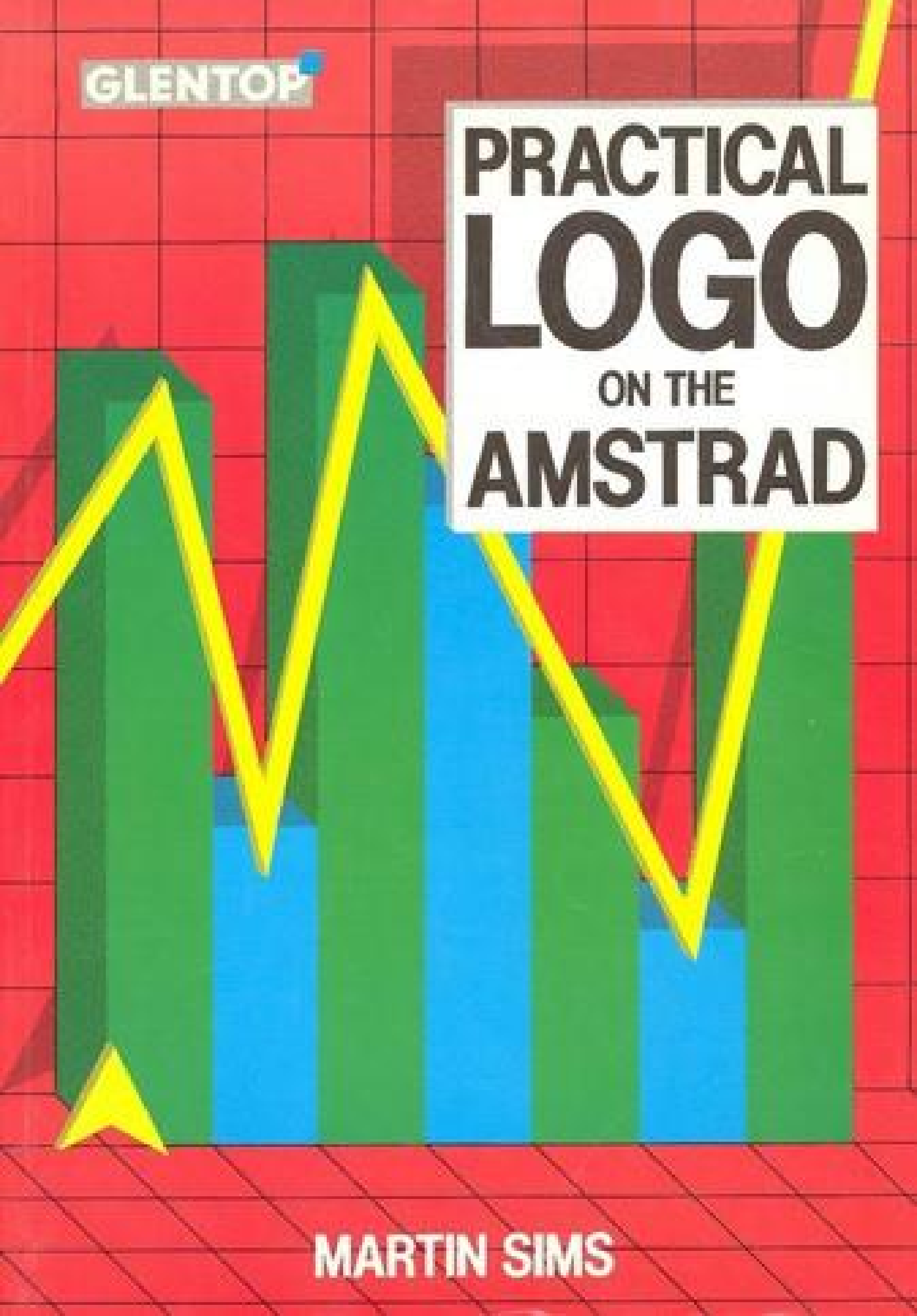

GLENTOP
PRACTICAL
LOGO
ON THE
AMSTRAD
MARTIN SIMS

Practical Logo on the Amstrad
CPC6128, PCW8256 and PCW8512

Practical Logo on the Amstrad

CPC6128, PCW8256 and PCW8512

by

M. Sims

GLENTOP
PUBLISHERS □ LIMITED

FOR WENDY, MARK AND ANDREW

MARCH 1986

ISBN 1 85181 046 3

Published by:

Glentop Publishers Ltd
Standfast House
Bath Place
High Street
Barnet
Herts EN5 5XE
Tel: 01–441–4130

Printed by The Eastern Press Ltd
London and Reading

Contents

Introduction

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 home microcomputers are sold with the BASIC language, many beginners use this to communicate with a computer for the first time. Increasingly, however, Logo has become available as an alternative to BASIC. Logo is unique in that 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.

Applied Logo is the second book in a series; the first, 'Using DR Logo on the Amstrad', adopts a more traditional approach using graphics and text computer applications that have a very wide appeal.

'Practical Logo' however makes use of more useful possible applications including decision analysis, stock control, investment analysis, simulation techniques, forecasting, production scheduling etc. It attempts to point the reader/user towards the more serious *uses* of Logo that make it a viable alternative to BASIC.

Throughout this book 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.

Chapter 1

A Brief Introduction to Logo

If you have a PCW8256 or a PCW8512, please turn to Appendix 1 to see how to load your Logo, then read on from the top of page 12. If instead you are using a CPC6128, read on!

Your DR Logo program is on side 3 of the System/utilities disc. To load Logo however, you must first insert the other disc, with side 1 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) and then release the CTRL and SHIFT keys. The computer is now set up. Now to load your Logo.

Now you must type:

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

CP/M Plus Amstrad Consumer Electronics plc

v 1.0, 61K TPA, 1 disc drive

A>

In the opposite corner of the screen will be a message saying which disc drive is being used.

Now take that disc out of the drive and place the other disc in, with side 3 uppermost. Then type:

logo3

and then press the RETURN or ENTER key. After some more clunking and a brief guest appearance by the turtle, the screen will come up with:

Welcome to

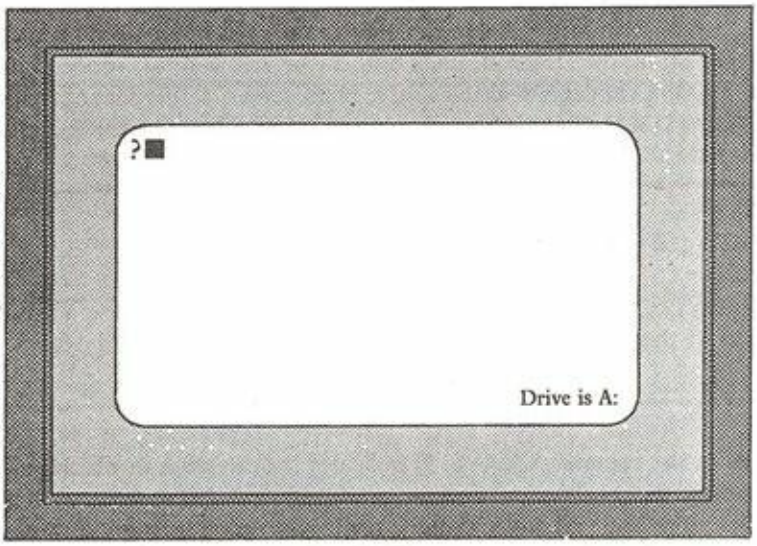
Amstrad LOGO V2.0
Copyright (c) 1985, Digital Research
Pacific Grove, California

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.



Start–Up Screen

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.

Whilst the computer is waiting for your input, it displays the Logo prompt '?' followed by the black cursor '■' where the command that you type in will appear.

The hollow arrow–head in the centre of the screen is the so–called turtle. This is the pencil point that will move around the screen at the user's command, usually drawing a line as it goes.

To make the turtle move, the user must give it a command. The basic graphics commands are:

```
FORWARD
BACK
RIGHT
LEFT
```

Try typing this command:

```
forward 1ØØ
```

Nothing has happened yet so press the RETURN key.

FORWARD (fd)

This command, which can be abbreviated fd, moves the turtle forward in the direction it is facing a distance determined by the number following the command.

Now, you can see that turtle has moved and this time drawn a line one hundred screen units long. Now try the companion command by typing:

```
back 5Ø
```
(remember to hit RETURN)

BACK (bk)

This is a command that, when followed by a space and an input moves the turtle a given number of screen units.

Notice that when either of these commands is used, the abbreviated version can be used. Before you hit RETURN, work out what will happen when you type this next command:

```
fd -3Ø
```

Now suppose you want to draw lines in some other direction: you need a command to turn the turtle. A command that does this is:

RIGHT (rt)

This command, when followed by a space and a number, will rotate the turtle in a clockwise direction the input number of degrees of rotation.

Now try these commands:

```
rt 90
rt 90
```

No more reminders to hit RETURN will be given.

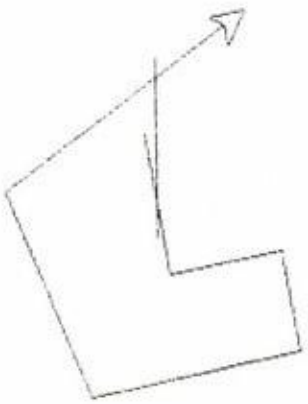
LEFT (lt)

This command, when followed by a space and a number, will rotate the turtle in an anti–clockwise direction the input number of degrees of rotation.

How about this command?:

```
rt -(-30)
```

Why not try moving the turtle around the screen using the four commands described thus far? After a while your screen might look like that shown below in Figure 1.2



Your Screen May Look Like This

FIGURE 1.2

To get rid of the mess use the command:

CLEARSCREEN (cs)

This command will ERASE the graphics screen and place the turtle in the middle of it facing 'North'.

Thus far you have been giving the turtle orders and, whenever you hit RETURN, they have been obeyed. If you wanted to, you could key in a series of commands separated by spaces and upon pressing RETURN all will be obeyed in the order they were typed in.

Try the following commands in a 'multistatement line' to draw a rectangle.

```
cs fd 100 rt 90 fd 50 rt 90 fd 100 rt 90 fd 50 rt
90
```

Now hit RETURN and your rectangle will be drawn.

By now you will have made a 'mistake' in one or two of your commands to the turtle and having typed in a 'bad' command and hit RETURN your screen will probably display an error message similar to the one shown below.

```
I don't know how to makk
```

What do you notice about this line? Well, you can see that apart from the 'CLEAR-SCREEN' command we have asked the turtle to:

```
fd 100 rt 90 fd 50 rt 90
```

Instead of writing out the above commands twice, you can, in Logo, write

```
repeat 2 [fd 100 rt 90 fd 50 rt 90]
```

REPEAT (repeat [])

This command, when followed by a space and an input, will carry out the commands contained within the instruction list, ([]), the input number of times.

A command to draw a square with sides measuring fifty screen units would therefore be:

```
repeat 4[fd 5Ø rt 9Ø]
```

Do four times — the commands in the instruction list.

What will happen with this next command?

Figure it out before typing it in and see if you were correct in your analysis.

```
repeat 3Ø[fd 13 lt 36Ø/3Ø]
```

Notice that your machine's arithmetic operations can work within Logo commands, so, for example, you could say:

```
fd 35Ø/1Ø*2
```

Note that the standard mathematical rules regarding precedence of arithmetic operators are used: i.e. in a long calculation involving × (*), ÷ (/), + and –, the multiplications and divisions will be done *first* (from left to right), *then* the additions and subtractions (also from left to right).

Well, the above line will draw a circle, or to be precise a THIRTY SIDED POLYGON.

If you wanted the execution of the circle to be a little more dramatic you could hide the turtle during the drawing operation:

HIDETURTLE (ht)

This command hides the turtle from view, which a) speeds up the drawing operation and b) makes the resultant drawing clearer.

Obviously there is a command to make the turtle reappear.

SHOWTURTLE (st)

This command will, if the turtle is hidden somewhere on the screen, make it visible.

If you wanted to, you could try:

PENUP (pu)

This command lifts the turtle's 'pen–point' and stops it from drawing during its various manoeuvres.

PENDOWN (pd)

This command puts the turtle's pen–point down for the resumption of drawing.

Now, all of this is quite interesting; what you have been doing is 'talking' to the turtle in 'DIRECT MODE' or, as it is sometimes called, 'TOP LEVEL'. However, once the computer has obeyed the command or commands, 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 commands that your computer will understand and obey. It stores each line as you type it in and obeys these commands when called upon to do so. The word 'TO' followed by a space and a procedure name followed by RETURN allows the user to create a list of instructions which in another computer language might be called a programme, but in terms of LOGO is called a 'PROCEDURE'. The last line of any procedure must be 'END'.

TO (to)

This is a special word that indicates the beginning of a procedure definition.

END (end)

This is a special word that, as the last line of all defined procedures, indicates the end of a procedure definition.

Let's define a procedure to draw the circle previously drawn. Type:

```
to circ
repeat 3Ø [fd 13 lt 36Ø/3Ø]
end
```

Notice that after pressing RETURN you receive the message:

```
circ defined
```

Now simply clear the screen and draw the required figure by typing:

```
cs circ
```

One thing to mention about procedures. There are certain names for procedures that can't be used, For example try typing:

```
to stop
```

You will receive a Logo messege telling you that 'stop is a primitive'. This means that the word STOP is one of the 'key' words in Logo like FORWARD, BACK etc. You can't use primitives as procedure names.

Consider this next procedure:

```
to pattern
repeat 12 [circ rt 3Ø]
end
```

In the above procedure a previously defined procedure has been 'nested'. Type:

```
cs pattern
```

and see what happens.

Now try this:

```
to pattern2
repeat 5 [pattern lt 36]
end
```

You have now 'nested' procedures three deep. Just imagine the power of LOGO when much larger procedures are nested FIFTEEN or TWENTY DEEP.

In order to get the full benefit from 'pattern2', it is best to give the graphics turtle a little more room to move. The easiest way to do this is to make use of the 'FULLSCREEN' command.

FULLSCREEN (fs)

This command selects a full graphics screen.

If you use FS, you won't be able to see the orders that you are giving the turtle, but you can revert to the old screen by saying:

```
SS
```

SPLITSCREEN (ss)

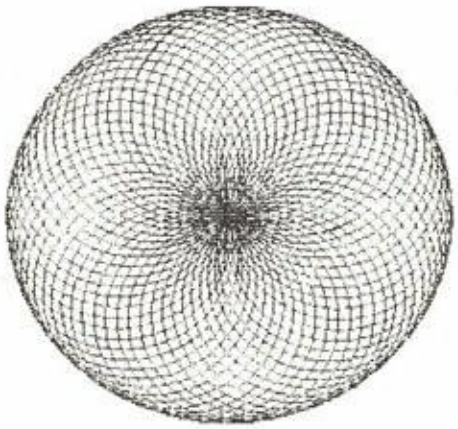
This command reserves a window for text on the graphics screen.

If you would like to control the number of lines available for text, you can use the command:

SETSPLIT (setsplit)

This command, when followed by an input number, defines the number of lines available for text in the split screen.

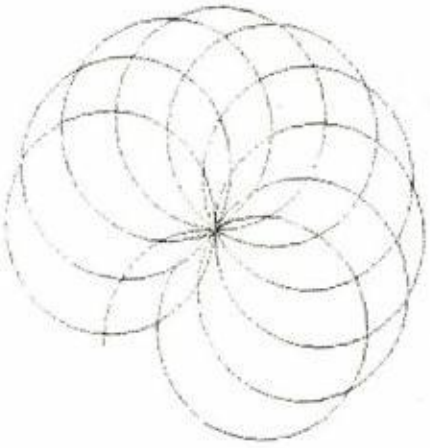
By now your screen will display the rather elegant display shown below in Figure 1.3.



PATTERN2
FIGURE 1.3

If you wanted to stop the procedure early, you could hit CTRL–G (or ALT–G or STOP on the 8256 and 8512). The screen will look something like Figure 1.4, with the message:

Stopped! in circ: fd (or something similar)



A Stopped Procedure

FIGURE 1.4

There is now an obvious need to be able to EDIT procedures. If you have an 8256 or an 8512, turn to Appendix 2.

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 square in a 'SQUARE' procedure.

In the event that this particular procedure is not in your computer, type it in as below:

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

Now type 'cs square' and run the procedure.

Having done this, your screen might look something like the one shown below in Figure 1.5.

SQUARE Procedure

FIGURE 1.5

Let's now have a look at the method provided in 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.

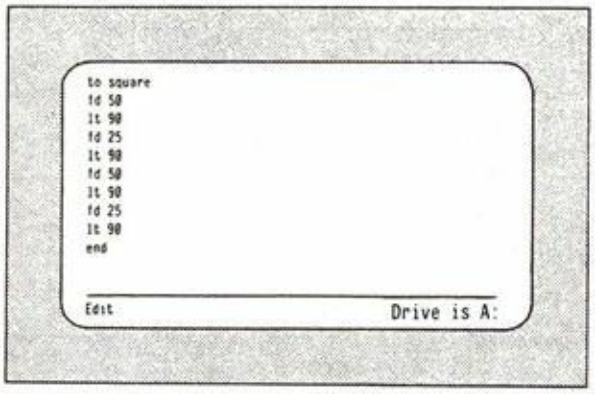
Now type 'square' and run the procedure. Having done this, next type:

```
ed "square
```

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

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 1.6



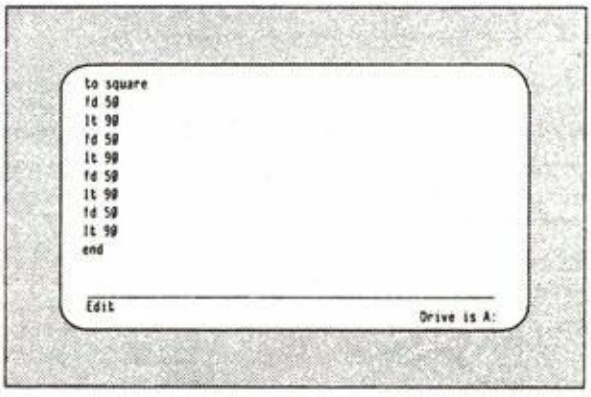
The Edit Screen

FIGURE 1.6

Let's assume that you want to change the distances in the 'fd 25' commands. The first thing to do is move the cursor to the first line to be changed. You might be able to use the arrow keys to move the cursor, in which case, use them instead of the CTRL keys described, if you prefer. Press CTRL–N three times and this will move the cursor down (don't hit RETURN 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 RETURN YET

You can now bring the cursor down to the end of the next 'fd 25'. Then move it to the left one place by hitting CTRL–B once. You can now delete the '2' and type in a '5'. The cursor is now over the '5' of the 'fd 25' command. Move the editing cursor one place to the right by hitting CTRL–F 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 1.7.



The Edited Procedure

FIGURE 1.7

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

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

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 "square
```

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 RETURN.

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 "square
```

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

```
RETURN
```

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 "square
```

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 - B moves the cursor back.

CTRL - F moves the cursor forwards.

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.

Note that on your particular model of computer, you may find that some of these control keys will only work in EDIT mode. Also, you may not be able to use the arrow keys to move the cursor, or use the COPY key to complete the edit.

Well, if you can, 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 (else CTRL–N) 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	Enables repetition of whatever is in [command list]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 – B	moves the cursor back.
CTRL – F	moves the cursor forwards.
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.

Parameters

There is one feature of Logo which deserves a mention in an introductory chapter like this: the ability to pass 'parameters'.

These are a means of allowing a procedure to perform the same sequence of actions to produce different results according to the numbers put in. For example the following procedure can draw many different rectangles:

```
to rectangle :dim1 :dim2
repeat 2 [fd :dim1 rt 90 fd :dim2 rt 90]

end
```

Procedure RECTANGLE

FIGURE 1.8

If you now type:

```
rectangle
```

you will get an error message. This is because RECTANGLE is expecting you to give it the numbers corresponding to :dim1 AND :dim2. Try typing:

```
rectangle 50 100
```

This time a rectangle will be drawn. Now try:

```
rectangle 70 120
```

A larger rectangle is drawn. The parameters, i.e. the numbers given, are used wherever the appropriate variable name appears within the procedure – in this case :dim1 OR :dim2.

As you read through this book, you might feel that some of the procedures given could be improved by the use of parameters, or that the jobs might be better done with different programs: well, there's no harm in trying! Have a go! The programs as they stand have been written so that they can be relatively easily understood; once they make sense to you, they may well benefit from any changes and improvements you can think of, and of course you can then tailor them to your own specific requirements.

Chapter 2

Part 1

Displaying Data

Having got to grips with some Logo basics, let's now consider a more serious use for Logo's abilities. Imagine a user wished to display some information visually in the form of a HISTOGRAM or as a PIE CHART.

Now since these procedures might prove useful you will need to save them on discs for future use. If you are using an 8256 or an 8512 computer, turn to Appendix 3 to find out how to format discs. If you are using a 6128, the method to use is described below.

Disc 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 reset your computer you will have to type them in again! This should not take too long and after a short while, your procedure list will look like this:

```
to circ
to pattern
to pattern2
to square
to rectangle :diml :dim2
```

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 "lotl
```

any name will do here

SAVE (save)

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

If you now type:

```
dir
```

DIRECTORY (dir)

This outputs a list of Logo file names.

The disc drive will hum and you should then see:

```
[lot1]
```

which indicates that file 'LOT1' has been saved onto the disc.

Now type

```
er "circ
er "pattern
er "pattern2
er "square
er "rectangle :diml :dim2
```

If you had wanted to empty the whole of your computer's memory you could have simply said:

```
erall
```

ERALL (erall)

This command will clear the computer's memory.

Having done this, you will be ready to re–load your procedures from disc by saying:

LOAD (load)

This reads the named file from disc and puts it into your computer's memory.

```
load "lotl
```

as you do this you will receive the message:

```
circle defined
pattern defined
pattern2 defined
square defined
rectangle defined
```

Now if you want to make a change in one of the procedures, you can use the EDIT facility as usual. So, change 'circ' to look like this:

```
to circ
repeat 2Ø [fd 1Ø rt 18]
end
```

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

```
save "lotl
```

You will receive a message saying:

```
File lotl already exists
```

Which on the face of it means that you can't alter any procedures saved on disc, unless you always save the new versions under a different name. However, there is a Logo command that will come to your rescue.

EDIT FILE (edf)

This will load a specific disc file onto the screen ready for editing.

Try typing

```
edf "lotl
```

Now wait for a few moments and the file will be loaded. Edit the procedures in whatever way you wish and when you CTRL–C or ALT–C or EXIT, the new procedures will be saved under the old 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. However, you may consider these procedures to be unworthy of the space allocated to them on disc and so you might like to erase one or all of the disc files.

ERASE FILE (erasefile)

This erases a specific disc file.

e.g.

```
erasefile "name
```

Try typing

```
erasefile "lotl
```

If you type

```
dir
```

you will see that this particular set of procedures no longer exists on your disc.

Let's suppose we wanted a procedure to display a variety of values in the form of a HISTOGRAM (bar chart). Now in order to store these values, variables have to be set. In order to set a variable the Logo primitive

```
make
```

is used.

MAKE (make)

This assigns a value or text to a given variable.

For example, type:

```
make "xx 3
```

(Don't forget the open quotes)

The quote marks in front of the the chosen variable name tells Logo to consider the work as an object rather than a procedure name.

Whenever you set a variable you must use this format.

After pressing Return type:

```
:xx
```

These dots tell us that whatever follows, followed by a space, is to be evaluated, in this case the variable XX.

You could now say:

```
make "xx :xx + 6
```

set variable xx to — the current value of xx plus 6

Now type:

```
:xx
```

and there's it's new value.

Now try:

```
make "ww martin
```

Upon pressing RETURN to you will get an error message. It's therefore obvious that we can't set a variable to a string unless we do something special.

Click the message and try this:

```
make "ww [martin]
```

Followed by:

```
:ww
```

As you can see, you can set variables to strings. We will be using this ability later in the book. Don't worry about the brackets for now.

Try setting a few variables by typing:

```
make "aa 90
make "bb 70/2
make "cc 50
make "dd 3 * 3
make "ee 100
make "ff 85
make "gg 23
make "hh 87.2
make "jj 43.75
make "kk 5 * 7
```

Now try typing:

```
fd :aa
```

or even

```
bk :cc * :bb
```

This will obviously be very useful when procedures are needed that will move the turtle any number as specified by a variable value.

Consider the procedure called PLOT shown below in Figure 2.1

```
to plot
pu lt 90 fd 50 rt 90 pd
fd 100 bk 100 rt 90 fd 150 bk 150 lt 90
fd :aa rt 90 fd 15 lt 90 bk :aa
fd :bb rt 90 fd 15 lt 90 bk :bb
fd :cc rt 90 fd 15 lt 90 bk :cc
fd :dd rt 90 fd 15 lt 90 bk :dd
fd :ee rt 90 fd 15 lt 90 bk :ee
fd :ff rt 90 fd 15 lt 90 bk :ff
fd :gg rt 90 fd 15 lt 90 bk :gg
fd :hh rt 90 fd 15 lt 90 bk :hh
fd :jj rt 90 fd 15 lt 90 bk :jj
fd :kk rt 90 fd 15 lt 90 bk :kk
end
```

Procedure PLOT

FIGURE 2.1

The first line of this procedure will move the turtle, then the second line will draw the X and Y axes within which the histogram will be plotted. Having completed this line, the turtle will have drawn the two lines shown below in Figure 2.2.

Axes

FIGURE 2.2

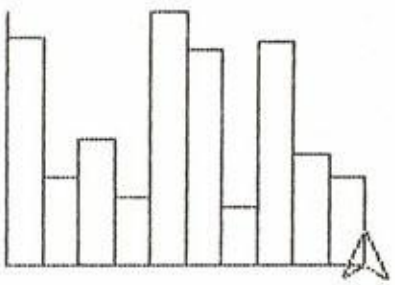
Now I've chosen 15 as the width of the 'bars' in the histogram and each of the next ten lines in the 'plot' procedure simply tells the turtle to

1) Move forward a certain number of units;
2) Turn to the right;
3) Move forward 15;
4) Turn to the left;
5) Go back a number of screen units.

Having typed in the 'plot' procedure and having previously set ten variables to different values, simply type:

```
cs plot
```

and your screen should look similar to the one shown below in Figure 2.3



A Histogram

FIGURE 2.3

Now how about a PIE CHART to visually display the same data?

How about making the chart one hundred screen units in diameter?

Well you could say:

Question: What's the circumference of a 100 unit diameter circle?

Answer: 3.142 × 100

Question: How many sides should this 'circle' have?

Answer: Let's say 30.

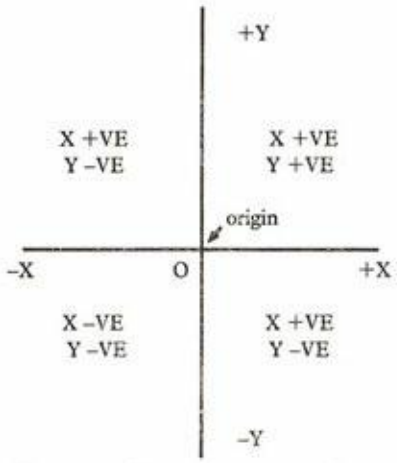
Question: What's the procedure?

Answer:

```
to circ
pu
fd 5Ø
rt 9Ø
pd
repeat 3Ø [fd (3.142 * 1ØØ) / 3Ø rt 36Ø / 3Ø]
end
```

Type it in and try it out!

As you make greater use of your Logo, you will discover that there are four sectors to the screen as shown below in Figure 2.4



Screen Sectors

FIGURE 2.4

Of one thing you can be sure, however, and that is that whenever you type 'clearscreen' the turtle will be placed at that point of the graphics area where X and Y equal zero.

Clear the screen and type:

```
circ
```

Now type:

```
clean
```

CLEAN (clean)

This Logo command will erase the graphics screen without affecting the position of the turtle.

It's also a useful feature of Logo that the turtle's position can be controlled in an absolute way as well as in a relative way which is all that has been happening thus far. To position the turtle in a specific screen location, you must make use of the command:

```
SET
```

Now there are many uses of 'SET' covering graphics, colour etc. At this point in this book we shall consider three of these uses.

```
set x
set y
setpos
```

SET X (setx 10)

This command, followed by a number, will move the turtle horizontally to the specified X position.

SET Y (sety 30)

This command, followed by a number, will move the turtle vertically to the specified Y position.

SETPOS (setpos [10 30])

This command, followed by a coordinate list of two numbers, will move the turtle directly to the specified position.

Now, if you remember, we were about to construct a PIE CHART. Now consider this procedure:

```
to pie
pu setpos [-50 -50] pd
circ
pu setpos [-50 -50]
pd fd 50 bk 50
make "total :aa + :bb + :cc + :dd + :ee
+ :ff + :gg + :hh + :jj + :kk
rt (360 / :total) * :aa fd 50 bk 50
rt (360 / :total) * :bb fd 50 bk 50
rt (360 / :total) * :cc fd 50 bk 50
rt (360 / :total) * :dd fd 50 bk 50
rt (360 / :total) * :ee fd 50 bk 50
rt (360 / :total) * :ff fd 50 bk 50
rt (360 / :total) * :gg fd 50 bk 50
rt (360 / :total) * :hh fd 50 bk 50
rt (360 / :total) * :jj fd 50 bk 50
rt (360 / :total) * :kk fd 50 bk 50
end
```

Procedure PIE

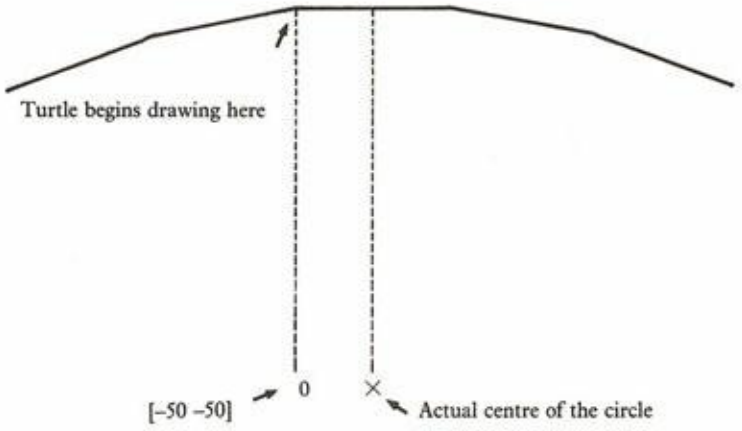
FIGURE 2.5

The first line of the procedure positions the turtle and the second line draws a circle of fifty screen units radius. Line three lifts the pen and places the turtle in the middle of the circle whilst line four puts down the pen and draws the first line. A variable called TOTAL (any name will do) adds up all the values of the various variables and the last ten lines turn the turtle through a proportion of 360 degrees equal to the variable value and then draw the other lines.

Try it! Type in

```
cs pie
```

Not very good, is it? This is because the centre of the circle is not actually at [–50 –50]. It is a little way to the right of this point because the turtle begins drawing it half a line before its top–most point. See Figure 2.6 below.



Why The Circle is Incorrect

FIGURE 2.6

This is easily fixed by adding a line after the RT 90 in CIRC:

```
bk (3.142 * 5Ø) / 3Ø
```

If I were to choose between the two presentations of data I think I prefer the histogram, however.

However, you are now in a position to change the variable values and use either the histogram or the pie–chart to visually represent the data.

Part 2

Using Random Numbers

Logo can, if you want it to, select a RANDOM value for a variable:

> **RANDOM (random)**
>
> This command will output a random whole number from zero to one less than the input number.

Before you can use this, you need to know how to get it to print its results on the dialogue screen. To do this, the command PRINT is used:

PRINT (pr)

This command causes Logo to write text onto the screen.

Try this command:

```
repeat 10 [pr random 5]
```

and you will see a selection of the integers 0 to 4 inclusive printed on screen. Or, you could try:

```
cs repeat 100 [fd random 100 home rt random 360]
```

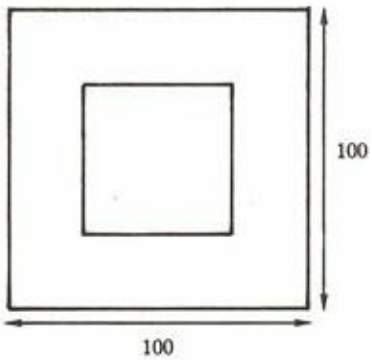
MMM! Quite an interesting 'STARBURST'.

HOME (home)

This command returns the turtle to the position [0 0] where both X and Y are zero at the centre of the screen; it also places the turtle's heading North (0 degrees or upwards).

On the more serious side, let's make use of 'RANDOM' in a technique called 'MONTE CARLO ANALYSIS' which demonstrates a statistical use of random numbers; as you know, statistical evidence is at the root of lots of business decision–making techniques.

In this simple analysis, consider the following problem.



The Simple Problem

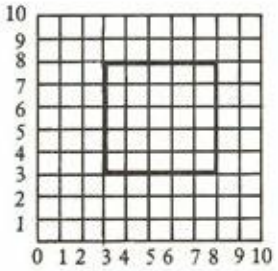
FIGURE 2.7

In Figure 2.7 above you are given a square of known dimensions: in this case it's 100 screen units square.

Now imagine a randomly sized square has been drawn in the middle of the known square and your job, without using any form of tape or rule or similar measuring device is to accurately gauge the area of the inner square.

Here's a possible solution using random numbers.

First of all divide the large square into smaller squares like the one shown below in Figure 2.8



The First Part Of The Solution

FIGURE 2.8

Now you can see the X and Y coordinates divided into 10. (You can also assess the side length and therefore area of the square but never mind).

Now pick 100 pairs of random numbers, each describing a coordinate position.

Whenever a coordinate falls within the small square make a variable 'IN' increase by 1.

Now at the end of this you will find that you have approximately 25 IN's and therefore approximately 75 outs. This means that the area of the small square is about 25% of the area of the large square. Thus it has an area of

$$\frac{\text{total area} \times \text{no of squares in}}{\text{no of trials}}$$

$$\frac{10 \times 10 \times 25}{100}$$

Which is 25 square divisions.

Before you consider a procedure to draw a randomly sized square inside a square measuring 100 x 100 units, consider a couple of Logo primitives.

TURTLEFACTS (TF)

This command outputs a list that describes the turtle's current position, its heading, pen state, pen colour and whether or not you can see the turtle.

e.g.

```
tf
[Ø Ø 297 PD 1 TRUE]
```

X position
Y position
Heading
Pen down
Pen colour
'TRUE' if turtle visible

TYPE 'TF' then move the turtle and see what difference it has made to the information in the brackets.

Now type:

```
item 2 tf
```

ITEM

This command, when followed by an input object, will output the required element of the input object.

You should receive the CURRENT Y position of the turtle.

Now try:

```
make "qq 5

item :qq tf
```

and you will be told that fifth item of TF is 1. Try:

```
item :qq "martin
```

and you will get:

i

Now you can move the turtle and, without knowing where you have moved it to, you can find its current location.

Back to the Monte Carlo Analysis:

Consider the procedure 'DRAW' shown below in Figure 2.9

```
to draw
cs pu setpos [50 50]
make "rr 20 + random 31
setx 0 + 50 - :rr pd
fd :rr rt 90 make "x item 1 tf
fd :rr * 2 rt 90 make "xx item 1 tf make
 "yy item 2 tf
fd :rr * 2 rt 90 make "y item 2 tf
fd :rr * 2 rt 90 fd :rr
pu home
end
```

Procedure DRAW

FIGURE 2.9

I shall now briefly describe the function of each line.

```
cs pu setpos [50 50]
```

This will clear the screen setting the turtle at the origin where the X coordinate=0 and the Y coordinate=0, then setting the turtle at a position where X and Y are 50.

```
make "rr 20 + random 31
```

This will pick a random number between 20 and 50 inclusive and set a variable called 'RR' to the randomly picked value.

```
setx 0 + 50 - :rr pd
```

This command will move the turtle from its current position:

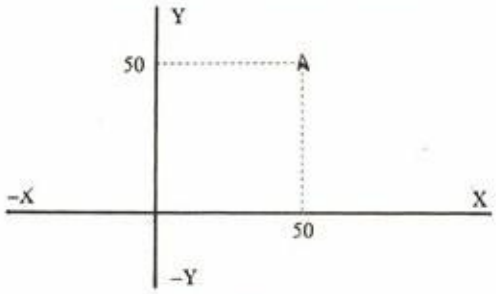


FIGURE 2.10

to a position along the X axis where X=50 – :RR. It will then put the turtle's pen down.

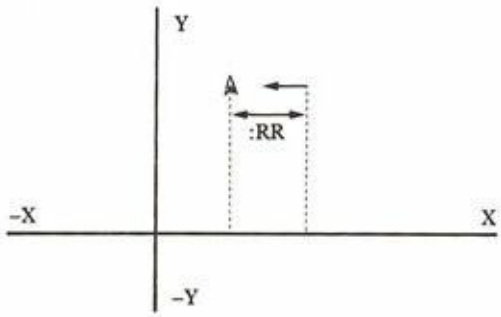


FIGURE 2.11

```
fd :rr rt 90 make "x item 1 tf
```

This will draw a line ':RR' screen units in length, turn to the right and set a variable 'X' to the current X coordinate position of the turtle.

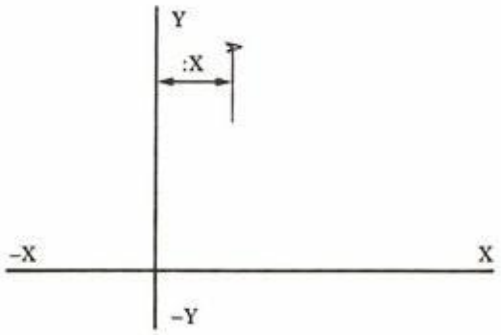


FIGURE 2.12

```
fd :rr * 2 rt 90 make "xx item 1 tf make "yy item 2 tf
```

This will draw a line twice ':RR' screen units in length, turn right and set values for the resultant position of the turtle.

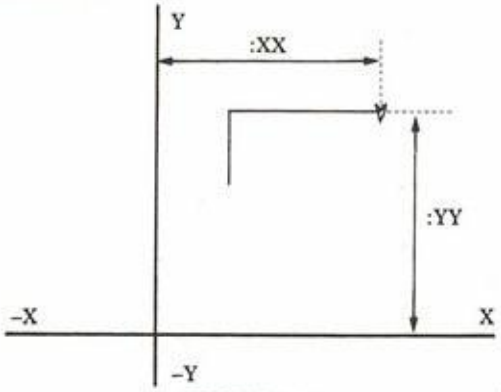


FIGURE 2.13

```
fd :rr * 2 rt 90 make "y item 2 tf
```

Again this will move forward, turn right and set a variable Y to the current Y coordinate position.

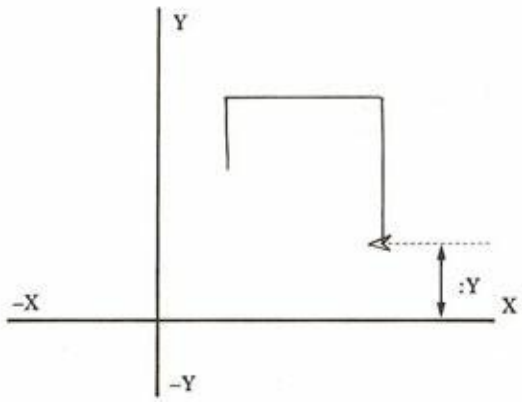


FIGURE 2.14

```
fd :rr * 2 rt 90 fd :rr
pu home
```

These last two lines will complete the randomly sized square and put the turtle in the HOME position.

As you can see below, the 'BIGBOX' procedure will draw a box 100 units square around the 50,50 position at the centre of the randomly sized square. Later on, in the final calling procedure, the variable 'LENGTH' is set to 100. For the time being, set it to 100 by saying:

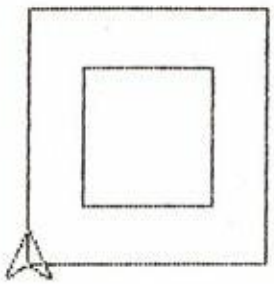
```
make "length 100
```

```
to bigbox
pu setpos [0 0] pd
repeat 4 [fd :length rt 90]
pu
end
```

Procedure BIGBOX

FIGURE 2.15

If you now call 'DRAW' and 'BIGBOX' you will have a graphics area something like the one shown below in Figure 2.16, except that the inner box may be a different size.



The Effect of DRAW and BIGBOX

FIGURE 2.16

Now we must devise a procedure

a) to randomly locate the turtle;
b) find out where it is and increment a variable if it is inside the randomly sized square.

Here are suggestions called:

```
locate
test1
test2
```

to do these two jobs.

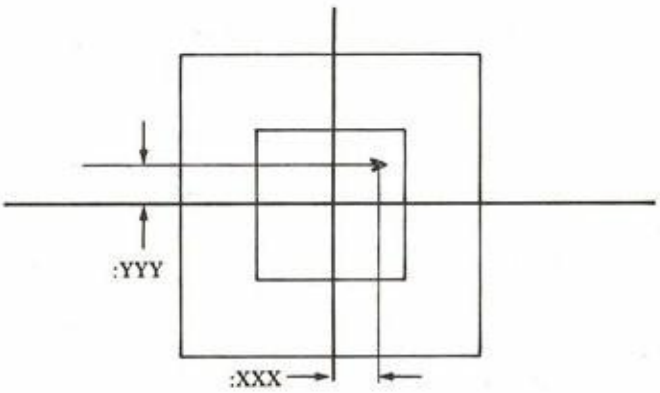
```
to locate
pu
home setx random :length + 1 sety random
 :length + 1
test1
end
to test1
make "xxx item 1 tf
make "yyy item 2 tf
if and (:xxx > :x) (:xxx < :xx) [test2]
end
to test2
if and (:yyy < :yy) (:yyy > :y) [make "i
n :in + 1]
end
```

The Procedures for Jobs a) and b)

FIGURE 2.17

Now 'LOCATE' is quite straightforward. With the turtle in the lower left-hand corner of the square (as a result of the 'HOME' command), it is then randomly re-located within the big box. LOCATE will then go to the first test procedure called TEST1.

Now the first two lines of TEST1 will set variables 'XXX' and 'YYY' to the current X and Y coordinates of the turtle as shown below:



The Operation of TEST1

FIGURE 2.18

The third line of TEST1 makes use of the conditional primitives:

```
if
```

and

```
and
```

IF (if)

This will carry out an instruction list if its inputs are TRUE

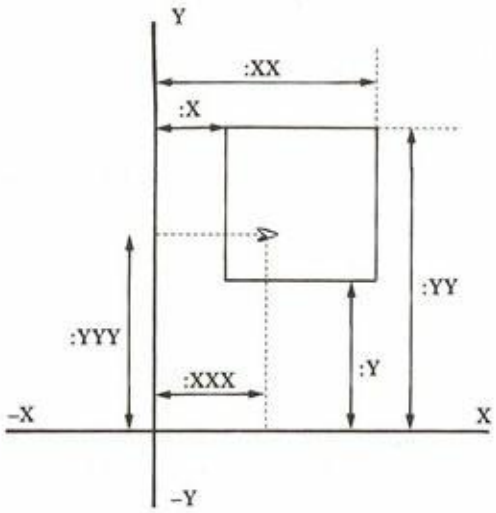
e.g.

```
if 6 > 3 [print [yes]]
yes
```

AND (and)

This will output TRUE if all input expressions are true. Otherwise it will output FALSE.

If you look at line three of TEST1 you will see that it tests a couple of conditions:



The Conditions in TEST1

FIGURE 2.19

Now the condition asks:

```
IF AND (:XXX > :X) (:XXX < :XX) [TEST2]
       ~~~~~~~~~~ ~~~~~~~~~~~~
         TRUE         TRUE
```

When both conditions are true, as they are in the above case, we can move onto the second test, TEST2.

```
if and (:yyy < :yy) (:yyy > :y) [make "i
n :in + 1]
```

So if all 4 conditions have been answered with TRUE, we increment a variable called 'IN' by one.

If we repeat 'LOCATE' and, because they are tested within 'LOCATE', TEST1 and TEST2, one hundred times, the variable 'IN' will be incremented to a value that in proportion to 100 will represent the area of the random square compared with the area of the known square.

Now to carry out the necessary calculations with procedure called CALCAREA:

```
to calcarea
make "area (((:length * :length) / :tria
ls) * :in)
make "actualarea :rr * :rr * 4
pr (se "estimated "area :area "sq. "unit
s)
pr (se "actual "area :actualarea "sq. "u
nits)
end
```

Procedure CALCAREA

FIGURE 2.20

Now you know that the area of the large box is 10,000 sq. units (100 x 100) and that the number of IN's will represent the relative proportion of the area of the randomly drawn box.

i.e.

$$\frac{\text{LENGTH} \times \text{LENGH} \times \text{dots in random square}}{\text{no. of TRIALS}}$$

$$\frac{10000}{100} \times :\text{IN}$$

or, as in our first example:

$$\frac{10 \times 10}{100} \times 25$$

That gives the estimated area.

You know of course that the real length of the side of the randomly drawn square is twice ':RR' and the real area twice ':RR' times twice ':RR' so you can compare the two with a final message in the last two lines of the CALC procedures.

Lastly here's the final super procedure:

```
to monte
cs
make "in 0 make "length 100 make "trials
 100
pd draw
bigbox
repeat :trials [locate]
calcarea
end
```

Monte-Carlo Procedure

FIGURE 2.21

Notice that within MONTE, the variables 'LENGTH' and 'TRIALS' have been set to 100. If you wanted to, you could of course set these to different values, or even have RANDOM values for them. Later on, you will learn how to set these values by means of an input whilst the program is running (if you don't happen to know already, of course). Meanwhile, why not try different numbers and see what happens? Try to find the number of trials needed to give a reasonable chance of getting the area correct to within 5%.

```
?
?monte
estimated area 9200 sq. units
actual area 9604 sq. units
?
?monte
estimated area 2000 sq. units
actual area 2916 sq. units
?
?monte
estimated area 2900 sq. units
actual area 2116 sq. units
?
?monte
estimated area 3300 sq. units
actual area 3600 sq. units
?
```

A Few Sample Runs of MONTE

FIGURE 2.22

Chapter 3

Decision Analysis

Imagine you have a management decision to make, a decision that a great many people have to make at least once or twice in their lives. It's a decision that is made all the more difficult because some of the factors that affect the decision–making process are non–quantifiable.

Let's outline the problem: it centres around the choice of a new house from two alternatives.

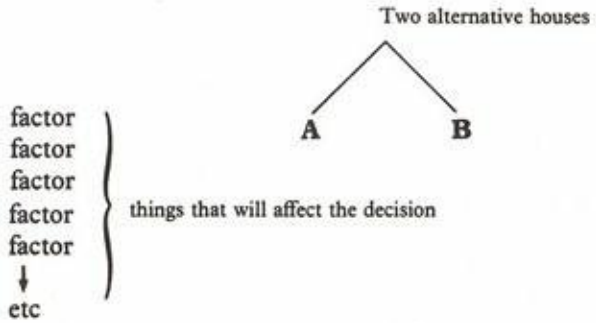
Now quite obviously there will be quantifiable, objective 'facts' regarding the two properties that will affect the decision. However, there are also non–quantifiable subjective factors that impinge upon the decision–making process, making coming to a decision more difficult.

Now wouldn't it be nice to be able to develop a small computer procedure to assist in the making of decisions that involve both subjective and objective criteria?

The model chosen for us in this area is that of:

DECISION
ANALYSIS

Suppose you are trying to choose between two houses, A and B, and that there are a number of factors affecting your choice. If this is the case your 'problem' will look like this:



Setting Out the Problem

FIGURE 3.1

Now to move the problem closer to a real world situation by elaborating upon some of the 'factors' shown above. Let's assume the list might look like the one shown below where some of the considerations that may affect our decision have been listed.

Factor	A	B
Cost price	14000	12000
Rates/year	400	300
Desirability of area		
Availability of schools		
Availability of shops		
Availability of sports		
Number of rooms		
Size of garden		
Distance from station		
Attractiveness of house		

Some Factors Filled In

FIGURE 3.2

As you can see from the above list, some of the factors are measurable in terms of cost and the relevant information has been put in the table. Given this objective information, house B books to be the best house to move to. However, the other considerations are equally important and their effect may alter the decision to move to house 'B'.

In order to carry out a computation and arrive at a decision, these subjective considerations should be rated on a scale of 1 to 100. The lower the rating, the lower the 'cost': a high number will indicate a high 'cost' (i.e. low desirability) rating. For example, both houses might be fairly close to a school but house A might be half a mile closer to the school than house B. In this case A might attract a rating of 50 as opposed to B's 60.

Imagine now that, in conjunction with whoever else involved on this particular management decision, you have arrived at the following ratings for the subjective criteria.

Factor	A	B
Cost price	14	12
Rates/year	0.4	0.3
Desirability of area	10	15
Availability of school	50	60
Availability of shops	80	40
Availability of sports	40	40
Number of rooms	30	30
Size of garden	10	30
Distance from station	15	25
Attractiveness of house	15	20

Ratings From 1 to 100

FIGURE 3.3

Now you must WEIGHT all the criteria. This weighting is carried out on a 1 to 10 scale with a low number indicating least important and high number being an indication of great importance. Of course any weighting can appear any number of times. With the agreed weighting added, your table might look this this:

Factor	A	B	Weighting
Cost price	14	12	8
Rates/year	0.4	0.3	5
Desirability of area	10	15	8
Availability of school	50	60	4
Availability of shops	80	40	1
Availability of sports	40	40	3
Number of rooms	30	30	9
Size of garden	10	30	10
Distance from station	15	25	6
Attractiveness of house	15	20	3
	Weightings		

FIGURE 3.4

Now the merit of house A will be given by:

$$(14)^8 \times (0.4)^5 \times (10)^8 \times (50)^4 \times (80)^1 \times (40)^3 \times (30)^9 \times (10)^{10} \times (15)^6 \times (15)^3$$

whilst the merit of house B will be given by:–

$$(12)^8 \times (0.3)^5 \times (15)^8 \times (60)^4 \times (40)^1 \times (40)^3 \times (30)^9 \times (30)^{10} \times (25)^6 \times (20)^3$$

Now in itself the 'merit number' is meaningless, but it is useful to be able to compare the two numbers. What you now have to say is:

$$\frac{\text{merit A number}}{\text{merit B number}}$$

If the result of this calculation is less than one, house A is the best choice and if the resultant number is greater than one, house B is the obvious choice.

Let's now devise a procedure that will:

1) accept factors from the user;
2) accept costs, RATES and WEIGHTS for the factors;
3) display the input information;
4) calculate the relative merits and recommend a decision.

Suppose we want to display the information in the form shown below:

Factor	A	B	Weighting
A	AA	AAA	AAAA
B	BB	BBB	BBBB
C	CC	CCC	CCCC
D	DD	DDD	DDDD
E	EE	EEE	EEEE
F	FF	FFF	FFFF
G	GG	GGG	GGGG
H	HH	HHH	HHHH
J	JJ	JJJ	JJJJ

The Form Of Display Required

FIGURE 3.5

Now the first thing you could do is devise a procedure to get some information from the user about the factors that affect the decision. You could for example make use of the READLIST command:

READLIST (rl)

Upon receiving this command, Logo will wait for an input from the keyboard before continuing

Just try typing:

```
rl
```

and nothing will happen until there is an input, followed by RETURN, from the user. That input will then be displayed. It's easy therefore to set a variable to any input from the user by saying:

```
make "qq item 1 rl
```

This sets the variable 'QQ' to the first item input by the user.

Note also the use of:

```
print
```

Just to remind you, here it is again:

PRINT (pr)

This Logo command will display the input objects on the text screen.

e.g.

```
pr "rat
rat
pr (rat roy girl)
rat roy girl
```

Thus you could say:

```
to info
ts ts pr [WHEN ASKED FOR FACTORS]
pr []
pr [TYPE IN YOUR TEN FACTORS]
pr []
pr [WITH A SPACE BETWEEN EACH ONE]
pr []
pr [THEN PRESS RETURN]
pr []
pr [NOW TYPE YOUR FACTORS] make "qq rl
make "a item 1 :qq
make "b item 2 :qq
make "c item 3 :qq
make "d item 4 :qq
make "e item 5 :qq
make "f item 6 :qq
make "g item 7 :qq
make "h item 8 :qq
make "j item 9 :qq
make "k item 10 :qq
end
```

Procedure INFO

FIGURE 3.6

Clear the text by typing:

```
ct
```

Now type:

```
info
```

And your screen will look like the one shown below in Figure 3.7

```
WHEN ASKED FOR FACTORS

TYPE IN YOUR TEN FACTORS

WITH A SPACE BETWEEN EACH ONE

THEN PRESS RETURN

NOW TYPE YOUR FACTORS
```

INFO's Display

FIGURE 3.7

May I suggest that you respond by using the caps lock key and typing:

```
PRICE RATES AREA. SCH'L SHOPS SPORT
ROOMS GAR'N TRAIN HOUSE
```

Having done that, press RETURN.

We now need a procedure to display the information, but first, a procedure to set all the variables, a, aa, aaa, aaaa etc. to zero:

```
to data
make "a 0 make "aa 0 make "aaa 0 make "a
aaa 0
make "b 0 make "bb 0 make "bbb 0 make "b
bbb 0
make "c 0 make "cc 0 make "ccc 0 make "c
ccc 0
make "d 0 make "dd 0 make "ddd 0 make "d
ddd 0
make "e 0 make "ee 0 make "eee 0 make "e
eee 0
make "f 0 make "ff 0 make "fff 0 make "f
fff 0
make "g 0 make "gg 0 make "ggg 0 make "g
ggg 0
make "h 0 make "hh 0 make "hhh 0 make "h
hhh 0
make "j 0 make "jj 0 make "jjj 0 make "j
jjj 0
make "k 0 make "kk 0 make "kkk 0 make "k
kkk 0
end
```

Procedure DATA

FIGURE 3.8

If you type in the above procedure and then type:

```
data
```

You will have set 40 variables to zero.

Now let's consider the procedure to display the variable values. In order to do this, you will need to utilise the Logo primitive SENTENCE.

SENTENCE (se)

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

```
to display
pr (se "factor " " " " " " " " " " " "a " "
" " " " " " "b " " " " " " "weight)
pr (se :a " " " " " " " " " " :aa " " " "
" " " :aaa " " " " " " " " " :aaaa)
pr (se :b " " " " " " " " " " :bb " " " "
" " " :bbb " " " " " " " " " :bbbb)
pr (se :c " " " " " " " " " " :cc " " " "
" " " :ccc " " " " " " " " " :cccc)
pr (se :d " " " " " " " " " " :dd " " " "
" " " :ddd " " " " " " " " " :dddd)
pr (se :e " " " " " " " " " " :ee " " " "
" " " :eee " " " " " " " " " :eeee)
pr (se :f " " " " " " " " " " :ff " " " "
" " " :fff " " " " " " " " " :ffff)
pr (se :g " " " " " " " " " " :gg " " " "
" " " :ggg " " " " " " " " " :gggg)
pr (se :h " " " " " " " " " " :hh " " " "
" " " :hhh " " " " " " " " " :hhhh)
pr (se :j " " " " " " " " " " :jj " " " "
" " " :jjj " " " " " " " " " :jjjj)
pr (se :k " " " " " " " " " " :kk " " " "
" " " :kkk " " " " " " " " " :kkkk)
end
```

Procedure DISPLAY

FIGURE 3.9

If you now type:

```
data info
```

Then respond correctly to INFO. Then type:

```
display
```

your screen should look like the one shown below in Figure 3.10

```
?display
FACTOR          A          B          WEIGHT
PRICE           0          0          0
RATES           0          0          0
AREA.           0          0          0
SCH'L           0          0          0
SHOPS           0          0          0
SPORT           0          0          0
ROOMS           0          0          0
GAR'N           0          0          0
TRAIN           0          0          0
HOUSE           0          0          0
```

DISPLAY's Display

FIGURE 3.10

Now for a procedure to collect the necessary information from the user in terms of costs, ratings and weightings for each of the ten criteria.

Let's call the procedure ASK and make it get some information from the user.

```
to ask
ct
pr [NOW GIVE ME SOME INFO']
pr []
pr [TYPE IN THE FIGURES FOR COST / RATIN
G]
pr [FOR THE FACTORS]
pr []
pr [WHEN THE FACTORS APPEAR ON THE SCREE
N]
pr []
pr [PRESS RETURN EACH TIME]
pr (se :a "for "A) make "aa item 1 rl
pr (se :b "for "A) make "bb item 1 rl
pr (se :c "for "A) make "cc item 1 rl
```

```
pr (se :d "for "A) make "dd item 1 rl
pr (se :e "for "A) make "ee item 1 rl
pr (se :f "for "A) make "ff item 1 rl
pr (se :g "for "A) make "gg item 1 rl
pr (se :h "for "A) make "hh item 1 rl
pr (se :j "for "A) make "jj item 1 rl
pr (se :k "for "A) make "kk item 1 rl
```

First 19 Lines of ASK

FIGURE 3.11

You can see that the procedure structure is quite simple. First of all it will print some messages on the dialogue screen after it has been cleared, and then there are a series of lines that will:

1) print the factor for which the information applies;
2) set the variable to the input from the user.

For example:

```
pr (se :c "for "a) make "cc item 1 rl
```

This will print on the screen

```
area for a
```

To which the user will respond with an input of

```
1Ø      <RETURN>
```

The second part of the line will set the variable CC equal to 10.

You can see that the procedure will display each factor in turn and await an input from the user in terms of its cost or RATING.

You must of course do the same for house B: here's a suggestion for the remainder of the ASK procedure:

```
ct pr [NOW FOR B]
pr (se :a "for "B) make "aaa item 1 rl
pr (se :b "for "B) make "bbb item 1 rl
pr (se :c "for "B) make "ccc item 1 rl
pr (se :d "for "B) make "ddd item 1 rl
pr (se :e "for "B) make "eee item 1 rl
pr (se :f "for "B) make "fff item 1 rl
pr (se :g "for "B) make "ggg item 1 rl
pr (se :h "for "B) make "hhh item 1 rl
pr (se :j "for "B) make "jjj item 1 rl
pr (se :k "for "B) make "kkk item 1 rl
end
```

A Suggestion For The Rest Of ASK

FIGURE 3.12

Again, each line is carrying out a similar task.

Of course the computer must store the various weightings for each of the factors and so, again, here's a suggested procedure to collect the necessary information.

```
to weight
ct
pr [NOW INPUT THE WEIGHTINGS AS ASKED]
pr (se :a "weight "please) make "aaaa it
em 1 rl
pr (se :b "weight "please) make "bbbb it
em 1 rl
pr (se :c "weight "please) make "cccc it
em 1 rl
pr (se :d "weight "please) make "dddd it
em 1 rl
pr (se :e "weight "please) make "eeee it
em 1 rl
pr (se :f "weight "please) make "ffff it
em 1 rl
pr (se :g "weight "please) make "gggg it
em 1 rl
pr (se :h "weight "please) make "hhhh it
em 1 rl
pr (se :j "weight "please) make "jjjj it
em 1 rl
pr (se :k "weight "please) make "kkkk it
em 1 rl
end
```

Procedure WEIGHT

FIGURE 3.13

Now having run 'INFO', run 'ASK' and 'WEIGHT', run 'DISPLAY' and your screen will look like the one shown in Figure 3.14.

```
1
SPORT weight please
3
ROOMS weight please
9
GAR'N weight please
10
TRAIN weight please
6
HOUSE weight please
3
?display
FACTOR          A          B       WEIGHT
PRICE          14         12            8
RATES         0.4        0.3            5
AREA.          10         15            8
SCH'L          50         60            4
SHOPS          80         40            1
SPORT          40         40            3
ROOMS          30         30            9
GAR'N          10         30           10
TRAIN          15         25            6
HOUSE          15         20            3
?
```

The Display After INFO, ASK, WEIGHT & DISPLAY

FIGURE 3.14

Now for a procedure to carry out the necessary calculations.

Remember that to calculate the merit of house A, the maths will look like this:

```
:AA ^ :AAAA × :BB ^ :BBBB × :CC ^ :CCCC  etc.
```

Rating — To the power of — Weight

So the rating of the first criteria is set to the power of it's weight and this is multiplied by the rating of the second criteria which has been set to the power of it's weighting...and so on.

Now we have a small problem in that ↑ or ^ or 'to the power of' is not available in Logo.

For example, in another language, you might be able to say:

```
PR 2 ↑ 2
```

and get the answer 4 (which is 2 to the power of 2 or 2 squared)

However you will get the answer:

"I don't know how to ↑ 2"

This means that a way around this problem must be found.

Consider the procedure 'POWER' shown below:

```
to power :a2 :p
make "c1 :a2
repeat :p - 1 [make "a2 :a2 * :c1]
make "a1 :a2
end
```

Procedure POWER

FIGURE 3.15

The 'make "c1 :a2' line is there because you cannot pass values out of a procedure using the parameters: they vanish as soon as you jump out, so another variable must be used. Now try this:

```
power 2 6
```

and you will receive the answer of 64 which is 2 to the power of 6.

Now if you look at 'CALC1' you will see that in each line POWER has been used.

```
to calc1
power :aa :aaaa make "ma :a1
power :bb :bbbb make "mb :a1
power :cc :cccc make "mc :a1
power :dd :dddd make "md :a1
power :ee :eeee make "me :a1
power :ff :ffff make "mf :a1
power :gg :gggg make "mg :a1
power :hh :hhhh make "mh :a1
power :jj :jjjj make "mj :a1
power :kk :kkkk make "mk :a1
make "merit.a :ma * :mb * :mc * :md * :m
d * :me * :mf * :mg * :mh * :mj * :mk
end
```

Procedure CALC1

FIGURE 3.16

In the last line of CALC1 all the 'powered up' values are mutliplied together to give a variable 'MERIT.A'.

Exactly the same procedure is followed in 'CALC2' resulting in a variable value 'MERIT.B'.

```
to calc2
power :aaa :aaaa make "ma :a1
power :bbb :bbbb make "mb :a1
power :ccc :cccc make "mc :a1
power :ddd :dddd make "md :a1
power :eee :eeee make "me :a1
power :fff :ffff make "mf :a1
power :ggg :gggg make "mg :a1
power :hhh :hhhh make "mh :a1
power :jjj :jjjj make "mj :a1
power :kkk :kkkk make "mk :a1
make "merit.b :ma * :mb * :mc * :md * :m
d * :me * :mf * :mg * :mh * :mj * :mk
end
```

Procedure CALC2

FIGURE 3.17

The last part of the calculations are carried out by the 'DECIDE' procedure shown below.

```
to decide
make "best :merit.a / :merit.b
if :best > 1 [pr (se "choose "b)]
if :best < 1 [pr (se "choose "a)]
if :best = 1 [pr [I don't know!]]
end
```

Procedure DECIDE

FIGURE 3.18

Lastly you can of course embody everything in a procedure called 'CHOOSE' outlined below. TRY IT OUT.

```
to choose
data
display
info
ask
display
weight
display
calc1
calc2
decide
end
?copyoff
```

Procedure CHOOSE

FIGURE 3.19

If there are fewer than ten factors you can simply put in the number 1 during ASK and WEIGHT and the answer will not be adversely affected.

Chapter 4

Sales Forecasting

Just imagine you are the sales manager of a company which buys and sells a quarterly magazine, say on train spotting.

Also assume that to have customers that can' be supplied means that there is a loss to a competitor and that therefore stocks of magazines left over can't be sold at a later date either. Now consider the following conversation between the managing director and the sales manager of the company:

MD "What on Earth's happening? You've been with us for over three years now and you still can't get your sales forecasts right. What have you to say for yourself?"
SM "Well I try but I'm very rarely successful."
MD "You're *never* successful! Just look at these figures. What method do you use, GUESSWORK?"
SM "Well, no, I use the figures from the previous sales period."
MD "It's not good enough! Every period we either lose sales through being understocked or we throw money away by being overstocked. Now get your house in order or you're fired!"

Well, as you wander back to your office you might be thinking to yourself:

WHAT CAN I DO?

The answer of course is to analyse and change your method of forecasting so that it fits both present and hopefully future patterns of sales.

The first thing to do is to consider some past data and, as luck would have it, you have kept a record since the start of your employment with the company. (Something your predecessor didn't do). Here are the facts:

Well, when you look at your records it looks as though no thought at all has gone into the forecasts. No wonder the MD is irate. Just look at those errors:

error = actual sales minus forecast

PERIOD	SALES	FORECAST	ERROR
1	100	0	0
2	180	100	–80
3	150	180	30
4	130	150	20
5	210	130	–80
6	160	210	50
7	150	160	10
8	240	150	–90
9	200	240	40
10	180	200	20

Forecasting Errors

FIGURE 4.1

Now you might say to yourself:

"I think I'll get a forecasting system computerised."

"I think I'll design a system that can be used to make forecasts for anything from sales to manufacturing scrap rates, and from absenteeism rates to soap consumption."

Having decided that, you might like to start programming:

First of all let's assume you want to manipulate say 40 variables: that's PERIOD, SALES, FORECAST, and ERROR multiplied by 10 periods. You will remember from Chapter 3 that a fairly easy way to set variables was by setting them all to zero in an initial procedure.

Here's a suggestion called 'MARKET'

```
to market
make "a 0 make "aa 0 make "aaa 0 make "a
aaa 0
make "b 0 make "bb 0 make "bbb 0 make "b
bbb 0
make "c 0 make "cc 0 make "ccc 0 make "c
ccc 0
make "d 0 make "dd 0 make "ddd 0 make "d
ddd 0
make "e 0 make "ee 0 make "eee 0 make "e
eee 0
make "f 0 make "ff 0 make "fff 0 make "f
fff 0
```

```
make "g 0 make "gg 0 make "ggg 0 make "g
ggg 0
make "h 0 make "hh 0 make "hhh 0 make "h
hhh 0
make "j 0 make "jj 0 make "jjj 0 make "j
jjj 0
make "k 0 make "kk 0 make "kkk 0 make "k
kkk 0
end
```

Procedure MARKET

FIGURE 4.2

You can now arrange to display those values at specific screen locations in a procedure called 'DISPLAY'

```
to display
ct
pr (se " "period " "sales " " " "f.cast
" " " "error)
pr (se " " :a " " " " " " " :aa " " " " " "
:aaa " " " " " " " :aaaa)
pr (se " " :b " " " " " " " :bb " " " " " "
:bbb " " " " " " " :bbbb)
pr (se " " :c " " " " " " " :cc " " " " " "
:ccc " " " " " " " :cccc)
pr (se " " :d " " " " " " " :dd " " " " " "
:ddd " " " " " " " :dddd)
pr (se " " :e " " " " " " " :ee " " " " " "
:eee " " " " " " " :eeee)
pr (se " " :f " " " " " " " :ff " " " " " "
:fff " " " " " " " :ffff)
pr (se " " :g " " " " " " " :gg " " " " " "
:ggg " " " " " " " :gggg)
pr (se " " :h " " " " " " " :hh " " " " " "
:hhh " " " " " " " :hhhh)
pr (se " " :j " " " " " " " :jj " " " " " "
:jjj " " " " " " " :jjjj)
pr (se " " :k " " " " " " " :kk " " " " " "
 :kkk " " " " " " " :kkkk)
end
```

Procedure DISPLAY

FIGURE 4.3

The first line makes use of the command:

CLEARTEXT (ct)

This command erases all the text from the LOGO DIALOGUE area and positions the text cursor in the upper left-hand corner.

The rest of the procedure simply prints a series of sentences made up of words and spaces or variable values and spaces.

If you now call up:

```
market
```

followed by:

```
display
```

Your screen will look similar to the one shown below in Figure 4.4.

```
?market
?display
 period   sales      f.cast      error
   0        0          0          0
   0        0          0          0
   0        0          0          0
   0        0          0          0
   0        0          0          0
   0        0          0          0
   0        0          0          0
   0        0          0          0
   0        0          0          0
   0        0          0          0
```

Initial Display

FIGURE 4.4

Now for a procedure that will invite a response from the user. Let's also make our procedure 'menu driven', which means that the user will have various choices of action open to him/her. I've chosen to call the suggested procedure 'MENU'.

```
to menu
display
pr [PLEASE TYPE 1 IF YOU WISH TO PUT]
pr [IN A SALES FIGURE] pr []
pr [OR TYPE 2 IF YOU REQUIRE A FORECAST]
 pr []
pr [OR 3 FOR ERRORS TO BE CALCULATED] pr
 []
pr [OR 4 IF YOU WISH TO EXIT THE PROGRAM
]
make "choice item 1 rl
if :choice = 1 [ask]
if :choice = 2 [forecast]
if :choice = 3 [errors]
if :choice = 4 [pr [OK] display]
end
```

Procedure MENU

FIGURE 4.5

The first line simply calls the DISPLAY procedure while the second, third, fourth and fifth lines print a series of messages.

The sixth line waits for an input from the user. This input will be the number 1, 2, 3 or 4 according to the user's requirements.

In lines seven, eight, nine and ten the input is considered and an appropriate procedure called up.

Let's assume the user wants to input some information and so presses '1'. At this, the program will move to the procedure called 'ASK'.

```
to ask
pr [WHICH SALES PERIOD?]
pr [TYPE -100 TO DISCONTINUE]
make "which item 1 rl
if :which = -100 [stop]
look1
pr [PUT IN SALES FIGURE FOR PERIOD]
make "input item 1 rl
look2
end
```

Procedure ASK

FIGURE 4.6

The first two lines of 'ASK' ask for some information and give the user a chance to discontinue the whole program if required by the use of

STOP (stop)

This will stop the execution of the current procedure and will return to the '?' prompt or to the calling procedure.

In line five, having previously made sure that user does not wish to exit the procedure, the input is analysed in a procedure called 'LOOK1'.

```
to look1
if :which = 1 [make "a 1]
if :which = 2 [make "b 2]
if :which = 3 [make "c 3]
if :which = 4 [make "d 4]
if :which = 5 [make "e 5]
if :which = 6 [make "f 6]
if :which = 7 [make "g 7]
if :which = 8 [make "h 8]
if :which = 9 [make "j 9]
if :which = 10 [make "k 10]
end
```

Procedure LOOK1

FIGURE 4.7

As you ca see, all that happens in 'LOOK1' is that an instruction list (the commands in the square brackets) is carried out according to the input by the user. Thus, if in answer to the message:

'WHICH SALES PERIOD?'

the user says:

6

the variable F will change from zero to six.

Continuing with the 'ASK' procedure, another message will now appear asking for the sales value associated with the previously input number. After setting the input variable called :INPUT, a procedure called 'LOOK2' is called up.

```
to look2
if :which = 1 [make "aa :input]
if :which = 2 [make "bb :input]
if :which = 3 [make "cc :input]
if :which = 4 [make "dd :input]
if :which = 5 [make "ee :input]
if :which = 6 [make "ff :input]
if :which = 7 [make "gg :input]
if :which = 8 [make "hh :input]
if :which = 9 [make "jj :input]
if :which = 10 [make "kk :input]
erasefile "sales
save "sales
end
```

Procedure LOOK2

FIGURE 4.8

This procedure will look at the current value of WHICH and change a variable to the current INPUT value.

Now for the last two lines of LOOK2. You found out in Chapter 2 that, assuming you have a formatted disc in your drive, if you type:

```
save "sales
```

the whole of your computer's memory of variables and procedures will be saved onto the disc under the filename 'SALES'. See that it works by trying it now.

Now if you were to type:

```
erasefile "sales
```

All this information would be lost from the disc (don't try that now: file 'SALES' will be needed later).

Now in our LOOK2 procedure the old information contained in our current 'SALES file is erased and the whole of our computer's current memory, including the latest inputs, is saved in the newly created sales file.

So far your procedures will be as follows:

MARKET	Sets all variables to zero
DISPLAY	Displays all variable values
MENU	Calls DISPLAY then asks the user what to do. If the choice is '1' the procedure 'ASK' is called.
ASK	This procedure asks for information from the user regarding PERIOD number and SALES figures.
LOOK1	Sets the period numbers.
LOOK2	Sets variables to various sales figures; the procedure then wipes out all the old information and saves all the latest.

Let's now use our procedures and check that the system for the input of sales information works.

Try typing the following (if you already have a file called 'SALES' on the disc, don't bother with the first instruction):

```
save "sales
market
menu

Wish to put in sales figure → 1
period number ————————→ 1
sales for period ————————→ 100 ←——— Disc drive hums assuming you've
                                    something called sales already
                                    on the disc for it to erase.
MENU
1
10
180 ←——— HUM from drive
MENU
```

Screen After Some Data Entry

FIGURE 4.9

Now put in the rest of the relevant period numbers and associated sales figures shown below in Figure 4.10.

period	sales	f.cast	error
1	100	0	0
2	180	0	0
3	150	0	0
4	130	0	0
5	210	0	0
6	160	0	0
7	150	0	0
8	240	0	0
9	200	0	0
10	180	0	0

Sales Data

FIGURE 4.10

Well, thus far you haven't done much to impress your managing director, so let's suppose you might at any time wish to make a forecast based upon the DATA in the system. (i.e. that saved in your sales file).

Type:

```
erall
```

to erase any procedures and variable values from the workspace.

Now type:

```
load "sales
```

You will see all the procedures re-defined. Variables will also be restored.

Now just type:

```
display
```

to make sure.

Let's now assume you want to make a forecast. Well, you would type:

```
menu
```

Followed by:

'2' But don't do that because you havn't yet defined the 'FORECAST' procedure. Take a look at a proposed procedure below:

```
to forecast
ct
display
pr []
pr [WHICH METHOD WOULD YOU LIKE TO EMPLO
Y?] pr []
pr [TYPE 1 FOR LAST SALES PERIOD METHOD]
 pr []
pr [OR TYPE 2 FOR THE SIMPLE AVERAGE MET
HOD] pr []
pr [OR 3 TO USE THE MOVING AVERAGE SYSTE
M]
make "wish item 1 rl
if :wish = 1 [forecast1]
if :wish = 2 [forecast2]
if :wish = 3 [forecast3]
end
```

Procedure FORECAST

FIGURE 4.11

Now being a clever sales manager you have done some research and discovered that you could start making forecasts in three ways:

1) by taking the last period sales;
2) by using a simple average;
3) by using a moving average.

There are others but you have chosen to start with these.

Now you know that your managing director is none too pleased with your efforts with the 'last period sales' method, but, with a little adjustment at a later date, it might be of some use so let's consider the 'FORECAST1 procedure.

```
to forecast1
make "bbb :aa
make "ccc :bb
make "ddd :cc
make "eee :dd
make "fff :ee
make "ggg :ff
make "hhh :gg
make "jjj :hh
make "kkk :jj
end
```

Procedure FORECAST1

FIGURE 4.12

As you can see this simply rearranges the variable values so that each of the variables BBB, CCC etc. is made equal to the SALES for the previous period.

Now run:

```
menu
```

and answer: 2
followed by: 1 to the question 'WHICH METHOD?'

Try typing:

```
display
```

and there will be all the forecasts using the 'LAST PERIOD SALES' method displayed on your screen.

Now for the 'SIMPLE AVERAGE' method of forecasting. The forecaster will take all the sales figures for the previous periods to date, add then all up and divide by the number of periods.

```
to forecast2
pr [WHICH PERIOD FORECAST?]
make "wish2 item 1 rl
if :wish2 = 2 [make "bbb :aa]
if :wish2 = 3 [make "ccc round (:aa + :b
b) / 2]
if :wish2 = 4 [make "ddd round (:aa + :b
b + :cc) / 3]
if :wish2 = 5 [make "eee round (:aa + :b
b + :cc + :dd) / 4]
if :wish2 = 6 [make "fff round (:aa + :b
b + :cc + :dd + ee) / 5]
if :wish2 = 7 [make "ggg round (:aa + :b
b + :cc + :dd + :ee + :ff) / 6]
if :wish2 = 8 [make "hhh round (:aa + :b
b + :cc + :dd + :ee + :ff + :gg) / 7]
if :wish2 = 9 [make "jjj round (:aa + :b
b + :cc + :dd + :ee + :ff + :gg + :hh)
/ 8]
if :wish2 = 10 [make "kkk round (:aa + :
bb + :cc + :dd + :ee + :ff + :gg + :hh
+ :jj) / 9]
end
```

Procedure FORECAST2

FIGURE 4.13

So with the above procedure, you can see that a question is asked of the user and if, for example, the user inputs the number '5', which means a forecast for period 5 is required, the computer will look down the list of conditions and when it comes to 5 it will say:

$$\text{Make the variable "EEE=}\frac{(:AA + :BB + :CC + :DD)}{4}$$

In this procedure use of 'ROUND' has been made.

ROUND (round)

This will output the input number to the nearest whole number.

This is because we don't want forecasts in 'parts of products'.

Now type:

```
menu
2
2
5
menu
```

You will see your 140 forecast for period five. As it happens, this is a long way from the actual sales for that period of 210!

Now continue to call up 'MENU'; answer the questions and get forecasts for each period two to ten inclusive using the simple average method. Notice that as a forecast for period two the period one sales have been used.

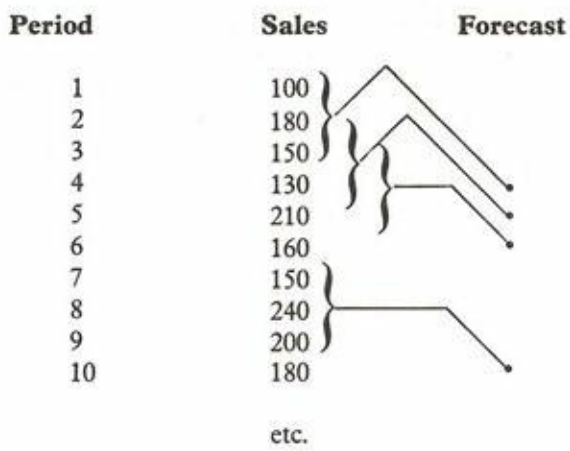
By the end of this your screen will look like the one shown below in Figure 4.14; you can press 4 to continue your procedure development.

period	sales	f.cast	error
1	100	0	0
2	180	100	0
3	150	140	0
4	130	143	0
5	210	140	0
6	160	154	0
7	150	155	0
8	240	154	0
9	200	165	0
10	180	169	0

Simple Average Forecasts

FIGURE 4.14

You can see from this that there are still some fairly large errors, so why not consider another method? Let's try a method that's somewhere between the 'last period sales' and the 'simple average'. Consider for example a 'moving average' method.



Moving Averages

FIGURE 4.15

Now the method: the forecaster chooses how many periods to use, in the example three have been chosen, then the sales for each of the periods are added together and divided by the number of periods. This will then give a 'moving average'.

For example, a forecast for period 7 will be:

$$\frac{\text{sales for period 6} + \text{sales for period 5} + \text{sales for period 4}}{3}$$

```
to forecast3
pr [WHICH PERIOD FORECAST?]
pr []
make "aaa 0 make "aaaa 0
make "bbb 0 make "bbbb 0
make "ccc 0 make "cccc 0
make "wish item 1 rl
if :wish = 4 [fore]
if :wish = 5 [fore]
if :wish = 6 [fore]
if :wish = 7 [fore]
if :wish = 8 [fore]
if :wish = 9 [fore]
if :wish = 10 [fore]
end
```

Procedure FORECAST3

FIGURE 4.16

If you look at 'FORECAST3' you will see that is starts off with the usual question. The forecasting and error variables for the first three periods are then set to zero. This is because the first period for which a forecast can be made is the fourth one and you won't want any previous forecasts or errors cluttering up your screen if you are using the moving average method. The forecaster, having answered the question with the sales period for which a forecast is required, is moved to the procedure 'FORE' as shown below.

```
to fore
if :wish = 4 [make "ddd round (:aa + :bb
 + :cc) / 3]
if :wish = 5 [make "eee round (:bb + :cc
 + :dd) / 3]
if :wish = 6 [make "fff round (:cc + :dd
 + :ee) / 3]
if :wish = 7 [make "ggg round (:dd + :ee
 + :ff) / 3]
if :wish = 8 [make "hhh round (:ee + :ff
 + :gg) / 3]
if :wish = 9 [make "jjj round (:ff + :gg
 + :hh) / 3]
if :wish = 10 [make "kkk round (:gg + :h
h + :jj) / 3]
end
```

Procedure FORE

FIGURE 4.17

This is quite a straightforward averaging job and needs no explanation.

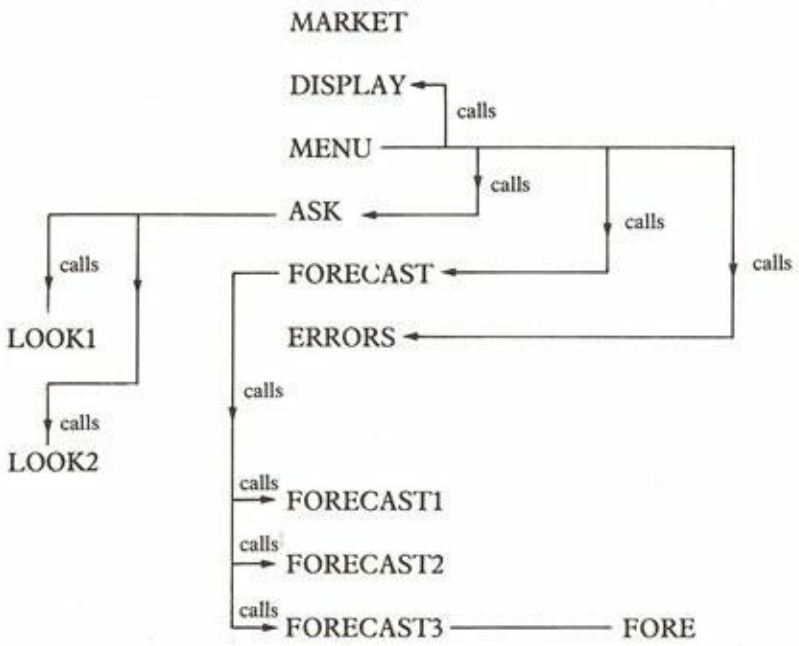
Call up 'MENU' and answer all the questions using the third forecasting method and eventually your screen will look like this:

```
period   sales     f.cast     error
 1        100       0         0
 2        180       0         0
 3        150       0         0
 4        130       143         0
 5        210       153         0
 6        160       163         0
 7        150       167         0
 8        240       173         0
 9        200       183         0
 10       180       197         0
```

Moving Average Forecasts

FIGURE 4.18

Thus far your 'procedure plan' looks like this:



Program Flow So Far

FIGURE 4.19

The 'ERRORS' procedure is now needed. This is quite straightforward. As you can see from the suggested procedure, the error is only calculated if both a sale has been made (even if it's zero) and a forecast has been made for that period.

```
to errors
if and (:aa > -1) (:aaa > 0) [make "aaaa
 :aaa - :aa]
if and (:bb > -1) (:bbb > 0) [make "bbbb
 :bbb - :bb]
if and (:cc > -1) (:ccc > 0) [make "cccc
 :ccc - :cc]
if and (:dd > -1) (:ddd > 0) [make "dddd
 :ddd - :dd]
if and (:ee > -1) (:eee > 0) [make "eeee
 :eee - :ee]
```

```
if and (:ff > -1) (:fff > 0) [make "ffff
 :fff - :ff]
if and (:gg > -1) (:ggg > 0) [make "gggg
 :ggg - :gg]
if and (:hh > -1) (:hhh > 0) [make "hhhh
 :hhh - :hh]
if and (:jj > -1) (:jjj > 0) [make "jjjj
 :jjj - :jj]
if and (:kk > -1) (:kkk > 0) [make "kkkk
 :kkk - :kk]
end
```

Procedure ERRORS

FIGURE 4.20

Now type:

```
errors
display
```

There are your three period moving average forecasts and their associated positive or negative errors. If you add up these errors (ignoring the signs) you will find that the total error for the last seven periods was 191; the cumulative error for the same seven periods using the last period sales method was 310. This technique appears to reduce forecast errors in his particular case by over a third.

period	sales	f.cast	error
1	100	0	0
2	180	0	0
3	150	0	0
4	130	143	13
5	210	153	-57
6	160	163	3
7	150	167	17
8	240	173	-67
9	200	183	-17
10	180	197	17

Errors

FIGURE 4.21

Now how about some visual method of representing the data? Here's a suggestion to show the sales figures for successive periods. Notice that it's based on the procedure outlined in Chapter 2 which drew a histogram.

```
to plot
cs pu setpos [-80 -80] pd
fd 100 bk 100 rt 90 fd 150 bk 150 lt 90
fd :aa rt 90 fd 15 lt 90 bk :aa
fd :bb rt 90 fd 15 lt 90 bk :bb
fd :cc rt 90 fd 15 lt 90 bk :cc
fd :dd rt 90 fd 15 lt 90 bk :dd
fd :ee rt 90 fd 15 lt 90 bk :ee
fd :ff rt 90 fd 15 lt 90 bk :ff
fd :gg rt 90 fd 15 lt 90 bk :gg
fd :hh rt 90 fd 15 lt 90 bk :hh
fd :jj rt 90 fd 15 lt 90 bk :jj
fd :kk rt 90 fd 15 lt 90 bk :kk
end
```

Procedure PLOT

FIGURE 4.22

Having seen that 'PLOT' works quite well here's a suggestion called 'PLOT.ERROR' which does just that.

```
to plot.error
cs pu setx -20 pd
seth 0 fd 100 bk 100 rt 90 fd 150 bk 150
 lt 90 rt 90 fd 15 lt 90
fd :bbbb rt 90 fd 15 lt 90 bk :bbbb
fd :cccc rt 90 fd 15 lt 90 bk :cccc
fd :dddd rt 90 fd 15 lt 90 bk :dddd
fd :eeee rt 90 fd 15 lt 90 bk :eeee
fd :ffff rt 90 fd 15 lt 90 bk :ffff
fd :gggg rt 90 fd 15 lt 90 bk :gggg
fd :hhhh rt 90 fd 15 lt 90 bk :hhhh
fd :jjjj rt 90 fd 15 lt 90 bk :jjjj
fd :kkkk rt 90 fd 15 lt 90 bk :kkkk
end
```

Procedure PLOT.ERROR

FIGURE 4.23

There, the errors from your three period moving average forecasting system have been drawn.

Now did you notice anything about your Sales Figures? Well there appears to be a 'SEASONAL' fluctuation. Let's see if this is the case — because if your magazine selling business is subject to seasonal demand then it would make sense to 'SEASONALLY ADJUST' any sales forecasts you make. Consider the following:

$$\text{AVERAGE PERIOD SALES FOR 1983} = \frac{100 + 180 + 150}{3} = 143$$

$$\text{AVERAGE PERIOD SALES FOR 1984} = \frac{130 + 210 + 160}{3} = 167$$

$$\text{AVERAGE PERIOD SALES FOR 1985} = \frac{150 + 240 + 200}{3} = 197$$

Now let's consider each period within each year as a percentage of the year's average sales per period.

Period	Year average	1983 143	1984 167	1985 197	Average
First		70%	78%	76%	75%
Second		126%	126%	122%	125%
Third		104%	96%	101%	100%

Now what we have done is to take each period's sales for each year as a percentage of the year's average sales per period. From this analysis it appears that on average the company sells 75% of the average sales per period in the first period.

In the second period of each year sales leap to 125% of the average for the year and in the third period sales are about average for the year at 100%.

This being the case, you as sales manager can quite rightly say "when I make a forecast for the first period of any year, I will make the forecast and then reduce that forecast by 25%. If, however, the forecast is for the second period in a year I will increase it by 25%. Third period sales forecasts I will not adjust."

All of this makes a procedure to seasonally adjust our forecasts a fairly straightforward matter. Consider the procedure 'ADJUST' shown below. However, before you type in the ADJUST procedure, try typing:

NODES (nodes)

This command tells you how much memory there is left for Logo to work with.

You will see probably only a couple of hundred or so nodes left – not much.

This being the case, it is suggested that you store the 'ADJUST' and 'SMOOTH' procedures which follow on a separate disc file and call them up when you feel you need them, having first of all made enough room in your Logo's memory by erasing unwanted procedures.

```
to adjust
make "bbb round ((:bbb * 125 / 100) + (:
bbb * 5 / 100))
make "ccc round (:ccc + (:ccc * 5 / 100)
)
make "ddd round ((:ddd * 75 / 100) + (:d
dd * 5 / 100))
make "eee round ((:eee * 125 / 100) + (:
eee * 5 / 100))
make "fff round (:fff + (:fff * 5 / 100)
)
make "ggg round ((:ggg * 75 / 100) + (:g
gg * 5 / 100))
make "hhh round ((:hhh * 125 / 100) + (:
hhh * 5 / 100))
make "jjj round ((:jjj + (:jjj * 5 / 100
))
make "kkk round ((:kkk * 75 / 100) + (:k
kk * 5 / 100))
end
```

Procedure ADJUST

FIGURE 4.24

Notice that as well as making the necessary adjustments in terms of seasonal variations, a growth rate of 5% per period has been included in adjusted figure. Remember also that this procedure can only be run once for each set of data unless some conditions are placed on the adjustments being made. If you want to put in a sales figure, followed by an adjusted forecast, a version of adjust that will only make an adjusted forecast after a period sales figure has been put in by the user is required (remember you can change ADJUST in the LOGO EDITOR screen by typing 'ED "ADJUST"):

```
to adjust
if :which = 1 [make "bbb round ((:bbb *
125 / 100) + (:bbb * 5 / 100))]
if :which = 2 [make "ccc round (:ccc + (
:ccc * 5 / 100))]
if :which = 3 [make "ddd round ((:ddd *
75 / 100) + (:ddd * 5 / 100))]
```

```
if :which = 4 [make "eee round ((:eee *
125 / 100) + (:eee * 5 / 100))]
if :which = 5 [make "fff round (:fff + (
:fff * 5 / 100))]
if :which = 6 [make "ggg round ((:ggg *
75 / 100) + (:ggg * 5 / 100))]
if :which = 7 [make "hhh round ((:hhh *
125 / 100) + (:hhh * 5 / 100))]
if :which = 8 [make "jjj round ((:jjj +
(:jjj * 5 / 100))]
if :which = 9 [make "kkk round ((:kkk *
75 / 100) + (:kkk * 5 / 100))]
end
```

Procedure ADJUST Adjusted

FIGURE 4.25

You will see that 'ADJUST' will now only alter the forecast for period 3 if the current value of the variable :WHICH is 2. i.e. if the user has just put in the latest sales figure and wants an adjusted forecast for the next period.

If you now run through the routine of calling 'MARKET' to reset all the variables to zero and then entering the periods and associated sales figures, then asking for forecasts and then adjusted forecasts you can begin to see how a quite sophisticated package might be designed to carry out sales forecasts in a variety of ways and display results visually. Shown below in Figure 4.26 is the result of ADJUSTed moving average forecasts; you can see that the errors are now quite small.

period	sales	f.cast	error
1	100	0	0
2	180	0	0
3	150	0	0
4	130	114	-16
5	210	199	-11
6	160	171	11
7	150	134	-16
8	240	225	-15
9	200	192	-8
10	180	158	-22

Seasonally Adjusted Moving Average Forecasts

FIGURE 4.26

The choice of packaging these various forecasting methods will be up to you as sales manager. You may wish to have separate super-procedures for different methods, or go in for more sophisticated file handling methods, or you may even come to use a forecasting method that does not rely on input data. The only information the current system needs is:

1) the last sales figures;
2) the forecast for that period.

The only other element needed is what known as a 'smoothing constant'. This constant falls between 0 and 1 and in a lot of cases has a value of about 0.2.

The technique is called 'exponential smoothing' and involves the following method:

New forecast for next period = Forecast for the last period + K × Last sales figure minus forecast

The smoothing constant

Here's a suggestion using a factor of 0.4, although the choice is yours.

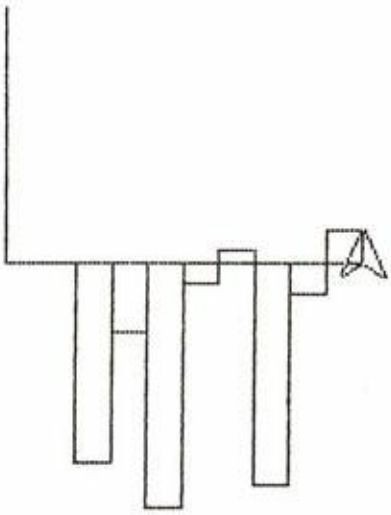
```
to smooth
make "ccc round (:bbb + (0.4 * (:bb - :b
bb)))
make "ddd round (:ccc + (0.4 * (:cc - :c
cc)))
make "eee round (:ddd + (0.4 * (:dd - :d
dd)))
make "fff round (:eee + (0.4 * (:ee - :e
ee)))
make "ggg round (:fff + (0.4 * (:ff - :g
gg)))
make "hhh round (:ggg + (0.4 * (:gg - :g
gg
make "jjj round (:hhh + (0.4 * (:hh - :h
hh)))
make "kkk round (:jjj + (0.4 * (:jj - :j
jj)))
end
```

Procedure SMOOTH

FIGURE 4.27

When you run this procedure and plot the errors however, you will see that the predictions are actually made worse, as shown below in Figure 4.28.

period	sales	f.cast	error
1	100	0	0
2	180	0	0
3	150	72	-78
4	130	103	-27
5	210	114	-96
6	160	152	-8
7	150	155	5
8	240	153	-87
9	200	188	-12
10	180	193	13



Exponentially Smoothed Errors

FIGURE 4.28

Clearly, the value chosen for the fiddle–factor — sorry, 'smoothing constant' — must be made carefully!

Chapter 5

Investment Analysis

Ever had trouble deciding where and how to invest the large sums of money that you have as liquid assets in your bank account? Do you have problems in choosing between investment opportunities on offer to you? Well, if this is the case here are a couple of suggested methods for assessing how best to invest the odd few thousand pounds or so.

Imagine the scene, you with anything up to £50,000 to invest and three opportunities confronting you.

What should you do?

Well, there are a lot of very clever people whose job it is to assess investment on your behalf using some very sophisticated methods and models. On the other hand you might, as a *first* step, like to use one of the methods outlined in this chapter.

The three investment proposals are shown below.

Year	A	B	C
0	(25000)	(50000)	(15000)
1	8000	5000	10000
2	8000	15000	20000
3	8000	32000	20000
4	8000	30000	0
5	8000	20000	0
6	8000	3000	0
7	8000	2000	0
8	8000	0	0
9	8000	0	0
10	8000	0	0

All figures are in pounds and the figures in brackets represent the initial investment, i.e. money to be invested in year 0. The other figures represent cash flow returns as a result of the investment at the end of each year indicated. Note that the projects have different lifespans and that zero is entered if there is no cash inflow that year.

As a first step in computing the information you will need to develop a procedure to gather the information about the alternative investments, which you can call A, B and C. Take a look at these suggestions:

```
to ask
pr [HELLO,I WANT SOME INFORMATION!]
pr [] pr [FIRST OF ALL,TYPE IN (A first,
B second,C third)]
pr [THE INVESTMENT RETURN DURATIONS]
pr [] pr [WITH A SPACE BETWEEN EACH]
pr [] pr [THEN PRESS RETURN]
make "input rl
make "zz item 1 :input make "zzz item 2
:input make "zzzz item 3 :input pr []
pr [NOW THE PROJECT COSTS IN THE SAME WA
Y]
make "input rl make "b 0
make "bb item 1 :input make "bbb item 2
:input make "bbbb item 3 :input
pr [NOW THE INVESTMENT RETURNS]
repeat 3 [pr []]
pr [FIRST YEAR 1] make "input rl make "c
 1
make "cc item 1 :input make "ccc item 2
:input make "cccc item 3 :input
ct pr [YEAR 2] make "input rl make "d 2
make "dd item 1 :input make "ddd item 2
:input make "dddd item 3 :input
ct pr [YEAR 3] make "input rl make "e 3
make "ee item 1 :input make "eee item 2
:input make "eeee item 3 :input
ct pr [YEAR 4] make "input rl make "f 4
make "ff item 1 :input make "fff item 2
:input make "ffff item 3 :input
ct pr [YEAR 5] make "input rl make "g 5
make "gg item 1 :input make "ggg item 2
:input make "gggg item 3 :input
ct pr [YEAR 6] make "input rl make "h 6
ask2
end
```

Procedure ASK

FIGURE 5.1a

```
to ask2
make "hh item 1 :input make "hhh item 2
:input make "hhhh item 3 :input
ct pr [YEAR 7] make "input rl make "j 7
make "jj item 1 :input make "jjj item 2
:input make "jjjj item 3 :input
ct pr []
ct pr [YEAR 8] make "input rl make "k 8
make "kk item 1 :input make "kkk item 2
:input make "kkkk item 3 :input
ct pr [YEAR 9] make "input rl make "l 9
make "ll item 1 :input make "lll item 2
:input make "llll item 3 :input
ct pr [NOW THE LAST YEAR] make "input rl
 make "m 10
make "mm item 1 :input make "mmm item 2
:input make "mmmm item 3 :input
erasefile "data save "data
end
```

Procedure ASK2

FIGURE 5.1b

The second line of procedure 'ASK' requires the input:

10 7 3 <RETURN>

Next the various items of this input are set to variables, thus in the fifth line of the procedure, ZZ is set to 10, ZZZ to 7 and ZZZ to 3.

The initial investment and the various cash flows are then input by the user remembering to input a zero if there isn't any cash flow in that year from a particular project. The procedure has been split into two, ASK and ASK2 because of the limits to procedure length in Logo.

Now for a procedure to display the information.

```
to display
pr (se " " " " " " " " " " " " :zz " " " " "
 " " :zzz " " " " " " " " " " " " :zzzz)
pr (se "YEAR " " " " " "A " " " " " " " "
" "B " " " " " " " " " " " " "C)
pr (se " " :b " " " " " " " :bb " " " " :b
bb " " " " " " " " :bbbb)
pr (se " " :c " " " " " " " :cc " " " " "
:ccc " " " " " " " " :cccc)
pr (se " " :d " " " " " " " :dd " " " " "
:ddd " " " " " " " " :dddd)
```

```
pr (se " " :e " " " " " " " :ee " " " " " "
:eee " " " " " " " " :eeee)
pr (se " " :f " " " " " " " :ff " " " " " "
:fff " " " " " " " " :ffff)
pr (se " " :g " " " " " " " :gg " " " " " "
:ggg " " " " " " " " :gggg)
pr (se " " :h " " " " " " " :hh " " " " " "
:hhh " " " " " " " " :hhhh)
pr (se " " :j " " " " " " " :jj " " " " " "
:jjj " " " " " " " " :jjjj)
pr (se " " :k " " " " " " " :kk " " " " " "
:kkk " " " " " " " " :kkkk)
pr (se " " :l " " " " " " " :ll " " " " " "
:lll " " " " " " " " :llll)
pr (se " " :m " " " " " " " :mm " " " " " "
 :mmm " " " " " " " " :mmmm)
end
```

Procedure DISPLAY

FIGURE 5.2

Now RUN 'ASK' making the appropriate responses.

Having done that call 'DISPLAY' and your screen should look similar to the one shown below in Figure 5.3. If you are unhunhappy with the display then edit 'DISPLAY'.

```
          10          7                   3
YEAR      A           B                   C
  0       25000       50000               15000
  1       8000        5000                10000
  2       8000        15000               20000
  3       8000        32000               20000
  4       8000        30000               0
  5       8000        20000               0
  6       8000        3000            0
  7       8000        2000            0
  8       8000        0           0
  9       8000        0           0
  10      8000        0           0
```

DISPLAY's Display

FIGURE 5.3

Now save the 'DISPLAY' procedure under the filename 'dis1' and the 'ASK' and 'ASK2' procedures under the filename 'ask1'.

Now the first of the analytical tools involves asking the question:

'How Long will it be before the investment has paid for itself?'

In order to calculate the answer to that question, consider the procedure 'PAYBACKA' shown below in Figure 5.4:

```
to paybacka
pr [FOR PROJECT A] make "x 0 make "q 0
make "x :cc notea pr (se :c "years "+ :q
) if :x > :bb [stop]
make "x :x + :dd notea pr (se :d "years
"+ :q) if :x > :bb [stop]
make "x :x + :ee notea pr (se :e "years
"+ :q) if :x > :bb [stop]
make "x :x + :ff notea pr (se :f "years
"+ :q) if :x > :bb [stop]
make "x :x + :gg notea pr (se :g "years
"+ :q) if :x > :bb [stop]
make "x :x + :hh notea pr (se :h "years
"+ :q) if :x > :bb [stop]
make "x :x + :jj notea pr (se :j "years
"+ :q) if :x > :bb [stop]
make "x :x + :kk notea pr (se :k "years
"+ :q) if :x > :bb [stop]
make "x :x + :ll notea pr (se :l "years
"+ :q) if :x > :bb [stop]
make "x :x + :mm notea pr (se :m "years
"+ :q) if :x > :bb [stop]
repeat 3 [pr []]
end
```

Procedure PAYBACKA

FIGURE 5.4

The first line of 'PAYBACKA' sets the variables X and Q to zero.

'X' is the cumulative cash flow for the investment and Q is the amount of money left over at the end of a given year after the investment has paid for itself.

The second line makes X equal to CC (8000) then moves to the procedure 'NOTEA'.

```
to notea
pr (se "total "return "to "date :x)
if :x > :bb [make "q :x - :bb pr (se "pa
id "back "within)]
end
```

Procedure NOTEA

FIGURE 5.5

This procedure shows the cash flow to date and looks to see if this total (X) is larger than the original investment (BB) (£25000). If it is, Q is calculated and the message

PAID BACK WITHIN

is printed. Now back to line 2 of 'PAYBACKA' where the number of years plus the surplus is printed and the procedure is stopped.

Type:

```
paybacka
```

and you will be told that the investment pays for itself within 4 years plus a bonus of £7000 at the end of that year. (The variables saved in e.g. 'dis1' will have to be present.)

You can now edit this procedure to enable the payback period on proposals B and C. I've called them 'PAYBACKB', 'PAYBACKC', 'NOTEB' and 'NOTEC' and here they are:

```
to paybackb
pr [FOR PROJECT B] make "x 0 make "q 0
make "x :ccc noteb pr (se :c "years "+ :
q) if :x > :bbb [stop]
make "x :x + :ddd noteb pr (se :d "years
 "+ :q) if :x > :bbb [stop]
make "x :x + :eee noteb pr (se :e "years
 "+ :q) if :x > :bbb [stop]
make "x :x + :fff noteb pr (se :f "years
 "+ :q) if :x > :bbb [stop]
make "x :x + :ggg noteb pr (se :g "years
 "+ :q) if :x > :bbb [stop]
make "x :x + :hhh noteb pr (se :h "years
 "+ :q) if :x > :bbb [stop]
make "x :x + :jjj noteb pr (se :j "years
 "+ :q) if :x > :bbb [stop]
make "x :x + :kkk noteb pr (se :k "years
 "+ :q) if :x > :bbb [stop]
make "x :x + :lll noteb pr (se :l "years
 "+ :q) if :x > :bbb [stop]
make "x :x + :mmm noteb pr (se :m "years
 "+ :q) if :x > :bbb [stop]
repeat 3 [pr []]
end
to noteb
pr (se "total "return "to "date :x)
if :x > :bbb [make "q :x - :bbb pr (se "
paid "back "within)]
end
```

Procedures PAYBACKB and NOTEB

FIGURE 5.6a

```
to paybackc
pr [FOR PROJECT C] make "x 0 make "q 0
make "x :cccc notec pr (se :c "years "+
:q) if :x > :bbbb [stop]
make "x :x + :dddd notec pr (se :d "year
s "+ :q) if :x > :bbbb [stop]
make "x :x + :eeee notec pr (se :e "year
s "+ :q) if :x > :bbbb [stop]
make "x :x + :ffff notec pr (se :f "year
s "+ :q) if :x > :bbbb [stop]
make "x :x + :gggg notec pr (se :g "year
s "+ :q) if :x > :bbbb [stop]
make "x :x + :hhhh notec pr (se :h "year
s "+ :q) if :x > :bbbb [stop]
make "x :x + :jjjj notec pr (se :j "year
s "+ :q) if :x > :bbbb [stop]
make "x :x + :kkkk notec pr (se :k "year
s "+ :q) if :x > :bbbb [stop]
make "x :x + :llll notec pr (se :l "year
s "+ :q) if :x > :bbbb [stop]
make "x :x + :mmmm notec pr (se :m "year
s "+ :q) if :x > :bbbb [stop]
repeat 3 [pr []]
end
to notec
pr (se "total "return "to "date :x)
if :x > :bbbb [make "q :x - :bbbb pr (se
 "paid "back "within)]
end
```

Procedures PAYBACKC and NOTEC

FIGURE 5.6b

How having RUN PAYBACKA, B and C you might be able to make a decision as to the best investment. However, the payback method of analysis does not consider TOTAL revenue earned. It might be useful therefore to know

1) The total revenue generated by each proposal;
2) The average annual revenue generated over the life of the proposal;
3) The nett total (allowing for 100% depreciation of the original investment during its life);
4) The nett average income (assuming straight line 100% depreciation over the life of the project).

Before you go on to do this however, your Logo working space will be a little full, so save the procedures thus far defined in this chapter under the file heading

ch5am (chapter 5 amstrad)

Now type:

```
erall
```

and re–load 'ch5am'.

Now we need some of this information to enable the next set of calculations to be made, so type:

```
er "ask
er "ask2
er "paybacka
er "paybackb
er "paybackc
er "notea
er "noteb
er "notec
```

and this should leave you with the 'DISPLAY' procedure in your computer's memory.

Now let's consider some other procedures to evaluate the information contained within the 'DISPLAY' procedure.

Take a look at the proposed procedures to do just this for investment proposal A.

```
to totala
if :cc > 0 [make "x :cc make "xx :x]
if :dd > 0 [make "x :x + :dd make "xx :x
 / 2]
if :ee > 0 [make "x :x + :ee make "xx :x
 / 3]
if :ff > 0 [make "x :x + :ff make "xx :x
 / 4]
if :gg > 0 [make "x :x + :gg make "xx :x
 / 5]
if :hh > 0 [make "x :x + :hh make "xx :x
 / 6]
if :jj > 0 [make "x :x + :jj make "xx :x
 / 7]
if :kk > 0 [make "x :x + :kk make "xx :x
 / 8]
if :ll > 0 [make "x :x + :ll make "xx :x
 / 9]
if :mm > 0 [make "x :x + :mm make "xx :x
 / 10]
```

```
repeat 3 [pr []] pr [for A] pr (se "TOTA
L "GROSS "RETURN :x) pr []
pr (se "AV. "ANNUAL "INCOME :xx) pr []
make "z :x - :bb make "qq (:xx - (:bb /
:zz)) pr (se "TOTAL "NETT "RETURN :z)
pr [] pr (se "AV. "NETT "ANNUAL "INCOME
:qq) repeat 3 [pr []]
end
```

Procedure TOTALA

FIGURE 5.7

This procedure during the first ten lines does two things. First of all, it keeps track of the cumulative earnings of the project, using the variable X. It also keeps track of the average annual income XX.

Then in line eleven it prints the gross cash income (X) and the average annual income (XX). It then, in line thirteen, sets a variable Z to X minus BB (cost of project) which will give the NETT (after depreciation) cash income earned. It also calculates QQ which is the average annual income minus the average rate of depreciation (BB/ZZ).

Thus, if you RUN 'TOTALA' you should receive this message:

```
FOR PROJECT A
TOTAL GROSS RETURN 80000

AVERAGE ANNUAL INCOME 8000

TOTAL NETT RETURN 55000

AVERAGE NETT ANNUAL INCOME 5500


?
```

Income From Project A

FIGURE 5.8

Now here are the procedures for the same calculations with B and C.

```
to totalb
if :ccc > 0 [make "x :ccc make "xx :x]
if :ddd > 0 [make "x :x + :ddd make "xx
:x / 2]
if :eee > 0 [make "x :x + :eee make "xx
:x / 3]
if :fff > 0 [make "x :x + :fff make "xx
:x / 4]
if :ggg > 0 [make "x :x + :ggg make "xx
:x / 5]
if :hhh > 0 [make "x :x + :hhh make "xx
:x / 6]
if :jjj > 0 [make "x :x + :jjj make "xx
:x / 7]
if :kkk > 0 [make "x :x + :kkk make "xx
:x / 8]
if :lll > 0 [make "x :x + :lll make "xx
:x / 9]
if :mmm > 0 [make "x :x + :mmm make "xx
:x / 10]
repeat 3 [pr []] pr [for B] pr (se "TOTA
L "GROSS "RETURN :x) pr []
pr (se "AV. "ANNUAL "INCOME :xx) pr []
make "z :x - :bbb make "qq (:xx - (:bbb
/ :zzz)) pr (se "TOTAL "NETT "RETURN :z
)
pr [] pr (se "AV. "NETT "ANNUAL "INCOME
:qq) repeat 3 [pr []]
end
```

Procedure TOTALB

FIGURE 5.9a

```
to totalc
if :cccc > 0 [make "x :cccc make "xx :x]

if :dddd > 0 [make "x :x + :dddd make "x
x :x / 2]
if :eeee > 0 [make "x :x + :eeee make "x
x :x / 3]
if :ffff > 0 [make "x :x + :ffff make "x
x :x / 4]
if :gggg > 0 [make "x :x + :gggg make "x
x :x / 5]
if :hhhh > 0 [make "x :x + :hhhh make "x
x :x / 6]
```

```
if :jjjj > 0 [make "x :x + :jjjj make "x
x :x / 7]
if :kkkk > 0 [make "x :x + :kkkk make "x
x :x / 8]
if :llll > 0 [make "x :x + :llll make "x
x :x / 9]
if :mmmm > 0 [make "x :x + :mmmm make "x
x :x / 10]
repeat 3 [pr []] pr [for C] pr (se "TOTA
L "GROSS "RETURN :x) pr []
pr (se "AV. "ANNUAL "INCOME :xx) pr []
make "z :x - :bbbb make "qq (:xx - (:bbb
b / :zzzz)) pr (se "TOTAL "NETT "RETURN
 :z)
pr [] pr (se "AV. "NETT "ANNUAL "INCOME
:qq) repeat 3 [pr []]
end
```

Procedure TOTALC

FIGURE 5.9b

Having defined TOTALA, TOTALB, TOTALC and with 'DISPLAY' still in memory, create a disc file called:

ch5bam

and having made that file, get rid of TOTALA, TOTALB and TOTALC leaving DISPLAY to be a procedure in the last disc file called:

ch5cam

Thus far the TIMING of future cash flows has not been taken into account when considering a given investment. In 'A' for example, where £25000 is to be invested at the start of the project, the £8000 income at the end of year 10 will be worth less than the £8000 at the end of year 1. What has to be considered of course is the 'TIME VALUE OF MONEY'. The notion of cumulative or compound interest will be useful here. Let's imagine that you invested £100 at 10% interest (paid at the end of each year) per annum. You will expect to see your investment grow in the following way:

Year	
0	(100)
1	110
2	121
3	133.1
4	146.41

etc.

From this you can see that, if you require a certain rate of return on money invested any future sums must be discounted so that a 'PRESENT VALUE' of a future cash inflow can be considered. Thus £8000 in ten years time will only be 'WORTH' just over £3000 in present day value terms if we require a return of 10% on an investment — i.e. if we invest £3084 now, we will get £8000 in ten year's time with a 10% compound interest rate.

The formula used to calculate the compound interest on a sum of money is

$$\text{The future sum} = X * (1+\text{interest rate})^N$$

The amount to be invested

Say 10% (0.1)

The number of years over which the investment will run

Let's see what £100 will be worth in FOUR year's time using this formula.

$$\text{Future sum} = 100 \times (1 + 0.1)^4$$

$$= 100 \times 1.4641$$

$$= £146.41$$

To work out instead the present value X given the future sum, you simply divide the future sum by the part in brackets to the appropriate power:

$$\text{Pres. value X} = \frac{\text{Future sum}}{(1 + 0.1)^N}$$

In order to calculate this present value, consider the procedure 'DEPA' shown below.

```
to work
make "ww 1 + (:pp / 100)
make "t :ww
repeat :w - 1 [make "ww :ww * :t]
end
to depa
work
make "x :x / :ww
make "w :w + 1
pr :x make "qqqq :qqqq + :x
end
```

Procedures WORK and DEPA

FIGURE 5.10

The work procedure does the job of:

$$(1 + \text{INTEREST RATE})^{N}$$

:pp will be given by the user when the DISCOUNT procedure is called.

This will be :w

you can see that the procedure to 'raise to the power of' has been used.

X is the amount of income in a given year and W is the year for which the calculation is to be made. Each time the procedure is run, W is incremented by adding 1 to its previous value. The variable QQQQ is used to keep track of the cumulative discounted totals.

Now consider the procedure 'TOTALADEP' which, for project A, looks at the future cash flow in the first year, sets that figure to X and carries out 'DEPA'. It then, of course, does the same for each of the other future cash flows. The total 'PRESENT VALUE' of the future cash flows will be displayed: also the NETT given above the required rate.

```
to totaladep
make "x :cc depa
make "x :dd depa
make "x :ee depa
make "x :ff depa
make "x :gg depa
make "x :hh depa
make "x :jj depa
make "x :kk depa
make "x :ll depa
make "x :mm depa
make "xxxx (:qqqq - :bb)
repeat 3 [pr []] pr [FOR PROJECT A]
pr (se "TOTAL "GROSS "CASH "INFLOW :qqqq
) pr []
pr (se "USING :pp "RATE "OF "RETURN)
pr (se "NETT "GAIN "ABOVE "REQUIRED "RAT
E :xxxx)
end
```

Procedure TOTALADEP

FIGURE 5.11

After putting in TOTALADEP type:

```
make "pp 10
make "w 1
make "qqqq 0
```

Now type:

```
totaladep
```

And the present values of future cash flows will be displayed for you. You will see that the inflow above the required rate is considerable for investment 'A'. Here are the procedures for 'Discounted Cash Flow' (DCF) analysis for projects B and C:

```
to totalbdep
make "x :ccc depa
make "x :ddd depa
make "x :eee depa
make "x :fff depa
make "x :ggg depa
make "x :hhh depa
make "x :jjj depa
make "x :kkk depa
make "x :lll depa
make "x :mmm depa
make "xxxx (:qqqq - :bbb)
repeat 3 [pr []] pr [FOR PROJECT B]
pr (se "TOTAL "GROSS "CASH "INFLOW :qqqq
) pr []
pr (se "USING :pp "RATE "OF "RETURN)
pr (se "NETT "GAIN "ABOVE "REQUIRED "RAT
E :xxxx)
end
```

Procedure TOTALBDEP

FIGURE 5.12a

```
to totalcdep
make "x :cccc depa
make "x :dddd depa
make "x :eeee depa
make "x :ffff depa
make "x :gggg depa
make "x :hhhh depa
make "x :jjjj depa
make "x :kkkk depa
make "x :llll depa
make "x :mmmm depa
make "xxxx (:qqqq - :bbbb)
repeat 3 [pr []] pr [FOR PROJECT C]
pr (se "TOTAL "GROSS "CASH "INFLOW :qqqq
) pr []
```

```
pr (se "USING :pp "RATE "OF "RETURN)
pr (se "NETT "GAIN "ABOVE "REQUIRED "RAT
E :xxxx)
end
```

Procedure TOTALCDEP

FIGURE 5.12b

Now for a procedure called 'DISCOUNT' which will carry out DCF analysis automatically if required.

```
to discount
pr [WHAT RATE OF RETURN > ?]
make "pp item 1 rl
make "w 1
make "qqqq 0
totaladep
make "w 1
make "qqqq 0
totalbdep
make "w 1
make "qqqq 0
totalcdep
end
```

Procedure DISCOUNT

FIGURE 5.13

In the first line, the required rate of return is put in by the user (say 25%) and after that the calculation and display is automatic. All that is needed is a procedure to control everything, but first save the procedures currently in memory (DISPLAY, TOTALADEP, TOTALBDEP, TOTALCDEP, WORK, DEPA & DISCOUNT) under the filename 'ch5cam'.

Now while you have been using various techniques during this chapter, your disc files will contain some unwanted and sometimes burdensome information; for example if you type:

```
erall
load "ch5am
poall
```

this will display the whole contents of Logo memory.

you will see, as expected, the various procedures saved under that filename and also all the variable values. Now when you load in that file, those variables are set as well. So, if you are to make use the various procedures, here's one called START which will clean up the variable values by setting them to zero.

```
to start
make "z 0 make "x 0 make "zz 0 make "q 0
 make "xx 0 make "zzz 0 make "m 0 make
"l 0
make "k 0 make "j 0 make "zzzz 0 make "h
 0 make "g 0 make "f 0 make "e 0 make "
d 0
make "c 0 make "b 0 make "qq 0 make "mm
0 make "ll 0 make "kk 0 make "jj 0 make
 "hh 0
make "gg 0 make "ff 0 make "ee 0 make "d
d 0 make "mmm 0 make "cc 0 make "bb 0 m
ake "lll 0
make "kkk 0 make "jjj 0 make "hhh 0 make
 "ggg 0 make "mmmm 0 make "fff 0 make "
input 0 make "llll 0 make "eee 0 make "
ddd 0
make "ddd 0 make "kkkk 0 make "ccc 0 mak
e "jjjj 0 make "bbb 0 make "hhhh 0 make
 "gggg 0
make "ffff 0 make "eeee 0 make "dddd 0 m
ake "cccc 0 make "bbbb 0
end
```

Procedure START

FIGURE 5.14

Another useful procedure will be one called DATA.

```
to data
start display
er "start er "display recycle
load "ask1 ask
er "ask er "ask2 erasefile "data save "d
ata
end
```

Procedure DATA

FIGURE 5.15

Now define DATA and save the procedures START, DISPLAY ad DATA under the filename 'DATA1'.

Now create a filename called DATA just by saying:

```
save "data
```

If you now erase everything from the computer's memory by typing:

```
erall
```

and then you

```
load data1
```

and then call

```
data
```

1) all variables will be set to zero;
2) the investment matrix will be displayed;
3) the START and DISPLAY procedures are then erased;
4) the ASK1 file is loaded and the necessary infromation collected from the user;
5) ASK and ASK2 are erased and the variable values saved under a filename 'DATA'.

If you now say:

```
erall
load "data
poall
```

You will see the values you've just put in come up on the screen. (You will have to press RETURN a few times).

Now you can say:

```
erall
load "ch5am
load "data
paybacka paybackb paybackc
```

and your latest data will be analysed.

Now say:

```
erall
load "ch5bam
load "data
totala totalb totalc
```

and of course you finish off with:

```
erall
load "ch5cam
load "data
discount totaladep totalbdep totalcdep
```

Now you can build a super procedure to carry out interactive investment analysis.

Chapter 6

Production Scheduling

In Chapter 2, you learned how

```
item
```

worked. For example if you say:

```
item 3 "logo
```

the G, i.e. the third item, will be the resultant output.

You could also ask for:

```
item 6 [a b c d e f]
```

and 'F' will result

You can also say:

```
make "gg [a b c d e f]
```

followed by:

```
pr item 3 :gg
```

with 'C' resulting.

Now try saying:

```
make "qq [logo works]
```

followed by:

```
pr :gg
pr :qq
```

Now try saying:

```
make "qq :gg
pr "qq
```

and you will see that the contents of the variable 'GG' are now the contents of the variable 'QQ'.

Now this facility might act as the basis for a data handling system and will prove very useful during this section of the book which introduces you to the art of PRODUCTION SCHEDULING.

Now imagine the scene. You, as production manager, are faced with the problem of having to decide the order in which a number of different jobs of different sizes are to be listed so that they can be worked on by two production areas in the shortest possible time. The other major constraint is that all the jobs must go through area A first, followed by B. The times required for each job in each area are shown below.

	A	**B**
JOB1	9	7
JOB2	8	11
JOB3	2	9
JOB4	3	1
JOB5	5	7
JOB6	2	11

Now you can see from the above list that 'JOB1' requires 9 days in production area A and 7 days in production area B and indeed that all of the jobs require time in each of the areas.

Suppose for a moment that the jobs are to be done in exactly the order in which they are listed above. What would be the total elapsed time through A and B for all the jobs, remembering that each of them must go through A first, then B? Well, here's the solution for the first job on the list.

	A		**B**	
	in	**out**	**in**	**out**
JOB1	0	9	9	16

The first job will enter A at time zero and will emerge after 9 days, so it will emerge from B after a total of 16 days.

i.e:

Now to continue with the second job:

	A		B	
	in	out	in	out
JOB1	0	9	9	16
JOB2	9	17	17	28

You can see that the second job can't start in A until the first job is finished and that it can then go straight into B because the first job is by then finished in B. In fact the B facility will not be working for a day. Now for the rest of the jobs:

	A		B	
	in	out	in	out
JOB1	0	9	9	16
JOB2	9	17	17	28
JOB3	17	19	28	37
JOB4	19	22	37	38
JOB5	22	27	38	45
JOB6	27	29	45	56

It looks as though using this order for the jobs, the total throughput time is 56 days and that most jobs will have to wait between A and B.

Let's try a different order to try and reduce the time:

	A		B	
	in	out	in	out
JOB4	0	3	3	4
JOB6	3	5	5	16
JOB1	5	14	16	23
JOB2	14	22	23	34
JOB3	22	24	34	43
JOB5	24	29	43	50

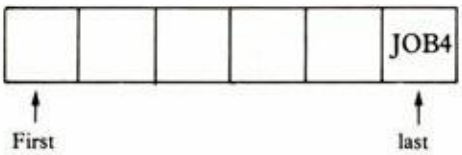
Now, using the order 4 6 1 2 3 5, the time has been cut to 50 days. There are 720 different combinations for these six jobs and it would take all day to work through all the options to find the shortest time. There is however an ALGORITHM (method of calculation) that guarantees to schedule any number of jobs through two centres (in the order A B) in the shortest possible time! Here's how it works:

1) Take the shortest time in the lists of times. (If there is duplication it does not matter which one you choose.)
2) If that time is for the second operation, schedule that job last. If it is for the first operation, schedule that job first instead.
3) Having scheduled that job, take it from the list.
4) Repeat operations 1) to 3), scheduling the jobs as close to first or last as possible.

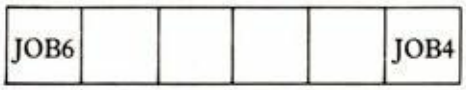
Here's our example:

	A	B
JOB1	9	7
JOB2	8	11
JOB3	2	9
JOB4	3	1
JOB5	5	7
JOB6	2	11

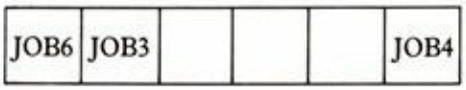
Now JOB4 contains the shortest time, and it's in the second operation, so JOB4 will be scheduled last.



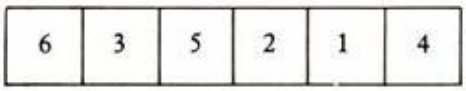
From the reduced list either JOB3 or JOB6 can be scheduled first. I've chosen JOB6.



This means that JOB3 will be scheduled as near to 'first' as possible, which will be in the second position.



If you carry on along these lines your schedule will look like this:–



Let's see what the total elapsed time is:

	A		B	
	in	out	in	out
JOB6	0	2	2	13
JOB3	2	4	13	22
JOB5	4	9	22	28
JOB2	9	17	29	40
JOB1	17	26	40	47
JOB4	26	29	47	48

Well it looks as though the minimum time is 48 days.

How about a LOGO procedure to carry out this particular algorithm?

Let's start off with a procedure that will 'SET' the times for the various jobs.

```
to set
make "j1 [1 9 7 0]
make "j2 [2 8 11 0]
make "j3 [3 2 9 0]
make "j4 [4 3 1 0]
make "j5 [5 5 7 0]
make "j6 [6 2 11 0]
end
```

Procedure SET

FIGURE 6.1

As you can see, the jobs J1 to J6 inclusive have been set to four items.

The first item is simply the job number.
The last item is zero and will be used as a 'FLAG' or 'INDICATOR' when a particular job has been scheduled.

The second and third items represent the times for each job in production centres A and B.

Now in order to be able to find the smallest times in each of the A and B columns, a method sometimes called a 'bubble sort' will be used. In our case, the bubble sort will make the smallest figures rise to the top of the column.

Consider the first two jobs:

		Job No.	A	B	flag
JOB1	J1 =	[1	9	7	0]
JOB2	J2 =	[2	8	11	0]

Now the logic for finding the smallest value for A will be:

If 'A' (9) for JOB1 is larger than 'A' for JOB2, swap the contents of J1 for the contents of J2.

Thus

```
J1   [1    9    7    0]
J2   [2    8   11    0]
J3   [3    2    9    0]
J4   [4    3    1    0]
J5   [5    5    7    0]
J6   [6    2   11    0]
```

Will become:

```
J1   [2    8   11    0]  } Notice the change here
J2   [1    9    7    0]  }
J3   [3    2    9    0]
J4   [4    3    1    0]
J5   [5    5    7    0]
J6   [6    2   11    0]
```

Now we can repeat the logic by saying:

"If item 2 of J2 is larger than item 2 of J3 then swap them over".

This will result (in our example) in:

```
J1   [2    8   11    0]
J2   [3    2    9    0]  } (Again the change)
J3   [1    9    7    0]  }
J4   [4    3    1    0]
J5   [5    5    7    0]
J6   [6    2   11    0]
```

Continuing the logic, the '9' as the largest 'ITEM 2' will 'SINK' to the bottom of the list.

```
J1   [2    8   11    0]
J2   [3    2    9    0]
J3   [4    3    1    0]
J4   [5    5    7    0]
J5   [6    2   11    0]
J6   [1    9    7    0]
```

Now type:

```
set
```

Followed by:

```
:j1
:j2
```

and the values will be displayed. Now type:

```
if item 2 :j1 > item 2 :j2 [make "j2 :j1]
```

Now type:

```
:j2
```

You will see that values have been swapped. OR HAVE THEY?

Type:

```
:j1
```

You will see that you have lost the original contents of J2.

It's fairly obvious what has happened; consider this example:

J1 = X
J2 = Y

If we say "MAKE J1 = Contents of J2, J1 will become Y and X will have been lost.

In order to overcome this, a dummy variable can be used by saying:

J1 = X
J2 = Y

Make 'C' the contents of J1 then make J1 the contents of J2 and J2 the contents of C. Then you will end up with:

J1 = Y
J2 = X

Using this technique, the first part of the first line of a procedure 'SORT.A' could read:

```
if and (item 4 :j1 = 0) (item 2 :j1 > item 2 :j2)
```

The last item of J1 is zero and therefore able to be considered (this 'flag' tells us whether we have already checked :J1. In this case we haven't).

The second item of J1 is greater than the second item of J2.

The last part of this line will read:

```
[make "c1 :j1 make "j1 :j2 make "j2 :c1]
```

This will swap the contents of the two variables by making use of the dummy variable C1.

The other lines in this procedure will continue the logic.

```
to sort.a
if and (item 4 :j1 = 0) (item 2 :j1 > it
em 2 :j2) [make "c1 :j1 make "j1 :j2 ma
ke "j2 :c1]
if and (item 4 :j2 = 0) (item 2 :j2 > it
em 2 :j3) [make "c1 :j2 make "j2 :j3 ma
ke "j3 :c1]
if and (item 4 :j3 = 0) (item 2 :j3 > it
em 2 :j4) [make "c1 :j3 make "j3 :j4 ma
ke "j4 :c1]
if and (item 4 :j4 = 0) (item 2 :j4 > it
em 2 :j5) [make "c1 :j4 make "j4 :j5 ma
ke "j5 :c1]
if and (item 4 :j5 = 0) (item 2 :j5 > it
em 2 :j6) [make "c1 :j5 make "j5 :j6 ma
ke "j6 :c1]
end
```

Procedure SORT.A

FIGURE 6.2

Let's now define an illustrative procedure to show that 'SORT.A' is working properly.

```
to z
pr :j1
pr :j2
pr :j3
pr :j4
pr :j5
pr :j6
end
```

A Procedure Z To Test SORT.A

FIGURE 6.3

This of course will print out the contents of the variables.

Try calling:

```
?set
?sort.a
?z
2 8 11 0
3 2 9 0
4 3 1 0
5 5 7 0
6 2 11 0
1 9 7 0
```

The Result Of Calling SORT.A

FIGURE 6.4

and you will see in Figure 6.4 that the first job has sunk to the bottom of the list.

Now try:

```
?set
?repeat 6[sort.a z pr [] pr []]
2 8 11 0
3 2 9 0
4 3 1 0
5 5 7 0
6 2 11 0
1 9 7 0

3 2 9 0
6 2 11 0
4 3 1 0
5 5 7 0
2 8 11 0
1 9 7 0

3 2 9 0
4 3 1 0
5 5 7 0
6 2 11 0
2 8 11 0
1 9 7 0

3 2 9 0
6 2 11 0
4 3 1 0
5 5 7 0
2 8 11 0
1 9 7 0

3 2 9 0
4 3 1 0
6 2 11 0
5 5 7 0
2 8 11 0
1 9 7 0

3 2 9 0
6 2 11 0
4 3 1 0
5 5 7 0
2 8 11 0
1 9 7 0
```

Bubble Sort In Action

FIGURE 6.5

Now it's a very simple step to edit 'SORT.A' to give 'SORT.B' as shown below.

```
to sort.b
if and (item 4 :j1 = 0) (item 3 :j1 > it
em 3 :j2) [make "c1 :j1 make "j1 :j2 ma
ke "j2 :c1]
if and (item 4 :j2 = 0) (item 3 :j2 > it
em 3 :j3) [make "c1 :j2 make "j2 :j3 ma
ke "j3 :c1]
if and (item 4 :j3 = 0) (item 3 :j3 > it
em 3 :j4) [make "c1 :j3 make "j3 :j4 ma
ke "j4 :c1]
if and (item 4 :j4 = 0) (item 3 :j4 > it
em 3 :j5) [make "c1 :j4 make "j4 :j5 ma
ke "j5 :c1]
if and (item 4 :j5 = 0) (item 3 :j5 > it
em 3 :j6) [make "c1 :j5 make "j5 :j6 ma
ke "j6 :c1]
end
```

Procedure SORT.B

FIGURE 6.6

All you have to do is to change the ITEM number from 2 to 3 in each case – and don't forget the procedure's name!

Now for a procedure called 'SCHEDULE' to complete the job.

```
to schedule
repeat 6 [sort.a]
make "first.job item 1 :j1 make "j1 [0 0
 0 1]
repeat 6 [sort.b]
make "last.job item 1 :j2 make "j2 [0 0
0 1]
repeat 6 [sort.a]
make "second.job item 1 :j3 make "j3 [0
0 0 1]
repeat 6 [sort.b]
make "fifth.job item 1 :j4 make "j4 [0 0
 0 1]
repeat 6 [sort.a]
make "third.job item 1 :j5 make "j5 [0 0
 0 1]
make "fourth.job item 1 :j6
pr (se :first.job :second.job :third.job
 :fourth.job :fifth.job :last.job)
end
```

Procedure SCHEDULE

FIGURE 6.7

The first line carries out the first bubble sort; the second line sets a variable 'FIRST.JOB' to the first item in J1. This of course is the job number of the first job to be scheduled. J1 is then reset, making the last item 1 so that it won't be considered again by the SORT procedures.

The logic continues until the jobs in the order in which they are to be scheduled are printed.

Now try typing:

```
set
schedule
```

and you will receive the list:

3 6 2 1 5 4

which will give the shortest total time through A and B. Let's try it out!

	A		B	
	in	out	in	out
JOB3	0	2	2	11
JOB6	2	4	11	22
JOB2	4	12	22	33
JOB1	12	21	33	40
JOB5	21	26	40	47
JOB4	26	29	47	48

Now earlier on, carrying through the algorithm manually we arrived at a schedule of

6 3 5 2 1 4

Which gave the same time as:

3 6 2 1 5 4

In terms of total time it does not matter which schedule is used.

The more observant among you may have noticed that 'SCHEDULE' will not necessarily work if the smallest times for both A and B are for the same job and if the B time is less than the A time, because A gets checked first. You might like to investigate ways around this problem, if you like puzzles!

Now under certain conditions, the algorithm as it stands will work for three work centres, for example:

	A	B	C
JOB1	27	16	13
JOB2	25	12	16
JOB3	26	13	11
JOB4	19	18	14
JOB5	34	17	15
JOB6	32	18	12

In order for the algorithm to work, *one* of the two following conditions must be true.

1) The smallest in A must be as large or larger than the largest in B; i.e.

$$AMIN > BMAX$$

2) The smallest in C is as large or larger than the largest in B; i.e.

$$CMIN > BMAX$$

Now you can see that the first condition is true, allowing the algorithm to apply.

The method is to add the 'A' times to the 'B' times and create a fictitious centre X.

Then add the 'C' times to the 'B' times and create another fictitious centre Z. Thus the scheduling problem looks like this:

	X	**Z**
JOB1	43	29
JOB2	37	28
JOB3	39	24
JOB4	37	32
JOB5	51	32
JOB6	50	30

The job of scheduling can now of course proceed as normal.

Chapter 7

Cost Effective Ordering

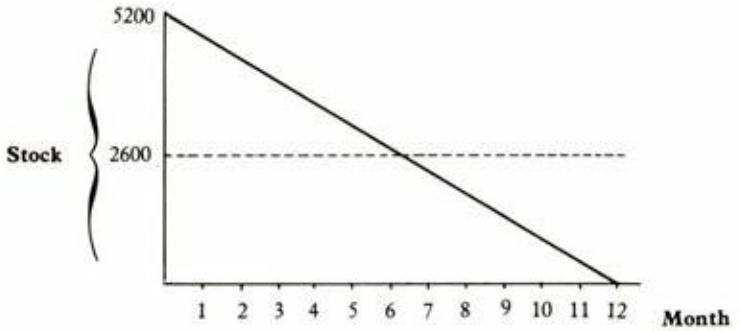
In most cases, the proprietor of a business knows that it costs money to hold stock. This 'stock–holding cost' is of course a disadvantage and should of course be minimised. One way of minimising the cost is to hold no stock, but then of course there are other disadvantages associated with loss of goodwill, loss of production while waiting for goods, loss of customers and money etc. Holding stock then, becomes a fine balance between the advantages and the disadvantages.

Let's for a moment consider the costs of holding an item of stock. Well there's shelf space; insurance; heating lighting and ventilation of stores; security; direct stores labour and of course the 'cost' of the money invested in the stock. Now in a simple model for stock systems all these costs are considered as a given percentage of stock worth. For example if you held 100 items valued at £2.50 each and your company's cost accountant told you that it costs 15% pa of stock worth to hold that stock, then the calculation is:

$$100 \times £2.5 \times 0.15 = £37.5$$

i.e. if you held £250 worth of stock for a year it would cost £37.50. Now the 15% used in this example will vary from company to company. For some the cost might be as high as 30% whilst for others it might be as low as 2%. The simple model also assumes that there are costs associated with placing an order. This cost will comprise paperwork, postage, inspection when goods arrive, accounting etc. Now, again, this cost will vary from company to company but a cost of £20 per order is not an unreasonable estimate.

Now let's assume the company holds stock of a particular item of which it sells or uses 5200 units a year and gradually diminishes its stocks like this:



One Way Of Using Up Stock

FIGURE 7.1

In Figure 7.2, the 5200 items are delivered on day one and are sold over the following 12 months.

What does this cost?

Well, the average stock level is 2600 items (5200÷2).

Let's assume each item costs £3 and that the stock–holding costs are 20%. Given this, it's easy to see that the business' stock–holding cost per item will be:

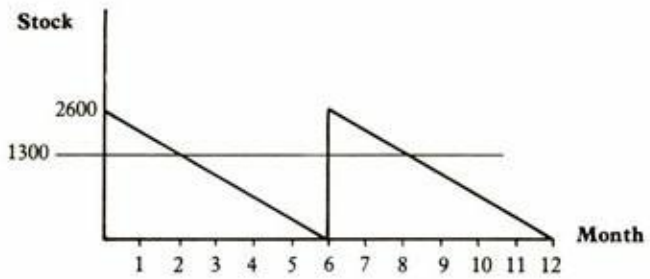
$$2600 \times £3 \times 0.2$$

- 2600: Average number of units in stock throughout the year
- £3: unit cost
- 0.2: % stock holding cost (20%)

Now this comes to £1560. The cost of ordering now has to be added; assuming £20 per order the total cost becomes:

$$£1560 + £20 = £1580$$

Now it will be argued that this cost could be reduced by ordering twice a year so that the consumption curve looks like this:



Another Way Of Using Up Stock

FIGURE 7.2

Thus the cost will be:

$$1300 \times £3 \times 0.2 + 40 = £820$$

- 1300: Average stock level
- £3: unit cost
- 40: 2 orders at £20 each

Now given the massive reduction in costs simply by increasing the number of orders from one to two each year, why not order each week?

Thus you will have:

$$50 \times £3 \times 0.2 + (52 \times 20) = £1070$$

Average stock level (5200 ÷ 50 ÷ 2)

Weekly orders at £20 each

As you can see, total costs have gone up again. So what is the best ordering frequency?

One way to find out is to design a LOGO procedure to work out the total costs for a given quantity ordered. First of all, let's tabulate the inputs:

1) The annual demand = D
2) The price of each item = P
3) The holding percentage = I
4) The cost of an order = C

Now consider the following:

3 0.2

Number of orders per year :AV.ST*:P*:I :HOLDING.COST+:ORDER.COST

"N.O"	"AV.ST"	"HOLDING.COST"	"ORDER.COST"	"TOTAL.COST"
1	2600	1560	20	1580
2	1300	780	40	820
3	867	520	60	580
5	520	312	100	412
10	260	156	200	356
12	217	130	240	370
15	173	104	300	404

Average Stock, this is calculated by (:D÷:N.O)÷2

Annual Demand

No. of orders per year

:N.O*:C

No. of orders per year Cost per order

Now you can see from the above table that the optimum average stock is about 260. Consider the procedure 'MINIMISE' shown below.

```
to minimise
pr [WHAT'S THE PRICE OF THE ITEM?]
make "p item 1 rl
pr [NOW THE HOLDING PERCENTAGE]
make "i item 1 rl make "i :i / 100
pr [WHAT IS THE ORDERING COST?]
make "c item 1 rl
pr [HOW MANY ITEMS PER YEAR?]
make "d item 1 rl
```

Procedure MINIMISE

FIGURE 7.3

As you can see, 'MINIMISE' simply collects relevant information from the user, remembering to reduce the holding percentage to a decimal fraction in line four.

Now have a look at 'CALCULATE1'. This procedure will work out values for the various types of costs associated with different order sizes in exactly the way demonstrated previously.

```
to calculate1
make "av.st ((:d / :n.o) / 2)
make "holding.cost ((:av.st * :p * :i) /
 10
make "ordering.cost ((:n.o * :c) / 10)
make "total.cost :holding.cost + :orderi
ng.cost
pr :ordering.cost
pr :holding.cost
pr :total.cost
make "b1 :n.o make "b2 :holding.cost
make "c1 :n.o make "c2 :total.cost
end
```

Procedure CALCULATE1

FIGURE 7.4

All will be clear in the above procedure apart from the last two lines. The variables B1, B2, C1, C3 are being used to remember some values which will be of use in the next part of the program. Notice also that the values of the costs have been reduced by a factor of 10. This again will enable the next part of the program to proceed normally.

If you now type:

```
make "n.o 2
minimise
```

Then answer the various questions with:

```
3
20
20
5200
```

Now type:

```
calculate1
```

You will get

4 ← Ordering cost (£40)
78 ← Holding cost (£780)
82 ← Total cost (£820)

Now edit 'MINIMISE' to include this last line:

```
make "n.o 1 calculate1 draw
```

Now let's take a look at the 'DRAW' procedure.

If you wanted to, you could 'MAKE' 'N.O' to any value and re–run 'CALCULATE1' again and again until you found an OPTIMUM value for 'N.O'; i.e. one that minimised total costs.

This, however, is a bit mundane so let's go to work on a procedure to display various costs for corresponding values of 'N.O'.

```
to draw
make "a2 :ordering.cost make "a1 :n.o
make "b2 :holding.cost make "b1 :n.o
make "c2 :total.cost make "c1 :n.o
graph
end
```

Procedure DRAW

FIGURE 7.5

Now the above procedure 'DRAW' simply 'remembers' some values, some of which are already in store, and then moves onto the procedure which will plot the various values.

```
to graph
ht fd 150 bk 150 rt 90 fd 250 bk 250 lt
90
setx :n.o * 10 pu fd :ordering.cost
make "a3 item 1 tf make "a4 item 2 tf
fd (:holding.cost - :ordering.cost)
make "b3 item 1 tf make "b4 item 2 tf
fd (:total.cost - :holding.cost)
make "c3 item 1 tf make "c4 item 2 tf
setpos [0 0] pd setpos (se :a3 :a4) pu
setpos (se :b1 :b2) pd setpos (se :b3 :b
4) pu
setpos (se :c1 :c2) pd setpos (se :c3 :c
4) pu
make "b1 :n.o * 10 make "b2 :holding.cos
t
make "c1 :n.o * 10 make "c2 :total.cost
calculate2
pu home graph
end
```

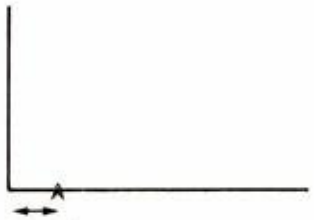
Procedure GRAPH

FIGURE 7.6

The first line of 'GRAPH' will draw the vertical 'money' (COST) axis 150 units long and the horizontal 'number of orders' axis 250 units long.

The second and third lines will:

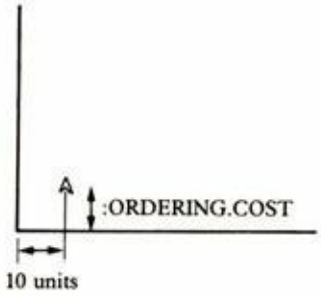
1) set the table to a value :N.O*10 (this has been done to stretch the scale of the horizontal axis);



The Operation Of GRAPH

FIGURE 7.7a

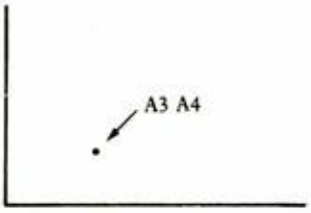
2) pen up and forward :ORDERING.COST;



The Operation Of GRAPH

FIGURE 7.7b

3) MAKE "A3 ITEM 1 TF (10) and then MAKE "A4 ITEM 2 TF (:ORDERING.COST).



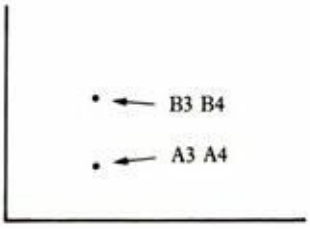
The Operation Of GRAPH

FIGURE 7.7c

This will 'remember' the position X=A3 and Y=A4.

The fourth and fifth lines will:

1) move forwards to B3 B4 and remember them.

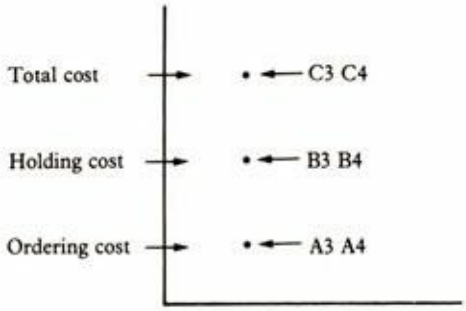


The Operation Of GRAPH

FIGURE 7.7d

The sixth, seventh and eighth lines will:

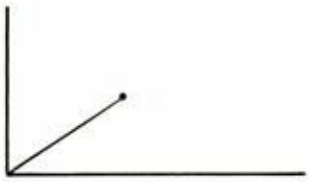
1) Remember C3 and C4



The Operation Of GRAPH

FIGURE 7.7e

2) set the turtle to the origin and, with it's pen down, move to A3,A4.

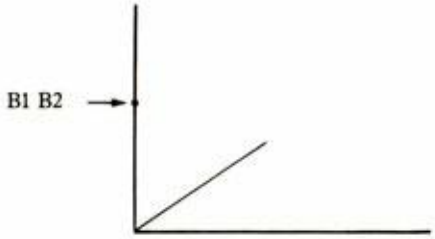


The Operation Of GRAPH

FIGURE 7.7f

The ninth line will:

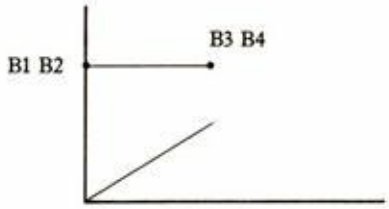
a) set the turtle's position to B1,B2;



The Operation Of GRAPH

FIGURE 7.7g

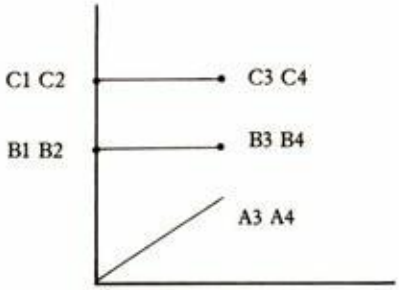
b) move to B3, B4;



The Operation Of GRAPH

FIGURE 7.7h

The tenth line will act in a similar manner to the ninth line.



The Operation Of GRAPH

FIGURE 7.7i

The next two lines will now reset the values of B1, B2, C1, C2 and move on to 'CALCULATE2' before re-calling 'GRAPH'

```
to calculate2
pr [HOW MANY ORDERS PER YEAR?]
make "n.o item 1 rl
make "av.st ((:d / :n.o) / 2)
make "holding.cost ((:av.st * :p * :i) /
 10)
make "ordering.cost ((:n.o * :c) / 10)
make "total.cost :holding.cost + :orderi
ng.cost
pr :ordering.cost
pr :holding.cost
pr :total.cost
end
```

Procedure CALCULATE2

FIGURE 7.8

This simply asks the user how many orders a year to try. It then goes on to calculate the new costs before 'GRAPH' re-calls itself ('RECURSIVE' activity) and the new costs plotted.

Now type:

```
cs
minimise
```

Followed by:

```
3
20
20
5200
```

This will give you your costs (divided by 10) for the order a year and your screen will look like the one shown below in Figure 7.9.

GRAPH's Actual Output

FIGURE 7.9

Compare the costs with those calculated earlier in the chapter.

Now answer with:

```
2
```

and you will get:

Two Orders Per Annum

FIGURE 7.10

Now try:

6

and you will see:

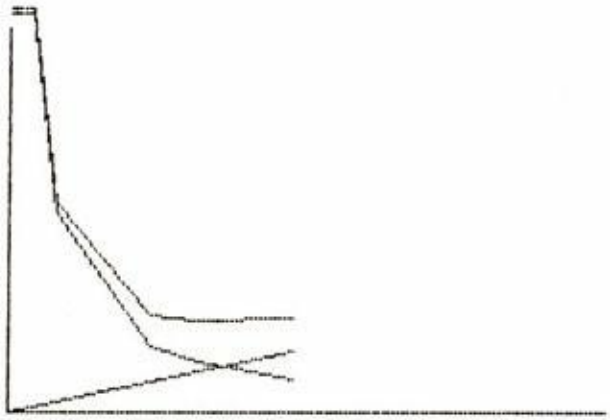
Six Orders Per Annum

FIGURE 7.11

Notice that total and holding costs are decreasing whilst ordering costs are increasing.

Try 8

Then 12:

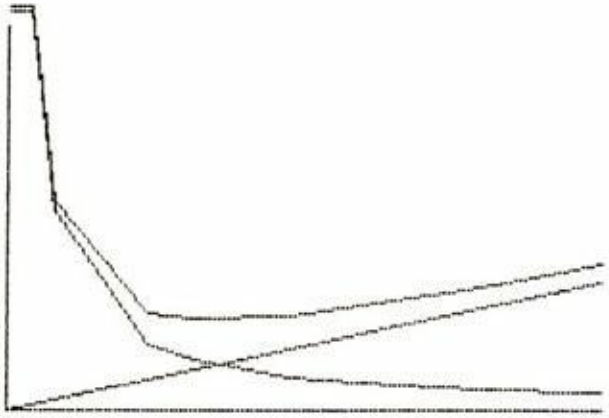


Up To Twelve Orders Per Annum

FIGURE 7.12

Notice that now the ordering costs are higher than the holding costs and that total costs have levelled out.

Now try 15, 20 & 25.



Up To Twenty-Five Orders Per Annum

FIGURE 7.13

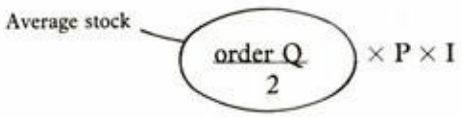
You can see now total costs rising quite steadily. You can of course achieve much smoother curves by taking ordering intervals of 1 or 0.5.

In order to stop this procedure hit CTRL–G on a 6128 or ALT–G or STOP on an 8256 or an 8512.

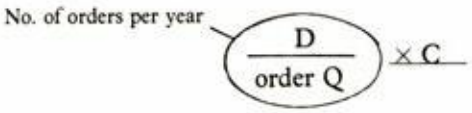
It's fairly obvious that the 'optimum' order quantity is where total costs are a minimum and this will be where the holding and ordering cost curves cross, i.e. where

$$\text{HOLDING COST} = \text{ORDERING COST}$$

Now the holding cost for any period is:



and the order cost is:



Thus

$$\frac{\text{order Q}}{2} \times P \times I = \frac{D}{\text{order Q}} \times C$$

So

$$(\text{order Q})^2 = \frac{2DC}{IP}$$

$$\text{order Q} = \sqrt{\frac{2DC}{IP}}$$

This is the 'ECONOMIC ORDER QUANTITY' expression.

Chapter 8

Stock Control In The Real World

In the previous chapter we considered the E.O.Q; i.e. the order quantity that would minimise our stock-holding and ordering costs. However, if you look closely at the total cost curve, you will notice that it has a fairly 'flat' bottom. This means that in terms of cost you can in fact purchase in numbers after the EOQ without increasing costs by a significant amount.

The E.O.Q model assumes that demand is absolutely constant; that replenishment of stock is instantaneous (i.e. at same moment as order is placed); and that the company therefore will never be out of stock. Now obviously none of the above assumptions hold true in the real world. It might therefore be useful to try and design a procedure that will SIMULATE stock behaviour so that a stock behaviour, ordering and safety stock model can be used to predict future patterns of demand, allowing truly cost-effective stock control and inventory reduction. The only assumptions made throughout this model are:

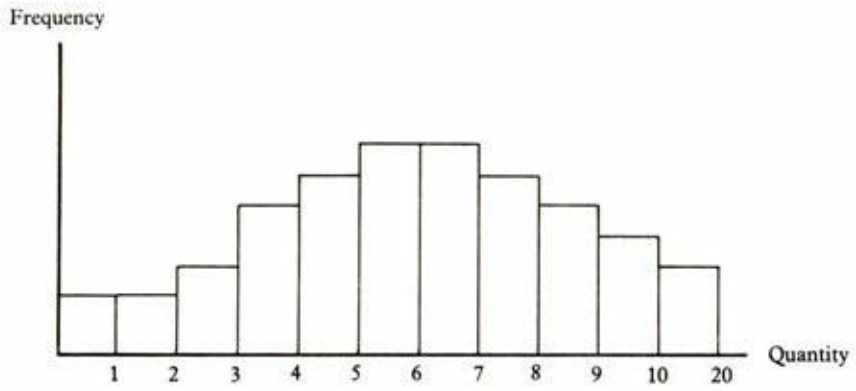
1) stock behaviour follows statistical rules;
2) the number of orders per week for the item is consistent.

Let's suppose that the company concerned is selling SNODS and that analysis of previous orders shows that:

6% of orders ask for 1 SNOD
8% of orders ask for 2 SNODS
10% of orders ask for 3 SNODS
12% of orders ask for 4 SNODS
14% of orders ask for 5 SNODS
14% of orders ask for 6 SNODS
12% of orders ask for 7 SNODS
10% of orders ask for 8 SNODS
6% of orders ask for 9 SNODS
4% of orders ask for 10 SNODS
4% of orders ask for 20 SNODS

(The last group of 4% take advantage of a price reduction for minimum order quantities of 20 SNODS).

Now if you were to construct a frequency distribution chart for the above items it would look like this:–



Frequency Distribution Chart For Orders

FIGURE 8.1

Now let's define a procedure called 'SALES' in which a variable QQ has been set to 50 numbers that represent the above percentages, for example the number 3 occurs five times, i.e. 10% of 50.

```
to sales
make "qq [1 1 1 2 2 2 2 3 3 3 3 3 4 4 4
4 4 4 5 5 5 5 5 5 5 6 6 6 6 6 6 6 7 7 7
 7 7 7 8 8 8 8 8 9 9 9 10 10 20 20]
end
```

Procedure SALES

FIGURE 8.2

Now if you type:

```
sales
pr :qq
```

you will get:

```
?sales
?pr :qq
1 1 1 2 2 2 2 3 3 3 3 3 4 4 4 4 4 4 5 5
5 5 5 5 5 6 6 6 6 6 6 6 7 7 7 7 7 7 8 8
 8 8 8 9 9 9 10 10 20 20
```

The Contents Of QQ

FIGURE 8.3

Now try this:

```
pr shuffle :qq
```

SHUFFLE (shuffle)
This will output a previously defined list in a random order.

This will result in something that looks approximately like this:

```
?pr shuffle :qq
5 6 7 4 5 4 1 9 5 6 5 6 8 10 4 7 7 6 3 8
 8 7 3 5 9 7 20 8 4 3 2 6 9 2 7 8 4 5 3
 5 3 1 2 20 1 10 6 2 4 6
```

Shuffled QQ

FIGURE 8.4

Now assuming that the size of the order the will be random (i.e. unpredictable), you can say:

```
repeat 17 [make "xx item 1 + random 50 shuffle :qq
pr :xx]
```

You can see that you have repeated picking a random member from a shuffled list seventeen times (the usual number of orders received each week). In this way we can simulate a week's sales. Having done that you might end up with a list of numbers similar to the one shown below

```
?repeat 17 [make "xx item 1 + random 50
shuffle :qq pr :xx]
3
1
4
2
4
5
4
5
3
2
2
5
4
7
5
4
2
```

A Week's Orders Simulated

FIGURE 8.5

Now to simulate say 6 weeks trading, try saying:

```
repeat 100 [make "xx item 1 + random 50 shuffle :qq pr
:xx]
```

You will see that the simulation is fairly good.

Let's have some more information about the SNODS we are stocking and selling:

SNODS INFO

COST PRICE £6
HOLDING COST 12%
ORDERING COST £25 each order
ANNUAL DEMAND (average) 4900

Now armed with this information and using the E.O.Q formula

$$EOQ = \sqrt{\frac{2\,CD}{IP}}$$

We can calculate the EOQ for SNODS.

This works out to be about 583 SNODS.

Now if we divide the annual usage (4900) by the EOQ we will see that orders for SNODS should be placed 8.4 times a year, i.e. about every 6 weeks.

Thus we can say that:

'We will order 600 SNODS every 6 weeks'

Now let's define a few procedures to SIMULATE a year's trading in SNODS.

Consider the 'TRY' procedure shown below.

```
to try
make "xx 600
repeat 102 [decrease]
end
```

Procedure TRY

FIGURE 8.6

Here you can see a variable XX set to 600 which in the EOQ; and a procedure called 'DECREASE' repeated 102 times: 6 weeks times 17 orders per week.

```
to decrease
pu sety :xx / 5 seth 90 pd fd 0.5
make "zz item 1 + random 50 shuffle :qq
rt 90 fd :zz / 5
make "xx :xx - :zz
end
```

Procedure DECREASE

FIGURE 8.7

In the DECREASE procedure shown above, the turtle is at a vertical position equivalent to XX (600), divided by 5 to get the curve on the screen. It will move to the right 0.5 units for every order and will move down the screen by an amount equivalent to a randomly picked order size.

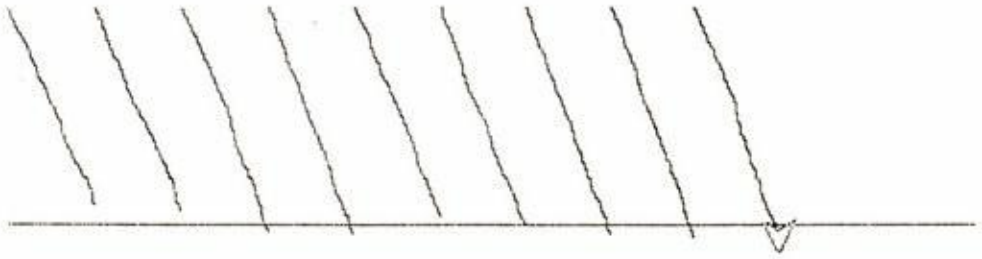
Having defined these two procedures, you could say:

```
cs
sales
try
```

and you will receive pictorial representation of a typical 6 weeks' trading in SNODS.

Now give the whole screen over to graphics display and say:

```
cs setx 300 set x -300 repeat 9 [try]
```



A Year's Trading Simulated

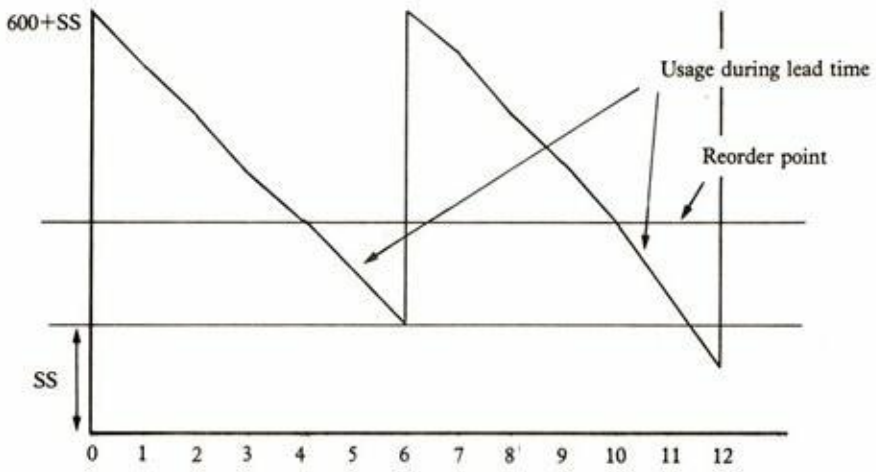
FIGURE 8.8

You can see from the above simulation that on several occasions throughout the year the company was out of stock of SNODS. (Count the number of orders below the horizontal line.)

Now suppose the Managing Directors issued a memorandum saying:

"We must never be in a position whereby we cannot supply SNODS, but we should maintain stock levels that minimise stock holding costs."

Added to this you also have the problem that SNODS take two weeks to be delivered to your company so your simulation must now look like this:



Trading Allowing For Delivery Times

FIGURE 8.9

This means that at the end of week 4, SNODS will be ordered and that during the next 2 weeks, SNODS will be used. At the end of week 6 a new stock of SNODS will arrive and the whole process will start again.

Now that point at which SNODS will be re-ordered will be after 4 weeks (68 orders from customers). If however, during the next 2 weeks (34 orders) the demand is greater than expected, a safety stock (SS) should fulfill customer needs.

The question now is:

How large should the safety stock be to

a) guarantee customer supply;
b) minimise stock holding costs.

Consider the SST (Safety Stock) and the 'TRY2' procedures shown below.

```
to sst
pr [WHAT IS THE STARTING STOCK LEVEL?]
make "ss1 item 1 rl
pr [WHAT IS THE SAFETY STOCK LEVEL?]
make "ss2 item 1 rl
setup
repeat 9 [try2]
end

to try2
make "xx :ss1 + :ss2
repeat 102 [decrease]
end
```

Procedures SST and TRY2

FIGURE 8.10

Now in answer to the first question, our answer will be 600, the EOQ. In answering the second question, well it's a matter of thinking of a number, trying a year's simulated sales of SNODS and seeing if an out of stock situation occurs.

The 'SETUP' procedure below simply draws the frame within which the simulation will take place with horizontal lines indicating zero, safety and maximum stock situations.

```
to setup
cs pu setx -300 pd fd 500 bk 500
setx 350 setx -300
fd :ss2 / 5 setx 400 setx -300
bk :ss2 / 5 fd :ss1 / 5 + :ss2 / 5 setx
400
setx -300 sety 0
end
```

Procedure SETUP

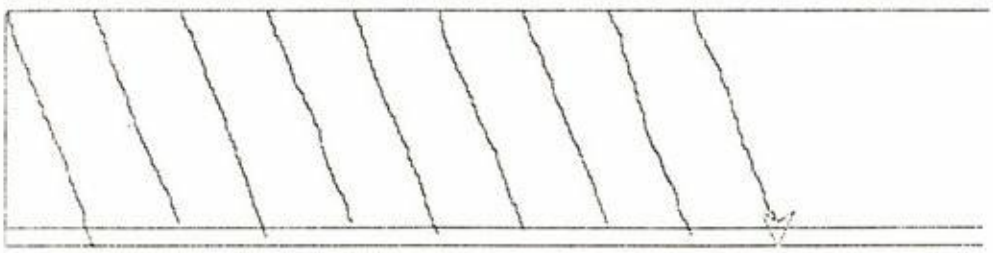
FIGURE 8.11

This is quite a straightforward procedure. Notice the numbers have been divided by 5, as in DECREASE, to allow the whole year's simulation to fit on the screen.

If you now type:

```
CS
SS
600
50
```

You will end up with a year's trading simulation similar to the one shown below in Figure 8.12



A Better Simulation

FIGURE 8.12

Thus you can see that if you order every 6 weeks in lots of 600 and maintain a safety stock of 50 you should almost never be out of stock.

Chapter 9

Linear Programming.

Let's imagine that you as a production manager of a manufacturing company are suddenly faced with a problem. As the production manager, one of your primary objectives, for better or worse, is to help maximise profits. Now as it happens a 'production window' has opened up in your manufacturing schedules and as a result you have some spare capacity in three production areas, e.g. machining, fabrication and assembly.

Imagine also that there are two products that require *only* the use of these three production facilities. Call these products ONE and TWO.

Now 1 unit of product ONE requires 5 hours of machine shop time, 7 hours of fabrication time and 14 hours in the assembly area. 1 unit of product TWO requires 12 hours in the machine shop, 7 hours in the fabrication area and 7 hours of machine shop time. Given these known times for the production of ONE and TWO and knowing the time available in each of the production areas, you as production manager should be able to decide how many of ONE and how many of TWO can be manufactured in the time available. First of all however it will be necessary to know the profits on the two products ONE and TWO.

Well let's assume that they are as follows:

Profit per unit on ONE = £10

Profit per unit on TWO = £20

How here are the time requirements again in a tabular form:

	Production Time Required		Time Available
	ONE	TWO	
Machining	5	12	600
Fabrication	7	7	560
Assembly	14	7	700

Now from the above table, you can see that the capacity available in each of the three production areas has been expressed in production hours available (600 machining, 560 fabrication and 700 assembly).

Now to try and answer a major question:

'How many ONE's and how many TWO's should I make?'

Well assuming that you can sell everything you make, you might say:

The most ONE's I can process through the machine shop in the time available is:

$$600 \div 5 = 120$$

The most ONE's I can put through the fabrication area is $560 \div 7 = 80$.

The most ONE's I can make in the assembly area is $700 \div 14 = 50$.

As you can see, the largest number of ONE's that can be processed through all three production areas is 50. If this were the chosen 'mix', i.e., 50 ONE's and no TWO's, the resultant profit would be:

$$50 \times 10 = £500$$

The same logic might also be applied to the production of TWO's, as follows.

The most TWO's I can process through the machine shop in the time available is:

$$600 \div 12 = 50$$

The most TWO's I can make in the fabrication area is:

$$560 \div 7 = 80$$

The most TWO's I can put through the assembly area is:

$$700 \div 7 = 100$$

Again there is a limit of 50 TWO's that can be made. However, if 50 TWO's were made, thus meaning that no ONE's could be made, the profit would be:

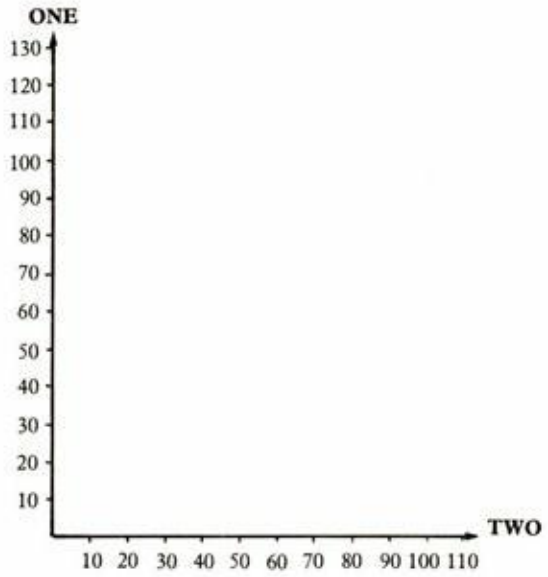
$$50 \times 20 = £1000$$

Now it is quite obvious that from a profit of view it would be better to make 50 TWO's for £1000 profit rather than 50 ONE's for £500 profit.

There is however still some doubt as to whether the manufacture of 50 TWO's gives the *highest* profit.

Now if the production manager were to plot the previously calculated figures, the optimum production mix of ONE's and TWO's could be read from the resulting diagram.

Consider the axes shown below:



Axes for a Linear Programming Graph

FIGURE 9.1

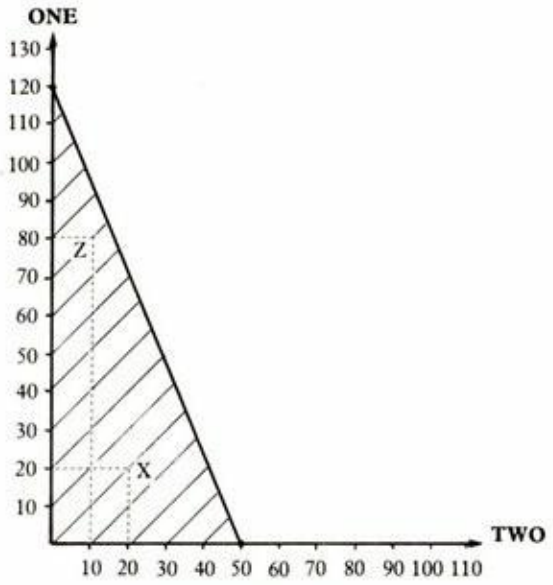
Now the vertical axis represents the number of product ONEs and the horizontal axis shows the number of product TWOs.

You will remember that if the production manager considers the machine shop, the can make either 120 ONE's or 50 TWO's.

If these two points are plotted and a line drawn between them, the diagram will look like the one shown in Figure 9.2.

Within the shaded area created by the line drawn between 120 ONE's and 50 TWO's, a variety of decisions can be made. For example at point X 20 ONE's and 20 TWO's might be put through the machine shop. Or at point Z, 80 ONE's and 10 TWO's might be produced. What you have done is to construct a **constraint line** which shows the limits on the choice of ONE's and TWO's that can be produced in the machine shop in the time available.

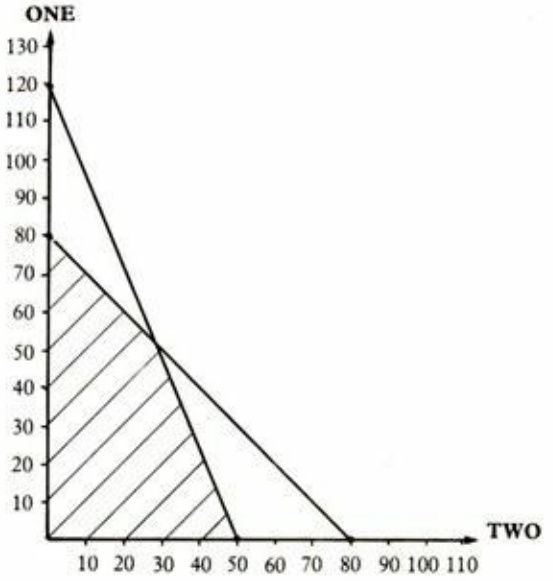
However, the production engineer will be constrained not only by machine–shop capacity but by the fabrication capacity and by the requirements of the two products.



Machine–Shop Production

FIGURE 9.2

Now you will remember that the most ONE's that can be produced is 80 and the most TWO's that can be produced in the fabrication areas is also 80. Thus, plotting the fabrication constraint, the diagram now looks like this:

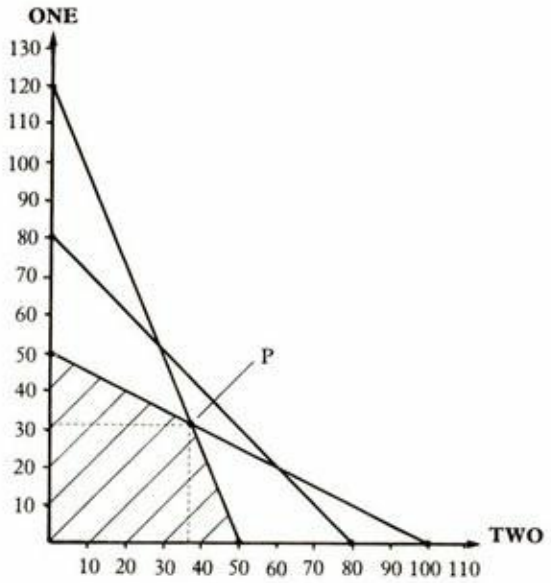


Adding the Fabrication Constraint

FIGURE 9.3

You will now see that the second constraint has reduced the number of choices available, e.g., 80 ONE's and 10 TWO's is no longer a viable choice because it doesn't fall within the shaded area; i.e. it does not satisfy both constraints.

Now for the third constraint, that of the assembly area. You will remember that either 50 ONE's or 100 TWO's can be processed. When you plot this constraint, the diagram will look like this:



Adding the Assembly Constraint

FIGURE 9.4

The introduction of the third constraint has reduced the choice even further and produced a shaded area known as the **feasible region**. This simply means that any mix of ONE's and TWO's picked from this region can be produced (i.e., fulfil the constraint requirements). You will note that it is possible, if we wanted to, to produce 50 TWO's and 0 ONE's thus making a profit of (50 × £20) £1000.

However, this might not be the best mix to maximise profit.

Why not consider point 'P'?

Well, it looks as though 36 TWOs can be produced and 32 ONEs.

Let's see what sort of profit this will bring about:

$32 \times £10 = £320$

$36 \times £20 = £720$

TOTAL £1040

Now check to see if 32 ONE's and 36 TWO's can be produced.

$(32 \times 5) + (36 \times 12) = 501$ hrs

$(32 \times 7) + (36 \times 7) = 476$ hrs

$(32 \times 14) + (36 \times 7) = 700$ hrs

So it looks as though there is capacity to spare in the machine shop and fabrication areas, but none to spare in assembly.

Now for some procedures to help with similar 'LINEAR PROGRAMMING' problems.

First of all, a small procedure to help draw the diagram axes.

```
to tiny
fd 5 rt 90 fd 2 bk 2 lt 90
end
```

Procedure TINY

FIGURE 9.5

The 'TINY' procedure will draw a small division of one of the diagram's axes.

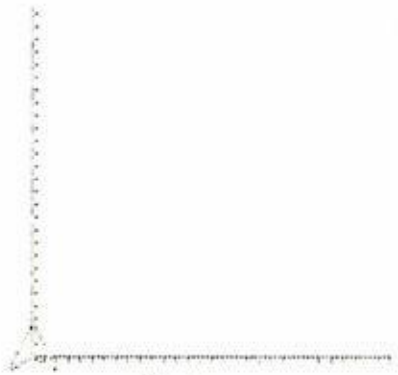
Now you could say:

```
to axis
cs repeat 30 [tiny]
pu setpos [0 0]
pd rt 90
repeat 30 [tiny]
setpos [0 0] seth 0
end
```

Procedure AXIS

FIGURE 9.6

If you define 'TINY' and 'AXIS', and call AXIS, a pair of axes should be drawn.



The Axes

FIGURE 9.7

Now for a procedure to collect sufficient information to allow Logo to perform the graphical solution to a problem.

```
to collect
pr (se "HOW "MANY "HOURS "AVAILABLE "IN
"AREA "A?)
make "a item 1 rl
pr (se "AND "IN "B?)
make "b item 1 rl
pr (se "AND "C?)
make "c item 1 rl
ct pr (se "NOW "THE "PRODUCTION "TIMES "
FOR "THE "TWO)
pr [PRODUCTS]
pr (se "HOW "MUCH "TIME "DOES "PRODUCT "
ONE "NEED)
pr (se "IN "AREA "A?)
make "onea item 1 rl
pr (se "AND "IN "AREA "B?)
make "oneb item 1 rl
pr (se "AND "IN "C?)
make "onec item 1 rl
ct pr (se "HOW "ABOUT "THE "TIME "FOR "P
RODUCT "TWO)
pr (se "IN "AREA "A?)
make "twoa item 1 rl
pr (se "NOW "FOR "PRODUCT "TWO "IN "B?)
make "twob item 1 rl
pr (se "AND "IN "C?)
make "twoc item 1 rl
end
```

Procedure COLLECT

FIGURE 9.8

You will notice that this is quite a straightforward procedure to collect information from the user and allocate it to various variables.

The production areas of machining, fabrication and assembly are referred to as areas 'A', 'B' and 'C'.

Using the times from the previous example and responding to the 'COLLECT' procedure correctly you will have set the variables in the following way:

a is 600
b is 560
c is 700
onea is 5
oneb is 7
onec is 14
twoa is 12
twob is 7
twoc is 7

and these relate in matrix form like this:

ONE	TWO	
5 (onea)	12 (twoa)	600 (a)
7 (oneb)	7 (twob)	560 (b)
14 (onec)	7 (twoc)	700 (c)

Now to represent this data as a series of constraints, thus enabling a solution to this problem via linear programming. Consider the procedure 'DRAW' shown below.

```
to draw
axis
sety :a / :onea
setpos (se :a / :twoa 0)
setpos [0 0]
sety :b / :oneb
setpos (se :b / :twob 0)
setpos [0 0]
sety :c / :onec
setpos (se :c / :twoc 0)
setpos [0 0]
end
```

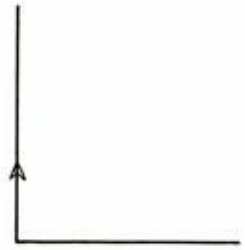
Procedure DRAW

FIGURE 9.9

Now the first line calls 'AXIS' and draws the necessary frame within which the constraints are to be plotted.

In the second line:

```
sety :a / :onea
```



The turtle will move to a position along the y axis equal to 600 ÷ 5 (120).

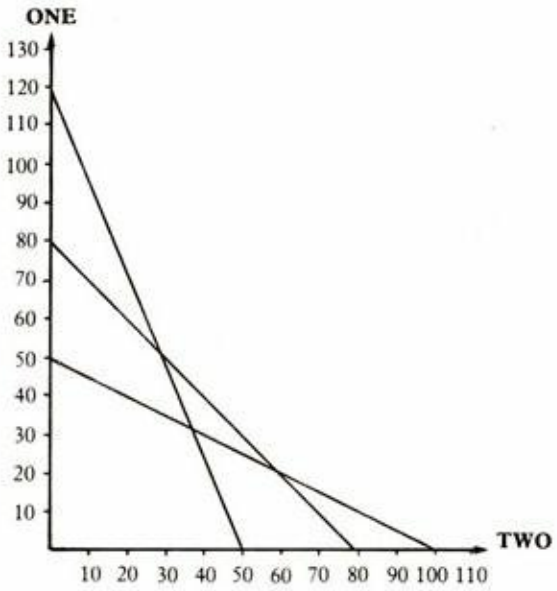
In the third line:

```
setpos (se :a / :twoa 0)
```

The turtle will move in pendown mode to a position along the x axis equal to 600 ÷ :12 (50).

Thus the first constraint has been constructed.

The second and third constraints are illustrated similarly and having called 'DRAW', your screen will look like this:



The Final Result

FIGURE 9–10

Now you might just be able to read off the values for products 'ONE' and 'TWO' but just to check that the procedures work correctly type:

```
rt 90 fd 36 lt 90 fd 32
lt 90 fd 36 ht
```

which confirms the correctness of the procedures.

You could of course introduce more constraints if you wanted to – as perhaps your first steps on your own into the world of Logo.

Appendix 1

Loading Logo on the 8256 and 8512

Before you can load Logo, you first of all have to load CP/M. To do this, you should carry out the following steps.

1) If your computer is switched on, reset it by removing any disc from the drive(s), then tap the EXIT key whilst holding down SHIFT and EXTRA. Then release SHIFT and EXTRA. If instead your computer is switched off, make sure there is no disc in the drive(s), then switch it on. The screen should be bright green.
2) Find side 2 of the system discs, and place the disc in the drive (or topmost drive if you have two) with that side facing left. There should be no need to force the disc in, it should slide in easily and click into place. If it doesn't, you are putting it in incorrectly.
3) Tap the space bar once. After much whirring of the disc drive, the screen will come up with something like this:

   PCW8256 CP/M Plus Amstrad Consumer Electronics plc

   v 1.0, 61K TPA, 1 disc drive, 112K drive M:

   A>

4) When the 'A>' prompt appears, CP/M is loaded.

Having loaded CP/M, the next job is to tell CP/M to load Logo.

1) Remove the CP/M system disc from the disc drive.
2) Place the other system disc into the drive, with side 4 to the left.
3) Type

```
submit logo
```

4) Logo will now load. The screen will come up with the message shown on page 12 as it is loading, so you can now turn back to page 12.

Appendix 2

Editing Procedures on the 8256 and 8512

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 square in a 'SQUARE' procedure.

In the event that this particular procedure is not in your computer, type it in as below:

```
to square
fd 5Ø
lt 9Ø
fd 25
lt 9Ø
fd 5Ø
lt 9Ø
fd 25
lt 9Ø
end
```

Now type 'cs square' and run the procedure.

Having done this, your screen might look something like the one shown below in Figure A2.1.

SQUARE Procedure

FIGURE A2.1

Let's now have a look at the method provided in 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.

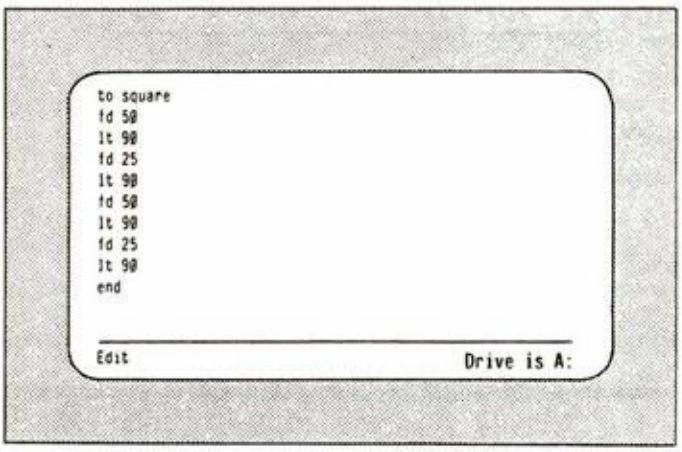
Now type 'square' and run the procedure. Having done this, next type:

```
ed "square
```

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

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 A2.2



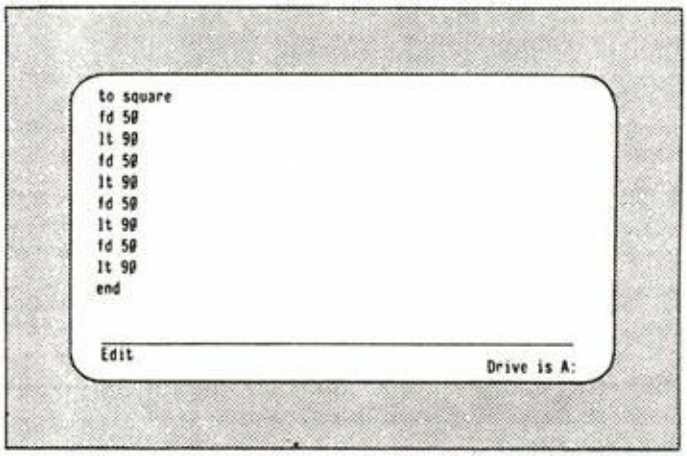
The Edit Screen

FIGURE A2.2

Let's assume that you want to change the distances in the 'fd 25' commands. The first thing to do is move the cursor to the first line to be changed. To move the editing cursor you can use the arrow keys on the right of the keyboard. Press the down–arrow key three times (or ALT–N if you prefer) and this will move the cursor down (don't hit RETURN 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 RETURN YET

You can now bring the cursor down to the end of the next 'fd 25' command by use of the down–arrow key (or ALT–N). Then move it to the left one place by hitting the left–arrow key (or ALT–B) 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 (or ALT–F) 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 A2.3.



The Edited Procedure

FIGURE A2.3

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

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

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 "square
```

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 RETURN.

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 STOP key.

You will now receive a message saying:

```
Stopped!
```

Now type

```
ed "square
```

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

```
RETURN
```

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

```
EXIT
```

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

```
ed "square
```

Now there is the procedure; let's imagine that the 'ht' command is no longer necessary. DR Logo uses various ALT 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.

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

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

ALT - H will act like the ← DELete key and remove the character to the left of the cursor.

ALT - D deletes the character covered by the cursor.

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

ALT - B moves the cursor back.

ALT - F moves the cursor forwards.

ALT - N moves the cursor down a line.

ALT - P moves the cursor up a line.

ALT - C ends the editing and re–defines the procedure just like the 'EXIT' key.

Note that on your particular model of computer, you may find that some of these control keys will only work in EDIT mode. Also, you may not be able to use the arrow keys to move the cursor, or use the EXIT key to complete the edit.

Well, if you can, 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 (else ALT–N) 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 ALT–C or EXIT 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	Enables repetition of whatever is in [command list]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.
ALT – A	moves to line start.
ALT – E	moves to line end.
ALT – H	removes the character and moves to the left.
ALT – D	deletes the character below the cursor.
ALT – O	opens up a space in the procedure.
ALT – N	moves the cursor down a line.
ALT – B	moves the cursor back.
ALT – F	moves the cursor forwards.
ALT – P	moves the cursor up a line.
ALT – C	ends editing session.
EXIT	ends editing session as well.
STOP	used to leave edit mode.

Now return to page 26.

Appendix 3

Formatting Discs on the 8256 and 8512

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 this method.

1) Reset the computer by pressing SHIFT, EXTRA and EXIT simultaneously. Then load CP/M in the usual way (see the first part of Appendix 1 if you're not too sure about this).
2) When you see the message 'A>', type:

```
diskit
```

and this message will be displayed after a short pause:

```
Please remove the disc from the drive.
Press any key to continue.
```

3) Do as it says!
4) The display will change to show a menu. Press f3.
5) A new message will be displayed. Place the disc to be formatted into the drive (topmost drive if you have more than one) with the side to be formatted facing left. Make sure that it is not write protected – the little hole in the top left corner of the disc should be covered by the little plastic tab. Remember also that formatting a disc destroys all information currently stored on it, so make sure you use a disc that has nothing important on it.
6) Press the Y key when you are ready. The screen will clear and the computer will count through the 0–39 tracks as it formats the disc.
7) When it has finished, remove the disc and tap the space bar. You can then eithr format another disc (or the other side of the current one) or return to the other menu. Assuming you don't press Y, but some other key (e.g space), then press EXIT.
8) Place the system disc with side 4 facing left into the drive and type

```
submit logo
```

Logo will now re–load. Go back to 'Having formatted your disc...' on page 28.

Index

PRACTICAL LOGO on the AMSTRAD

This book is about practical applications of Logo. Rather than using the more usual graphics-based approach, it shows how Logo can be used to write programs on serious topics such as decision making, simple investment analysis, sales forecasting and stock control.

The book moves quickly from a brief introduction to Logo for those who are not familiar with the language, to various case study areas for which procedures are developed in later chapters.

The approach of this book is exploratory; it is an attempt to broaden the Logo user's horizons well beyond simple graphics applications.

GLENTOP
PUBLISHERS LIMITED

ISBN 1-85181-046-3

Glentop Press Ltd
Standfast House
Bath Place
Barnet
Herts EN5 5XE

£6-95

9 781851 810468