though, let's take out the commands that increase the value of the variable 'cat' and put them in one procedure called 'cats'.

```

to cats
if :n = :a [c make "n Ø]
if :w = :b [c make "w Ø]
if :t = :c [c make "t Ø]
if :f = :d [c make "f Ø]
end

```

You will notice that the first line of ‘one’, the second line of ‘two’, the third line of ‘three’ and the fourth line of ‘four’ have, in effect, been taken and placed in the separate procedure, ‘cats’.

Notice also that once a variable ‘n’, ‘w’, ‘t’ or ‘f’ has been counted it is assigned a value of zero thus barring it from being counted twice. Now let’s edit the other four procedures to look like this:

<pre> to one if :w = :a [d make "w Ø] if :t = :a [d make "t Ø] if :f = :a [d make "f Ø] end </pre>	<pre> to three if :n = :c [d make "n Ø] if :w = :c [d make "w Ø] if :f = :c [d make "f Ø] end </pre>
<pre> to two if :n = :b [d make "n Ø] if :t = :b [d make "t Ø] if :f = :b [d make "f Ø] end </pre>	<pre> to four if :n = :d [d make "n Ø] if :w = :d [d make "w Ø] if :t = :d [d make "t Ø] end </pre>

Again notice that once two variables have been seen to be the same, ‘dog’ is increased by one and the computer-chosen variable set to zero.

Let’s see how this works in practice using the previous example:

n	w	t	f	a	b	c	d
9	9	1	1	1	1	1	1

CATS

$n = a ?$	NO, $c = \emptyset$
$w = b ?$	NO, $c = \emptyset$
$t = c ?$	YES, $c = 1, t = \emptyset$
$f = d ?$	YES, $c = 2, f = \emptyset$.

ONE

$w = a ?$	NO, $d = \emptyset$
$t = a ?$	NO, $d = \emptyset$ because 't' was set to 0 by 'cats'
$f = a ?$	NO, $d = \emptyset$ because 'f' was set to 0 by 'cats'

TWO

$n = b ?$	NO, $d = \emptyset$
$t = b ?$	NO, $d = \emptyset$ because 't' was set to 0 by 'cats'
$f = b ?$	NO, $d = \emptyset$ because 't' was set to 0 by 'cats'

THREE

$n = c ?$	NO, $d = \emptyset$
$t = c ?$	NO, $d = \emptyset$
$f = b ?$	NO, $d = \emptyset$ because 'f' was set to 0 by 'cats'

FOUR

$n = d ?$	NO, $d = \emptyset$
$w = d ?$	NO, $d = \emptyset$
$f = d ?$	NO, $d = \emptyset$ because 'f' was set to 0 by 'cats'.

If you now look at the value of the variables 'cat' and 'dog' you will see that

cats = 2
dogs = \emptyset

Which will give the message

2 CATS and \emptyset DOGS

Which, given the numbers

computer	user
9 9 1 1	1 1 1 1

is quite correct.

Now of course is the time to edit the relevant line of 'try' to read:

cats one two three four

There is now however another problem. The variables 't' and 'f' using our previous example now carry the value of zero and accordingly will never be equal to 'a' or 'b'. It is therefore necessary to find a way of switching them back to their original values.

The easiest way of doing this is to make some changes in 'finish'; at the same time the use of random can be changed so that we get numbers from 1 to 9 instead of from 0 to 9 as before.

```
to finish
ct ts
make "zz Ø
make "i 1 + random 9
make "j 1 + random 9
make "k 1 + random 9
make "l 1 + random 9
try
end
```

Note that the labels have been changed and now, at the beginning of 'try' you can say:

```
to try
make "n :i make "w :j make "t :k
make "f :l make "cat Ø make "dog Ø make
    "p Ø
```

You can see from these commands that as 'try' is re-run the variables 'n', 'w', 't' and 'f' are reset to their original values ready to compare with yet another guess from the user.

You will now of course have to compare

```
i to a  
j to b  
k to c  
l to d
```

This will mean changes in the ‘if and’ line of ‘try’:

```
if and (:i = :a) (:j = :b) [next]
```

Of course changes in ‘next’ will be necessary:

```
to next  
if and (:k = :c) (:l = :d) [make "p 9 g  
ame]  
end
```

Here are the final procedures:

```
to cats  
if :n = :a [c make "n Ø]  
if :w = :b [c make "w Ø]  
if :t = :c [c make "t Ø]  
if :f = :d [c make "f Ø]  
end
```

```
to c  
make "cat :cat + 1  
end
```

```
to d  
make "dog :dog + 1  
end
```

```
to one  
if :w = :a [d make "w Ø]  
if :t = :a [d make "t Ø]  
if :f = :a [d make "f Ø]  
end
```

```
to two
if :n = :b [d make "n Ø]
if :t = :b [d make "t Ø]
if :f = :b [d make "f Ø]
end

to three
if :n = :c [d make "n Ø]
if :w = :c [d make "w Ø]
if :f = :c [d make "f Ø]
end

to four
if :n = :d [d make "n Ø]
if :w = :d [d make "w Ø]
if :t = :d [d make "t Ø]
end

to next
if and (:k = :c) (:l = :d) [make "p 9
game]
end

to game
cs ts repeat 5 [pr [well done]]
pr (se "you "did "it "in :zz "goes)
end
```

```

to try
make "n :i make "w :j make "t :k
make "f :l make "cat 0 make "dog Ø make
  "p Ø
pr [put in a 4 digit number please]
make "input rl make "input item 1 :input
t
make "a bl bl bl :input
make "b bf bl bl :input
make "c bf bf bl :input
make "d bf bf bf :input
cats one two three four
make "zz :zz + 1
if and (:n = :a) (:w = :b) [next]
pr (se :cat "CATS "and :dog "DOGS)
wait 2ØØ
if :p = 9 [stop]
try
end

to finish
ct ts
make "zz Ø
make "i 1 + random 9
make "j 1 + random 9
make "k 1 + random 9
make "l 1 + random 9
try
end

```

Now type ‘finish’

Chapter 12

Solutions to Projects

Chapter One

Project 1

To find out the height of the screen you need to position the turtle at either the top or bottom of the screen and then move it all the way to the other edge.

- i) The best way is to start off from a known point by typing in ‘cs’. This will get the turtle to the ‘home’ position.
- ii) Next move the turtle to the top of the screen. Try a ‘forward 50’.

i.e. `fd 50 <ENTER>`

Not enough eh? Keep moving the turtle ‘forward’ until reaches the top of the screen and then add up all the small steps. If you want to check this, move the turtle home with a ‘cs’ and then do the ‘forward’ move in one go.

- iii) Now turn the turtle round by means of two ‘right’ commands;

```
rt 90<ENTER>
rt 90<ENTER>
```

- iv) The field is now clear for a run right down the screen. You can do this in bits as earlier or in one go.

A sample run is given below:

<code>cs <ENTER></code>	gets turtle home
<code>fd 50 <ENTER></code>	not far enough
<code>fd 100 <ENTER></code>	still not enough
<code>fd 60 <ENTER></code>	too far
<code>bk 10 <ENTER></code>	

Adding that up gives:

forwards: $50+100+60$
backs: 10

Doing the sums gives a distance from home to the top of the screen of $210-10=200$.

Next, turn around:

```
rt 90 <ENTER>
rt 90 <ENTER>
fd 350 <ENTER>      Oops! too far. Try
bk 40<ENTER>       Maybe 10 left?
fd 10<ENTER>
```

That's it! Now the sums

forwards: $350+10 = 360$
backs: 40

This gives a screen height of $360-40 = 320$.

Project 2

To find out the width of the turtle's field of movement clear the screen using 'cs' and, using a similar procedure to that outlined in the solution to project 1, you should discover that the screen width is 640 units.

Project 3

In solving the problem of the height and width of the turtle, let's first of all consider the height.

- i) The best way is to move the turtle back a set amount and then forward by the same amount and see by how much the line drawn is longer or shorter than the turtle.

- ii) Type in 'cs' which will place the turtle in the home position and now try:

`bk 20 <ENTER>`

and you will notice that the turtle has acquired a 'nose'.

- iii) Now, clear the screen and try

```
bk 18 <ENTER>
```

and you will see that the length of the "nose' has been halved.

- iv) Now try

```
cs <ENTER>  
bk 16 <ENTER>
```

and you will see that the line, 16 units in length, bisects the turtle and represents its height. Now let's use a similar method to find the length of half the base of the turtle. When you have done this, simply double that distance and you will have discovered the full width of the turtle:

- a) First of all clear the screen and turn the turtle to the right. Now try these commands:

```
fd 12 <ENTER>  
bk 12 <ENTER>  
lt 90 <ENTER>  
fd 5 <ENTER>
```

- b) You will notice that the 'fd 12' command needs to be reduced slightly, so try

```
cs <ENTER>  
rt 90 <ENTER>  
fd 10 <ENTER>  
bk 10 <ENTER>  
lt 90 <ENTER>  
fd 5 <ENTER>
```

and you will see that the 'tail' is shorter.

- c) Finally type:

```
cs <ENTER>  
rt 90 <ENTER>  
fd 9 <ENTER>  
bk 9 <ENTER>  
lt 90 <ENTER>  
fd 5 <ENTER>
```

Thus you will now have discovered that the maximum width of the turtle is twice nine, which is 18 screen units.

Project 4

You will have noticed that a small ‘box’ measuring twenty by eighteen units into which the turtle must fit is just a little larger than the turtle itself, and so some precise calculation is needed. Here’s the solution:

```
rt 90 <ENTER>
fd 58 <ENTER>
rt 90 <ENTER>
fd 62 <ENTER>
```

Project 5

The turtle is facing in the right direction to start the solution and so the actual answer to the problem will be something along the lines of:

```
fd<ENTER>
lt<ENTER>
fd<ENTER>
```

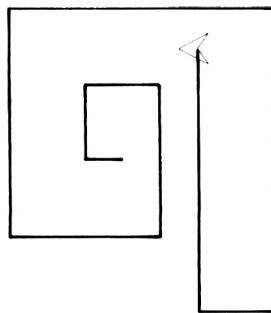
Follow this routine and you will see that if the forward distances are correctly computed the turtle will be returned to its original home position.

You will have noticed that the corridor width is 20 screen units. Therefore your first command should be:

```
fd 120<ENTER>
```

this is the ‘fd 130’ command less half the width of the corridor.

After a 'lt 90' command, you should have a figure that looks like this:



The next command will be

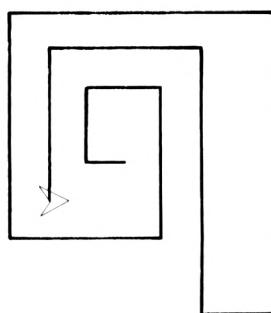
fd 60<ENTER>

Which is 'fd 70' minus the corridor half-width of 10.

Again the 'lt' command followed by

fd 90<RETURN>

After a 'lt 90' command, your screen will look like this:



You must now say:

fd 40 <ENTER> (50 minus 10)
lt 90 <ENTER>

Then

fd 50 <ENTER>

which is the corridor width plus the first 'fd 10' command.

Now can say

```
lt 90 <ENTER>
fd 20 <ENTER>
```

and the turtle will be nearly in its home position.

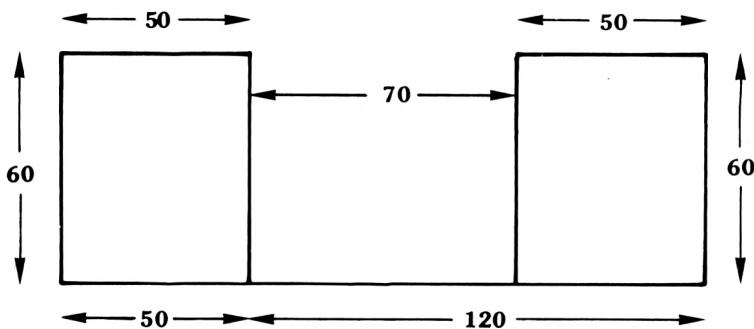
Thus your final solution will look like this:

```
fd 120 <ENTER>
lt 90 <ENTER>
fd 60 <ENTER>
lt 90 <ENTER>
fd 90 <ENTER>
lt 90 <ENTER>
fd 40 <ENTER>
lt 90 <ENTER>
fd 50 <ENTER>
lt 90 <ENTER>
fd 20 <ENTER>
lt 90 <ENTER>
fd 10 <ENTER>
```

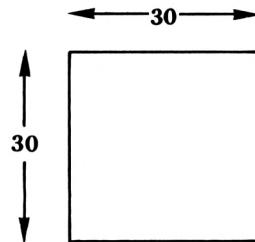
Chapter Two

Project 1

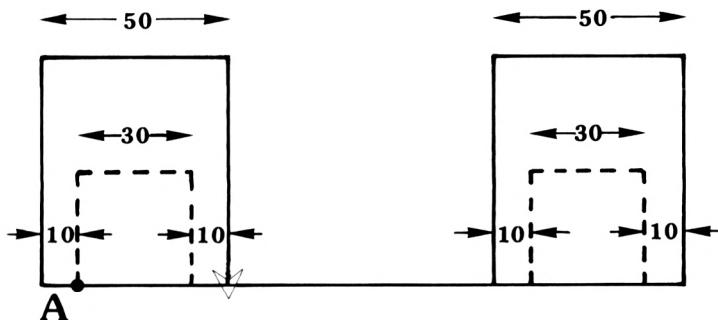
You can see from your screen that the shape created is symmetrical about the vertical axis and the first thing to do is to add some dimensions (sizes) to it. This is best done on paper and you should end up with a sketch that looks like this:



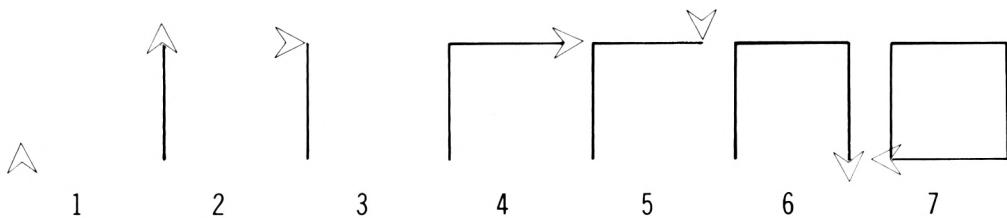
Now the 'box' that you have defined has these dimensions:



Now when you finish typing in the direct mode commands the turtle is in the position shown below:



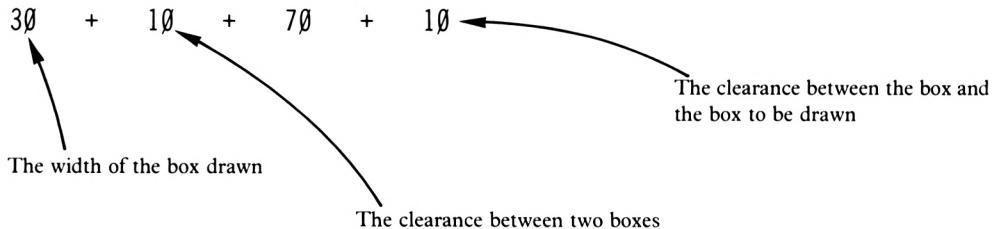
Now you must remember that under the procedure 'box', the rectangular shape is produced in the following way:



This being the case, the first two commands must move the turtle to the point 'A' where it can start to draw the box. Therefore the first commands must be

```
rt 90  
fd 40  
rt 90  
box
```

Having successfully drawn the box in the first rectangle the turtle must be repositioned to draw the second box. To do this it must be moved backwards by



Therefore the command is

`bk 120`

followed by

`rt 90
box`

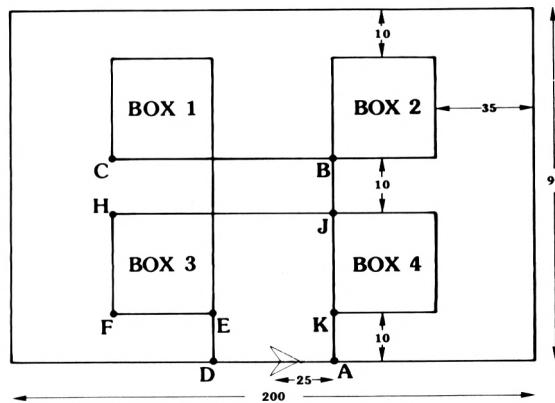
If you want to you could type 'ht' and your drawing is complete.

Project 2

In order to solve this problem it would be best to draw the maze to scale and then measure out the most direct route through it. If you do this you will probably find the following commands conform to your scale route map.

```
fd 66
lt 90
fd 66
lt 90
fd 48
lt 90
fd 40
lt 90
fd 30
lt 90
fd 18
lt 90
fd 17
ht
```

Project 3



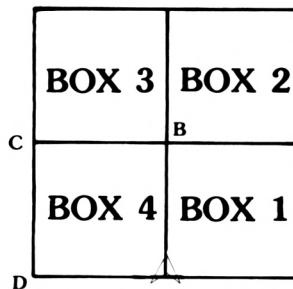
The sketch above shows the frame with the four boxes connected in the manner required by the question. The current position of the turtle is also shown. Here's a possible plan of action:

```

fd 25      (move to point A)
lt 90      (turn to face point B)
fd 50      (move to point B)
box        (draw box 2)
fd 80      (move to point C)
rt 90      (orientation)
box        (draw box 1)
bk 80      (move back to point B)
lt 90      (turn to face point J)
fd 40      (move to point K)
lt 90      (orientation)
lt 90      (orientation)
box        (draw box 4)
rt 90      (turn to face point J)
fd 30      (move to point J)
lt 90      (turn to face point H)
fd 80      (move to point H)
rt 90      (orientation)
bk 30      (move to point F)
box        (draw box 3)
bk 30      (move to point E)
rt 90      (orientation)
fd 40
bk 50      (finish drawing)

```

Project 4



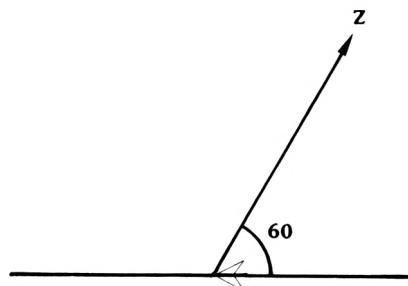
Again, planning is essential for accurate speedy drawing and the best thing to do is to plan on paper exactly how the pattern is to be created.

Here's a possible solution.

box	(draws box 1)
rt 90	(turn to face point B)
fd 30	(move to point B)
box	(draws box 2)
fd 30	(move to point C)
rt 90	(orientation)
box	(draws box 3)
rt 90	(orientation)
bk 30	(move to point D)
box	(draws box 4)

Project 5

The clue given in the question said that all the internal angles are 60 degrees as shown below:



If you want the turtle to move off in direction Z, it must turn to the right by 120 degrees ($180 - 60 = 120$).

Another feature of the pattern is that the space between the parallel lines is the same. Therefore every successive line must be a set amount longer than the previous line.

Here's a solution.

```
lt 90      (turns the turtle before drawing first line)

fd 10      (draws first line)
rt 120     (positions turtle to draw 2nd line)

fd 20      (draws 2nd line)
rt 120     (orientation)
fd 30      (note that the length is increased by a set amount each time
            a new line is drawn)

etc.
```

Chapter Three

First of all let's define a procedure to draw the following windows on the building.

```
to window1
repeat 4 [fd 10 rt 90]
end
```

For the tops of the trees you could use:

```
to tri
lt 90
fd 18
rt 120
fd 36
rt 120
fd 36
rt 120
fd 18
rt 90
end
```

Now you can write a procedure for one of the vertical lines of windows in the building by saying first of all

```
to line
pu
fd 30
pd
window1
end
```

and then say

```
to build
repeat 4 [line]
end
```

To start the building you can say:

```
to house
pd
fd 150
rt 90
fd 70
rt 90
fd 150
rt 90
fd 20
rt 90
build
end
```

and then to almost finish the house you can say:

```
to finish
house
pu
fd 10
lt 90
fd 30
lt 90
bk 30
pd
```

```
build
pu
fd 40
pd
lt 90
fd 50
end
```

Now to draw some trees, the procedure will be:

```
to tree
fd 30
tri
end
```

Now to draw the picture you can say:

```
to picture
pu
bk 50
pd
lt 90
fd 100
bk 20
rt 90
finish
pd
fd 20
lt 90
tree
end
```

Project 1

To draw this face it's suggested that you start off with the necessary commands to draw the outline of the person's head.

```
pu  
fd 50  
pd  
rt 90  
fd 25  
rt 90  
fd 50  
rt 90  
fd 50  
rt 90  
fd 50  
rt 90  
fd 50  
rt 90  
fd 25
```

Now to draw the mouth and one eye.

```
lt 90  
pu  
bk 40  
lt 90  
pd  
fd 10  
bk 20  
rt 90  
pu  
fd 20  
pd  
fd 5  
pu
```

and to finish off:

```
lt 90
fd 20
lt 90
pd
fd 5
lt 90
pu
fd 10
pd
rt 90
fd 10
ht
```

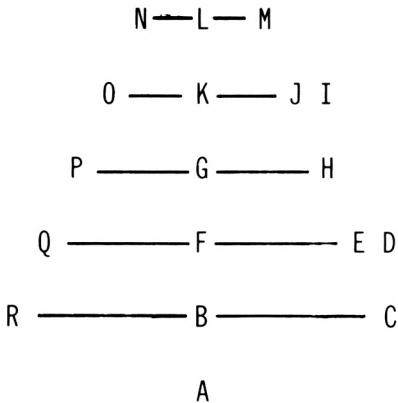
Project 2

Its up to you to try and draw your own name. If however it happens to be Andrew, here's a possible solution.

```
to name
pu fd 50 lt 90 fd 65 pd fd 5 lt 60 fd
20
bk 10 lt 120 fd 15 rt 60 fd 10 bk 20 p
u lt 60
15 pd rt 90 fd 18 bk 18 lt 45 fd 25
lt 135 fd 18 pu rt 90 fd 5 pd fd 10 rt
45
fd 10 rt 45 fd 10 rt 90 fd 18 rt 90 fd
18 rt 90
pu fd 23 pd fd 15 rt 90 fd 10 lt 45 fd
10
bk 10 rt 135 fd 15 rt 90 bk 8 fd 18 rt
90
pu fd 30 pd fd 15 bk 15 rt 90 fd 18 lt
90
fd 15 bk 15 lt 90 fd 9 rt 90 fd 15 pu
fd 4
lt 90 fd 9 rt 150 pd fd 20 lt 90 fd 8
rt 60 fd 8 lt 90 fd 20
end
```

Project 3

There are many different ways in which the tree may be drawn. The method outlined below illustrates the use of the penup and pendown commands.



The plan is move from 'A' to 'R' alphabetically using penup and pendown commands wherever necessary. You can see from the drawing of the tree in the project question that each branch is ten units longer than the branch above it and all the branches are ten units apart. Follow the proposed plan and if you agree with it, try it out. You will notice that multistatement lines have been used. This will facilitate adaptation into a procedure for the next project.

```
to tree
fd 10 rt 90 fd 25 lt 90 pu fd 10 lt 90
  fd 5
pd fd 20 rt 90 fd 10 rt 90 fd 15 lt 90
  pu fd 10
  lt 90 fd 5 pd fd 10 rt 90 fd 10 rt 90
  fd 5
  lt 90 lt 90 fd 10 bk 5 rt 90 bk 10 rt
  90 bk 10
  fd 10 rt 90 fd 10 rt 90 fd 15 bk 15 lt
  90 fd 10
  rt 90 fd 20 bk 20 lt 90
  fd 10 rt 90 fd 25 bk 25 rt 90 bk 10
end
```

The first line moves the turtle from A to B to C in pendown mode and from C to D to E in penup mode.

The second line moves from E to F to G to H in pendown mode and from H to I in penup mode.

The third line moves from I to J in penup mode and from J to K to L to M in pendown mode.

The fourth line draws the uppermost branch of the tree and moves from N through L and K to O backwards.

The fifth line moves the turtle from point O to point F.

The next line goes from Q back to F. The last line moves the turtle from point F back to point A through R.

Project 4

Let's call the procedure SNOW; it's very simple.

```
to snow
  tree rt 90 tree rt 90 tree rt 90 tree
  end
```

Chapter Four

The width of the screen in terms of letter spaces will not vary. If you hold down a key and then count the spaces you will find that your Logo screen is 40 character spaces wide. If you now type 'cs' and hold down the return key you will be able to count the 25 vertical character spaces available to you on FULLSCREEN. Thus the SPLITSCREEN allows you 20 vertical locations.

Project 1

To draw a series of concentric circles simply

- a) Draw the first tiny circle
- b) Move and orientate the turtle
- c) Draw the second circle, etc.

The procedure outlined below draws five concentric circles.

```
to circle
lt 90 repeat 30 [fd 1.05 rt 12]
lt 90 pu fd 5 rt 90 pd
repeat 30 [fd 2.095 rt 12]
lt 90 pu fd 5 rt 90 d
repeat 30 [fd 3.142 rt 12]
lt 90 pu fd 5 rt 90 pd
repeat 30 [fd 4.19 rt 12]
lt 90 pu fd 5 rt 90 pd
repeat 30 [fd 5.234 rt 12]
ht
end
```

Project 2

To draw the eye you need to plan a series of coordinated arcs and semi circles. Again, there are many possible solutions and the one outlined below is only a suggestion.

```
to eye
rt 90 repeat 20 [bk 3.142 rt 360/100]
pu setpos [0 0] seth 0 pd
```

The first two lines of 'eye' draw the upper eyelid then send the turtle home. At this stage your screen will look like this:



```
rt 90 repeat 20 [bk 3.142 lt 360/100]
```

This line draws the lower lid and your screen will look like this:

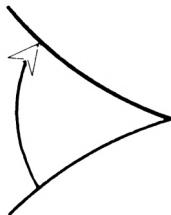


```
repeat 5 [rt 360/100 fd 3.142]
```

This command moves the turtle back along the lower lid to a position where it can start to draw the eyeball. Note that if you want follow an arc forwards that has just been drawn backwards you must reverse the order of the commands in the brackets.

```
lt 90 repeat 20 [fd 2.618 rt 360/60]c
```

This line will draw the eyeball and by now your screen will look like this:



```
pu repeat 10 [bk 2.698 lt 360/60] rt 90 fd 3  
pd repeat 10 [fd 0.8 rt 36] ht  
end
```

The last two lines will position the turtle and draw the pupil.

Project 3

To draw these linked circles you will need a basic procedure to draw the circle. Here's a suggestion:

```
to sym  
pu fd 25 rt 90 pd repeat 30 [fd 5.234  
rt 12]  
lt 90 pu bk 25  
end  
  
to oly  
sym repeat 2 [rt 90 fd 38 lt 90 sym]  
pu setpos [0 0] seth 0 bk 30 rt 90 f  
d 20 lt 90  
fd 10 sym rt 90 fd 40 lt 90 sym ht  
end
```

The procedure 'oly' will in the first line

- a) draw a circle

- b) turn to the right

- c) move forward
 - d) turn to the left
 - e) draw a circle
- Twice

in the second line it will

- a) send the turtle to its home position
- b) lift the pen
- c) reverse 30 units
- d) turn to the right
- e) move forward 20 units
- f) turn to the left

and in the third line the bottom two circles will be drawn.

Chapter Five

Making use of the chain procedure outlined in the chapter you could say:

```
to line
repeat 5 [fd 5 pu fd 5 pd]
end
```

Followed by:

```
to t
rt 60 line repeat 2 [rt 120 line]
end
```

```
to t2
rep 12 [t rt 30]
end
```

- 1) You could say:

```
cs
fd 200
lt 90
fd 320
```

or setpos [-300 200]

- 2) You might say:

```
cs
bk 120
rt 90
fd 320
```

or setpos [320 -120]

Project 1

You will see from the drawing in the project that the wings of the butterfly are made up of three circles. The best way of drawing these is to define a procedure making use of the facility to pass a parameter. You can then call up this procedure to draw the right and left wings.

```
to circle :scale
repeat 60 [fd :scale rt 6]
repeat 60 [fd :scale lt 6]
end
```

The first line of 'circle' will draw a circle to the right and the second line will draw a circle to the left.

Let's now consider a procedure to draw the complete butterfly; call it 'butter'.

```
to butter
  circle 3 circle 4 circle 5
```

The first line of 'butter' shown above will draw a series of circles to give the wings of the butterfly.

```
pu fd 30 pd rt 90 repeat 20 [fd 3.142
lt 18] lt 90 pu
```

The second line lifts the pen, moves forward 30 units, puts the pen down, turns to the right, draws the head, turns left and lifts the pen.

```
fd 20 setpos [-5 50] pd repeat 20 [fd
3 lt 6]
```

This line moves the turtle forwards 20 units and then moves it to point 155 on the x axis. It then puts the pen down and draws the first eyebrow.

```
repeat 23 [bk 3 rt 6] pu setpos [5 50]
pd
```

This line will move the turtle backwards down the antenna it has just drawn, lift the pen and move to the right ready to draw the second antenna.

```
repeat 20 [fd 3 rt 6]
```

This line starts the antenna.

```
repeat 23 [bk 3 lt 6] ht
```

and the last line finishes the drawing.

Project 2

You will notice that the insect has legs on the right of its body and others on the left. To facilitate the drawing of these two sets of legs you can define two similar procedures that you can call 'leg' and 'leg2'.

```
to leg
  fd 40 rt 30 fd 35 rt 30 fd 20
end
```

```
to leg2
fd 40 lt 30 fd 35 lt 30 fd 20
end
```

Having defined the legs let's consider a procedure to draw the whole creature; we'll call it 'insect'.

```
to insect
pu fd 50 pd rt 90 repeat 60 [fd 4 rt 6]
```

The line draws the creature's body.

```
repeat 60 [fd 1.5 lt 6] pu setpos [0 0]
seth 0
```

This line draws its head and homes the turtle.

```
fd 20 rt 45 fd 30 pd leg pu setpos [0
0] seth 0
fd 20 lt 45 fd 30 pd leg2 pu setpos [0
0] seth 0
fd 10 rt 90 fd 38 pd leg pu setpos [0
0] seth 0
fd 10 lt 90 fd 38 pd leg2 pu setpos [0
0] seth 0
rt 125 fd 30 pd leg pu setpos [0 0] se
th 0
lt 125 fd 30 pd leg2 ht
end
```

The last six lines of the procedure draw each of the six legs on the insect.

Project 3

The first thing to do is to draw the grid within which the letters are going to appear. The procedure 'plot' outlined below is only a suggestion: there are of course thousands of different possible ways of drawing this grid.

```
to plot
rt 90 fd 75 bk 150 fd 50
rt 90 fd 50 bk 150 fd 50 rt 90 fd 50
bk 150 fd 50 rt 90 fd 50 bk 150 pu set
pos [0 0] seth 0
end
```

Now for a couple of procedures to draw the 'X'es and 'O's.

```
to x
  rt 45 fd 8 bk 16
  fd 8 lt 90 fd 8 bk 16
end

to o
  pu fd 8 pd rt 90
  repeat 12 [fd 4 rt 30]
end

to win
  plot
  pu fd 25 pd o pu
  setpos [50 75] seth 0 pd x pu
  setpos [-50 75] seth 0 pd o pu
  setpos [-50 25] seth 0 pd x pu
  setpos [-50 -25] seth 0 pd x pu
  setpos [50 -25] seth 0 pd o ht
end
```

Chapter Six

Project 1

Since the boxes concerned have a similar shape you can use Logo's facility for passing a parameter and define a basic procedure. Thus we can say:

```
to box :zz
  rt 90 fd :zz rt 90 fd :zz*2 rt 90 fd :
  zz*2
  rt 90 fd :zz*2 rt 90 fd :zz
end
```

This procedure will draw a box with sides that have a length of 2 times zz. To draw a series of boxes, try this:

```
to squares
  box 12 lt 90 pu fd 12 pd box 24
  lt 90 pu fd 12 pd box 36
  lt 90 pu fd 12 pd box 48
  lt 90 pu fd 12 pd box 60
end
```

Project 2

Let's first of all define a procedure to draw a random sized box up to say 50 units square; for example:

```
to box :qq
repeat 4 [fd :qq rt 90]
end
```

Now you can say

```
to anybox
make "qq random 40
box :qq
end
```

If you now run 'anybox', a randomly sized box will be produced. Now you can say

```
to map
pu seth random 360 fd random 100 pd
anybox pu setpos [0 0]
```

Which will draw a randomly sized box at a random location. Now finally

```
to map2
repeat 20 [map]
end
```

Chapter Seven

Project 1

Here, we have the bare bones of a simple coin-tossing game where both the computer and the user toss a coin. If two heads or two tails show, this is a win for the user. Otherwise, it is a win for the computer. Several variables are used; here is a brief explanation of them:

cc is the number chosen by the computer

choice is the number chosen by the user

ss is the user's score

kk is the computer's score

```

to coin
make "cc random 3 if :cc = 0 [coin repe
at 4 [pr []]]
pr [input 1 for heads or 2 for tails]
make "choice rl
if :cc = item 1 :choice [pr [you win] m
ake "ss :ss + 1]
if :cc > item 1 :choice [pr [I win] mak
e "kk :kk + 1]
if :cc < item 1 :choice [pr [I win] mak
e "kk :kk + 1]
pr :kk pr [points to me] pr :ss pr [poi
nts to you]
wait 120 ts ct
coin
end

to toss
make "kk 0
make "ss 0
coin
end

```

Project 2

Here is a suggestion for a small procedure to ask for dimensions and print the required information.

```

to calc
repeat 4 [pr []]
pr [how wide is the room?]
make "wd rl
pr [and its length?]
make "wt rl
wait 160
pr [you need] pr [item 1 :wt] * [item 1
:wd] / 0.5 pr [titles]
end

```

Appendix 1

Loading LOGO on the 6128

Part 1

If you have not made a backup copy of your 'CP/M PLUS SYSTEM/UTILITIES' and 'DR. LOGO & HELP CP/M PLUS' discs, you must do that first. If you already have copies, then ignore the rest of Part 1 of this appendix and go on to Part 2. **NOTE:** it is not recommended that you try to load LOGO from the original discs using the method described in Part 2, as it will not work.

To backup the System/utilities disc, do the following:

- Insert the System/utilities disc in the drive, side 1 uppermost.
- Type

```
| cpm
```

and press the RETURN key once. The ' | ' symbol is obtained by holding down SHIFT and then pressing the @ key.

- Type in

```
disckit3
```

and press RETURN.

- At the bottom of the screen a 'menu' will appear. Press the f7 key to 'Copy' your discs.
- Press 'Y' if the word 'Copy' is left on the screen, else press any other letter and go back to the previous step.

- The disc drive will whirr, and soon the screen will say ‘Insert disc to WRITE’.
- Take out the System/Utilities disc and insert the new, blank, disc that you wish to copy the System/Utilities disc onto.
- Press any letter, and the computer will begin ‘writing’ onto the disc.
- Then the screen will say ‘Insert disc to READ’. Remove the current disc and re-insert the System/Utilities disc, side 1 uppermost as before.
- Press any letter key. Continue this disc swapping process until the screen says ‘Copy completed’.
- Press any letter key. Now repeat the copying process for side 2 of the System/Utilities disc, and for both sides of the Dr. Logo and Help disc. Start off by pressing the ‘Y’ key to copy another. **Note:** be careful to insert the discs with the sides you wish to copy to or from uppermost.
- When you have copied all four sides onto the four sides available on your two original blank disc, press any letter key to stop the copying process.
- Press the f0 key to stop the utilities program.

You should now have copies of your two system discs. From now on, you should not need to use them again, except to make new copies should anything go wrong with the copies you’ve just made. Make sure you write on the labels of the new discs so you know what’s on them. Also, renumber the sides as 1, 2, 3 and 4 so that they match the numbering on the original discs.

Part 2

To load Logo, do the following:

- Do a full reset of the computer – i.e. hold down SHIFT, CTRL and ESC simultaneously for a moment. The screen should then look the same as it does when the computer is first switched on.
- Insert the copy of the disc which had ‘CP/M 2.2 WITH DR. LOGO’ on it; i.e. side 4. Type in

| cpm

and press the RETURN key. **Note:** the ‘ | ’ symbol is obtained by holding down SHIFT and pressing @.

- When the disc has finished whirring and the ‘A>’ prompt appears, type

submit logo2

- Various things will happen, but after about 5 seconds, the screen will clear except for a ‘?■’ in the top left corner. Logo is now loaded and ready for you to use! Don’t forget to remove the disc from the disc drive.

Index

A

absolute commands 5–4
AND (and) 9–15
arithmetic 8–1 et seq.

B

BACK (bk) 1–4
BASIC 1–1
bf (BUTFIRST) 10–3
bk (BACK) 1–4
bl (BUTLAST) 10–3
BUTFIRST (bf) 10–3
BUTLAST (bl) 10–3
BYE (bye) 5–14

C

CAPS LOCK key 1–3
characters 3–16
circles 4–3
CLEAN (clean) 6–9
CLEARSCREEN (cs) 1–7
CLEARTEXT (ct) 8–8
colour 6–9
COPY key 3–8
CPC 6128 1–2, Appendix 1
cs (CLEARSCREEN) 1–7
ct (CLEARTEXT) 8–8
CTRL key 3–9

D

DELETE key 1–2, 3–8
dir (DIRECTORY) 5–12
direct mode 2–1
disc
 directory (dir) 5–12
 formatting 5–10
 loading 1–2
 operating system 1–2
DOT (dot) 5–2

E

ed (EDIT) 3–6
EDIT (ed) 3–6
editing procedures 3–6
end 2–2
er (ERASE) 3–17
ERASE (er) 3–17
exiting logo 5–14

F

fd (FORWARD) 1–4
fence (FENCE) 6–15
FIRST (first) 10–1
FIRST PUT (fp) 10–10
flow diagram 7–5
formatting discs 5–10
FORWARD (fd) 1–4
fp (FIRST PUT) 10–10
fs (FULLSCREEN) 1–17
FULLSCREEN (fs) 2–17

H

HIDETURTLE (ht) 1–10, 3–4
ht (HIDETURTLE) 1–10, 3–4

I

IF (if) 6–23, 7–7
ITEM (item) 7–9

K

keyboard 1–1

L

leaving logo 5–14
LEFT (lt) 1–6
LOAD (load) 5–12
lt (LEFT) 1–6

M

MAKE (make) 6–20
maths 8–1 et seq.
multistatement lines 3–15

N

nodes 6–30

O

out of space 6–29

P

parameter passing 5–16
passing parameters 5–16
pd (PENDOWN) 3–1
pe (PENERASE) 6–2
PENDOWN (pd) 3–1
PENERASE (pe) 6–2
PENUP (pu) 3–1
po (PRINT OUT) 5–9
pots (PRINT OUT TITLES) 3–17
primitive 2–2
pr (PRINT) 4–1
PRINT (pr) 4–1
PRINT OUT (po) 5–9
PRINT OUT TITLES (pots) 3–17
procedure 2–1
pu (PENUP) 3–1

R

RANDOM (random) 6–12
READ LIST (rl) 7–6
recursion 6–19
RECYCLE (recycle) 6–29
relative commands 5–4
REPEAT (repeat) 2–6
RIGHT (rt) 1–7
rl (READ LIST) 7–6
rt (RIGHT) 1–7

S

SAVE (save) 5–12
scaling procedures 5–16
se (SENTENCE) 4–3
SENTENCE (se) 4–3
seth (SETHEADING) 5–7
SETHEADING (seth) 5–7
setpc (SETPENCOLOUR) 6–9
SETPENCOLOUR (setpc) 6–9
setpos (SETPOSITION) 5–3
SETPOSITION (setpos) 5–3
setsplit (SETSPLIT) 5–9
SETSPLIT (setsplit) 5–9
SHIFT keys 1–2
SHOWTURTLE (st) 1–10, 3–4
SPLITSCREEN (ss) 2–17
ss (SPLITSCREEN) 2–17
st (SHOWTURTLE) 1–10, 3–4
STOP (stop) 6–23, 7–12
storing procedures on disc 5–12
SOUND (sound) 3–18, 6–10
system reset 1–2

T

TEXTSCREEN (ts) 2–7
tf (TURTLE FACTS) 6–17
to 2–1
ts (TEXTSCREEN) 2–7
TURTLE FACTS (tf) 6–17

V

variable 6–20

W

WAIT (wait) 7–15
WINDOW (window) 6–15
WORD (word) 10–6
WRAP (wrap) 6–15

USING DR LOGO ON THE AMSTRAD

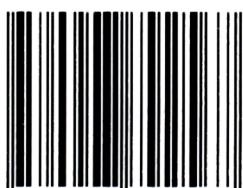
Logo, the exciting new language, allows the novice immediate access to the world of computers. Designed with the user in mind, **Logo** builds programs from simple English-like commands that are immediately understandable.

This book makes Learning **Logo** fun. By working through it, project by project, the reader will explore **Dr Logo** and soon be able to write programs using sound, colour, text and graphics. Applications are developed stage by stage and range from interactive games to random poetry generation.

Step into **Logo** and experience the exciting microworlds of sound and colour provided by **DR Logo**.

Glentop Publishers Ltd.,
Standfast House, Bath Place,
High Street, Barnet, Herts. EN5 1ED
Tel: 01-441 4130

ISBN 0-907792-56-1



9 780907 792567

GLENTOP
PUBLISHERS ■ LIMITED

£8.95 net.

THE
HARVEST

THE HARVEST LOGON THE AMERICAN STRAD

BY MARTIN SINGELTON

WITH ILLUSTRATIONS BY HENRY K. HARRIS

ILLUSTRATION BY HENRY K. HARRIS

ILLUSTRATION BY HENRY K. HARRIS

AMSTRAD CPC

MÉMOIRE ÉCRITE MEMORY ENGRAVED MEMORIA ESCRITA



<https://acpc.me/>

[FRA] Ce document a été préservé numériquement à des fins éducatives et d'études, et non commerciales.

[ENG] This document has been digitally preserved for educational and study purposes, not for commercial purposes.

[ESP] Este documento se ha conservado digitalmente con fines educativos y de estudio, no con fines comerciales.