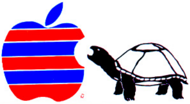
LEARNING LOGO ON THE APPLE II

ANNE McDOUGALL
TONY ADAMS
PAULINE ADAMS



G F Brand
55 River Road West
LANE COVE NSW 2066

LEARNING LOGO ON THE APPLE II

LEARNING LOGO on the APPLE II

ANNE McDOUGALL
Faculty of Education, Monash University

TONY ADAMS
Department of Computing, Royal Melbourne Institute of Technology Ltd

PAULINE ADAMS
Binnak Park Kindergarten

PHA

PRENTICE-HALL OF AUSTRALIA

A Prentice-Hall Direct Edition

Prentice-Hall of Australia Pty Ltd, Sydney
Prentice-Hall International Inc., London
Prentice-Hall Canada Inc., Toronto
Prentice-Hall of India Private Ltd, New Delhi
Prentice-Hall of Japan Inc., Tokyo
Prentice-Hall of Southeast Asia Pte Ltd, Singapore
Editora Prentice-Hall da Brasil LTDA., Rio de Janeiro
Whitehall Books Ltd, Wellington
Prentice-Hall Inc., Englewood Cliffs, New Jersey

Printed in Australia by Globe Press Pty Ltd.

2 3 4 5 87 86 85 84 83

National Library of Australia
Cataloguing-in-Publication Data

McDougall, Anne, 1945- .
Learning LOGO on the Apple II

Bibliography.
Includes index.
ISBN 0 7248 0732 2

1. Apple computer - Programming.
2. LOGO (Computer program language).
I. Adams, Tony, 1944- . II. Adams, Pauline, 1944- . III. Title

001.64'24

Logo was initially developed in 1968 as part of a National Science Foundation sponsored research project conducted at Bolt, Beranck & Newman, Inc., in Cambridge, Massachusetts

For Christopher, Gareth, Kirsty and Rosalind

PREFACE

This book is intended to introduce the LOGO computer programming language to computing novices.

LOGO is a programming language for students. The more advanced parts of LOGO embody many of the challenging concepts of modern computer science; however the language is so designed that the beginner can also undertake many interesting and worthwhile projects. Our intention in this book is not to provide a complete description of LOGO, but to introduce its major features at a simple non-technical level. For readers who wish to learn more about the language, a bibliography is included in Appendix H.

The book uses both the versions of LOGO currently available for the Apple II computer, MIT LOGO and Apple LOGO.

Many people have helped us in writing this book. We acknowledge an intellectual debt to Seymour Papert, Harold Abelson, Cynthia Solomon, Jim Howe, Mike Sharples, Sandra Wills and others involved in research and development work with LOGO. Jim Howe's comments on early drafts of some chapters have influenced our methods of presentation throughout. David Squires, Sandra Wills, Margaret Wilson, Gillian Barclay, Jeff Richardson and Kathy Webb read later drafts and offered helpful criticism. Many of the designs and projects in the book have been developed from the LOGO work of Charles Nevile, Suzanne Milburn, Elizabeth Swann, and Christopher and Gareth Adams. The photographs were taken by Lee Adams. Technical support has been provided throughout our LOGO work by Gordon Perkins. We would also like to acknowledge the energy and patience of Chris Nicol and Wayne Cosshall in editing and production of the book.

We appreciate the forebearance of our families, friends, colleagues and students, and especially Derek McDougall, throughout the project. Finally we wish to note the reliability of the Apple hardware, and the robustness of the Zardax word processing system with which the book was written.

Anne McDougall
Tony Adams
Pauline Adams

CONTENTS

BEFORE YOU BEGIN

LOGO is a computer language for solving problems, learning things, doing projects, experimenting and playing.

The computer will do exactly what you command it to do. At times this might not be quite what you expected it to do. Don't panic; you are in charge. Unexpected results are a way of learning. Typing an erroneous command might produce an error message from the computer; it cannot damage the computer. Usually you just need to type the correct command next, and you can forget the wrong one.

This book is written in such a way that at times you might be typing into the computer commands that you only partly understand. It is possible to use LOGO procedures without completely understanding their details. In fact experimenting with these pre-written procedures is a good way of learning just how they work.

The book is organized in the following way.

Chapters 1 and 2 provide the basic tools for using LOGO on the Apple computer. In these chapters the emphasis is on doing things rather than understanding; complete understanding will come later.

Chapters 3 and 4 are made up of LOGO projects using the tools introduced in the first two chapters. Some new ideas are introduced, but these chapters mainly develop and consolidate earlier material.

Chapter 5 is largely about mathematics. Scan this chapter; read it more closely if you are interested in mathematics.

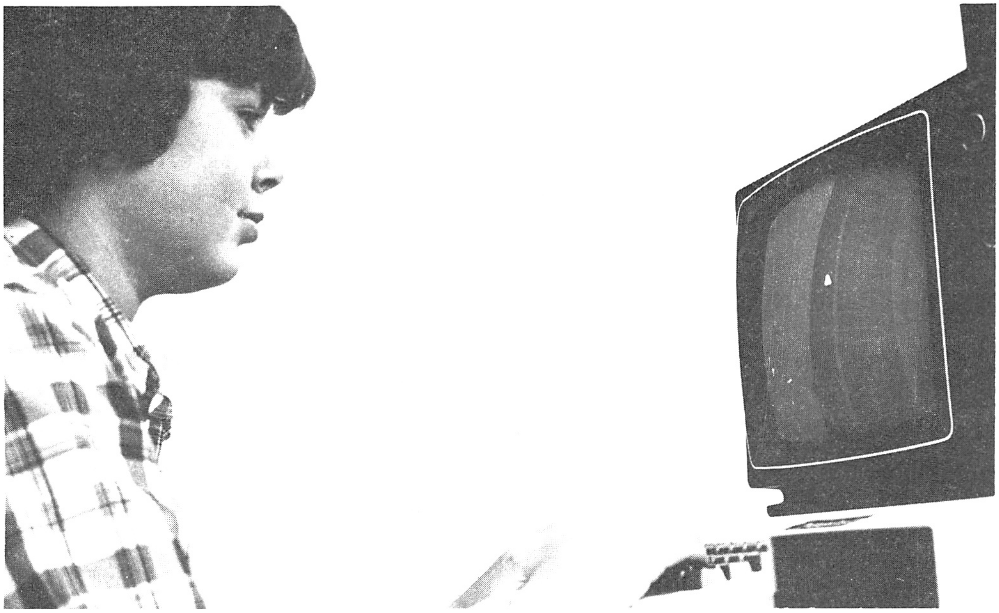
Chapter 6 is probably the most difficult; it introduces the concepts of text processing in LOGO. Even if you haven't completely understood the material in Chapter 6, you will still be able to use the procedures in Chapters 7 and 8 to manipulate English language text.

Each chapter is followed by a summary of ideas and LOGO commands introduced in that chapter. At the end of the book, in Appendix A, is a complete summary of LOGO commands introduced in the book, for reference.

Appendix F discusses details of how to save procedures on a diskette, and the use of a printer connected to the Apple. This Appendix might best be read after Chapter 2.

1 STARTING UP THE TURTLE

SETTING UP THE SYSTEM



You will need:

a 64K Apple II computer,
a video monitor,
a disk drive,
a LOGO language diskette.

A printer interfaced to the computer and a spare blank diskette will enable you to make printed copies of LOGO procedures and to save them for later use.

TWO DIFFERENT LOGOS

Two different versions of LOGO are available for the Apple.

MIT LOGO was developed at the Massachusetts Institute of Technology. This version is marketed by two companies; Terrapin Inc. of Boston, and Krell of New York. As far as the definition of the LOGO language is concerned, both of these are the same. When we refer to MIT LOGO we will mean either of these.

APPLE LOGO is the version of LOGO developed by Logo Computer Systems Inc. and marketed by Apple Computer Inc. as an official Apple product. We will refer to this version as APPLE LOGO.

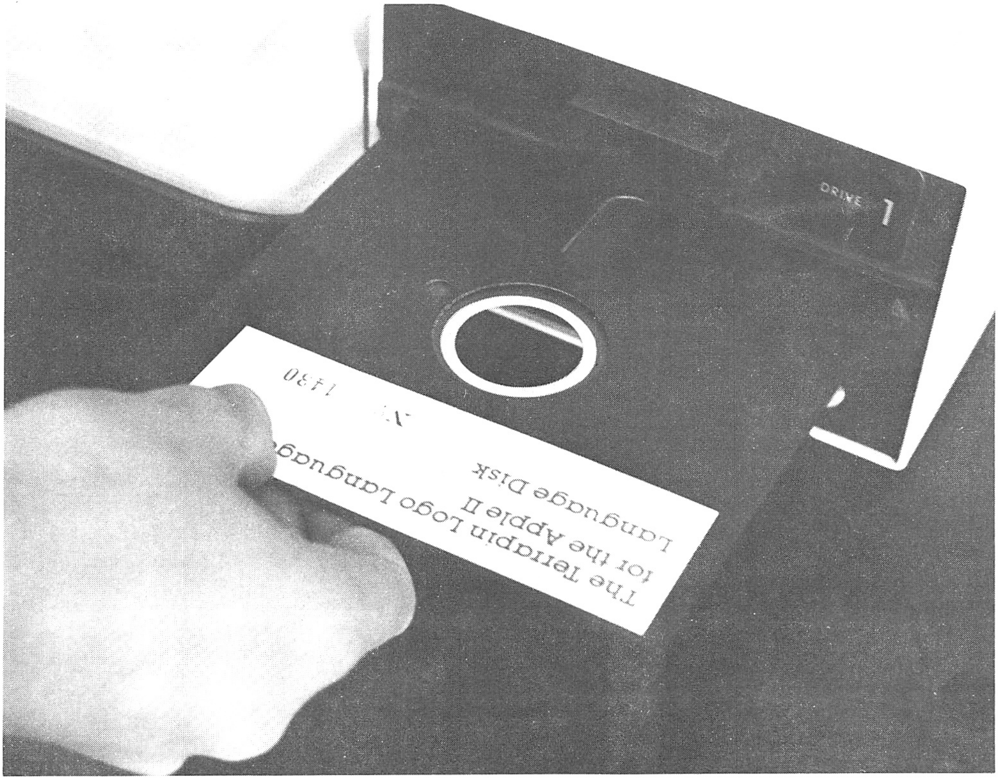
The MIT and Apple versions of LOGO are substantially the same, but they have some differences. When we show only one program or command, it will operate with both versions. When we show two programs or statements side by side, the left hand one will be MIT LOGO and the right hand one will be APPLE LOGO.

Remember if you have Terrapin or Krell LOGO that we refer to these as MIT LOGO. If you have LOGO from Logo Computer Systems Inc., remember that we refer to this as APPLE LOGO.

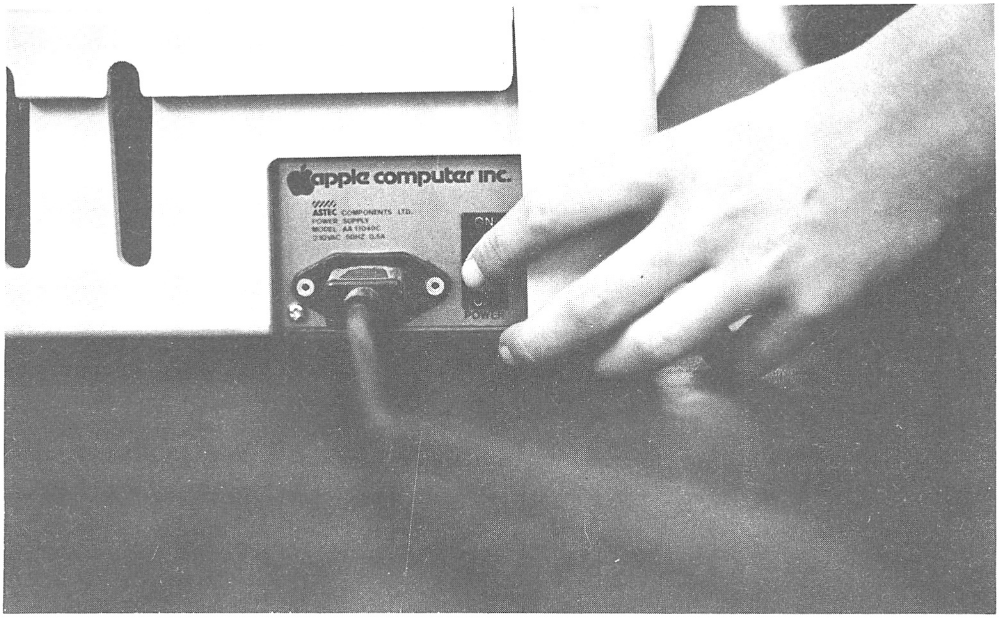
GETTING STARTED

Full details about the use of the Apple computer are provided in operating manuals. We will cover just what you need to use LOGO.

The computer and the video monitor must both be plugged into power points, and the power switched on. Don't switch the computer itself on yet. Insert the LOGO language diskette into the disk drive as shown in the photograph. Make sure the diskette label is facing upwards, and the diskette has the orientation shown. Remember to close the door of the disk drive.



Switch on the monitor and the computer. The computer's on/off switch is at the back on the left hand side.



You will have to wait a few moments while the disk drive hums and the LOGO program is loaded into the computer. During this time, if you are using APPLE LOGO the following message will appear:

```
PRESS THE RETURN KEY TO BEGIN
```

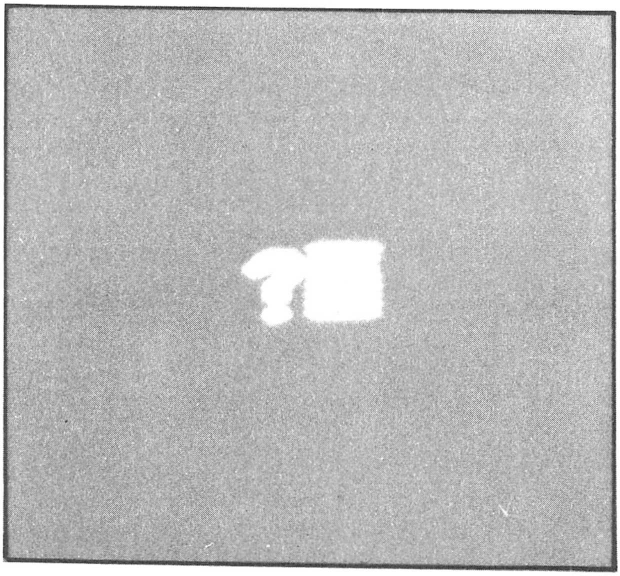
With this there will be a message telling you to insert your own diskette. Ignore these messages; just press the RETURN key. The disk drive will hum some more.

Copyright messages will appear during the loading time in both MIT and APPLE LOGO. When loading is complete the following message will appear:

```
WELCOME TO LOGO
?
```

If this sequence of steps has not proceeded as described, you should start again. Check particularly that everything is switched on. If the LOGO program will still not load, you should consult someone who is familiar with the Apple computer.

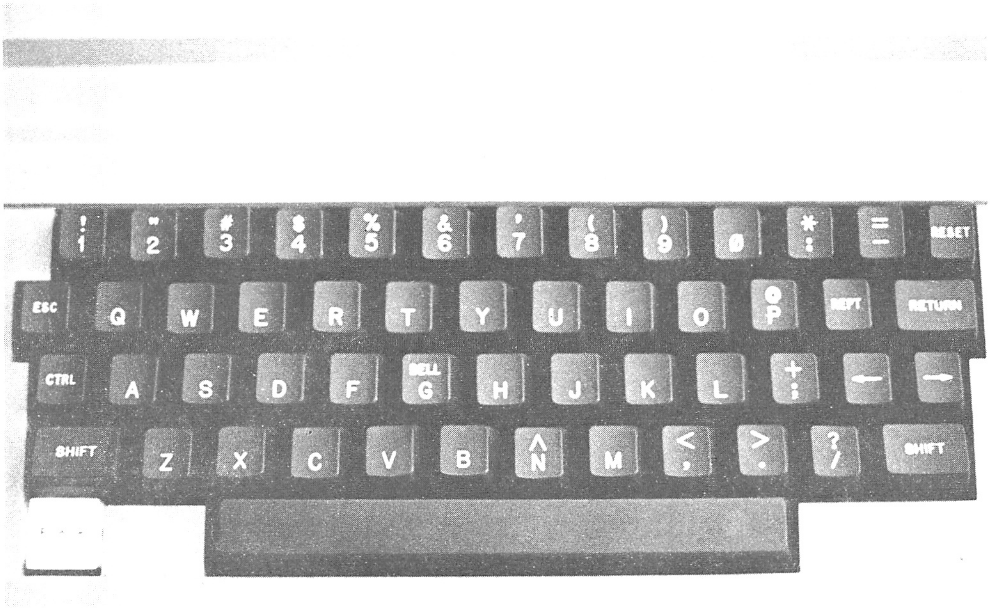
The question mark at the left of the screen is called a prompt. It indicates that the LOGO system is ready for you to type a command.



Next to the ? you will see a flashing square. This is called the cursor. It can move across the screen to indicate which position the character to be typed will occupy. When the ? and cursor appear together at the beginning of a line, the computer is ready for you to type a new command.

Once the LOGO language is loaded into the computer, the LOGO diskette should be removed and returned to its packet. It should be stored away from magnetic fields and dust, so do not leave it lying on the monitor or the disk drive, or where it is exposed to dust.

THE KEYBOARD



You will notice that the keyboard resembles the keyboard of a typewriter. There are some additional keys; these will be described when they are needed later.

Unlike a typewriter, on the computer keyboard the SHIFT key is not needed for typing upper case letters. However it is needed to type special characters such as ", +, = and so on.

The Space Bar:

Note that the space bar is nearest to you. Spaces in LOGO commands are very important. Until you understand the language well, take care to type spaces exactly as they are indicated.

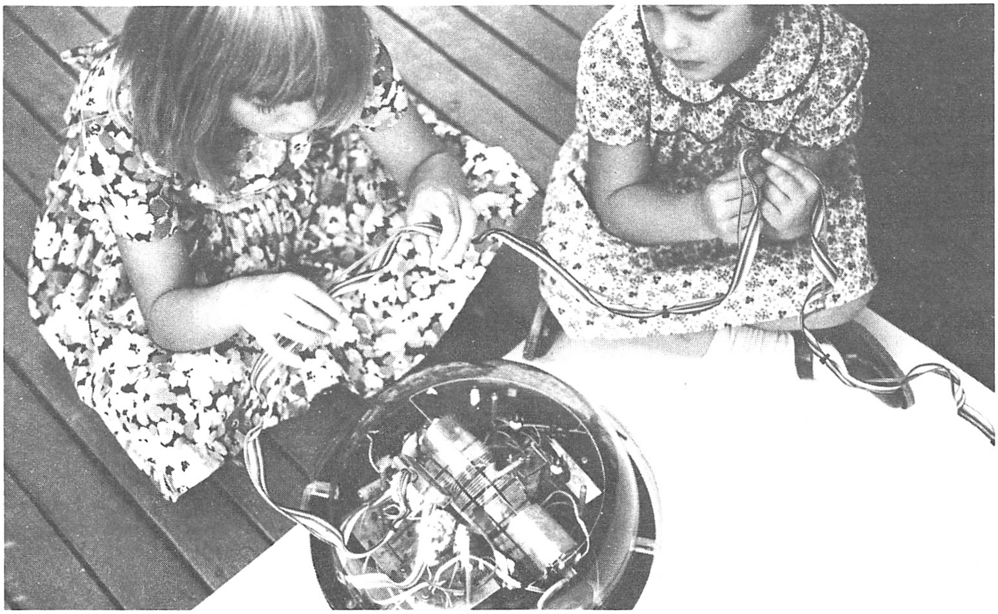
The RETURN Key:

The RETURN key is located on the right of the keyboard. As you type each command, the command will appear on the screen. However it is not actually sent to the computer until you press the RETURN key, so you MUST do this at the end of EACH command.

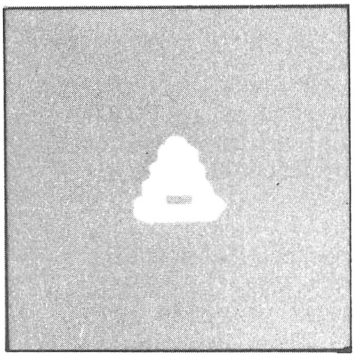
LOGO AND TURTLES

The LOGO programming language can be used to command a robot, called a turtle. The commands are movement instructions, such as go FORWARD (abbreviated FD) and turn LEFT (abbreviated LT). The turtle can leave a trail as it moves, and so produce drawings. Turtles come in various kinds.

A robot turtle is a robot on wheels (see the photograph below). It has a felt-tip pen underneath. It moves about on a large piece of paper on the floor, and can draw its path as it goes. A robot turtle usually has as well a pair of little lamps (near the front, positioned like eyes), a small speaker which can produce a tooting sound, and touch sensors around its rim.



Another kind of turtle is the screen turtle, which moves about the screen of a computer. Like the robot turtle, it can draw its path as it moves, though it moves much more quickly than a robot turtle can.



Now let us see what the screen turtle can do.

DRAWING A SQUARE

We are going to prepare a procedure for drawing a square.

Before we write any new procedure we must give it a title. The title can be any word or letter, but it is a good idea to use descriptive titles as these are easier to remember. Different procedures must have different titles. We shall call our square-drawing procedure SQ. Type

```
TO SQ
```

this stands for
TO draw a SQuare

Remember to leave spaces where we have left spaces (for example, between TO and SQ). Remember also to press RETURN at the end of the command.

The next statement to type is REPEAT 4 [FD 50 LT 90] but don't start it without reading the steps below.

1. Type REPEAT 4 (don't use the REPT key, and don't press RETURN yet).

2. Type a space after the 4.

3. Now you need a square bracket. Square brackets do not appear on the keyboard. To type a [you need to hold down the SHIFT key and press N.

4. Type FD 50 LT 90 (don't forget the spaces; be sure to use the numeral 0 and not the letter O).

5. To type a] hold down the SHIFT key and press M.

6. Press RETURN.

On the next line type

```
END
```

and press RETURN.

The following should be on the screen:

```
TO SQ
REPEAT 4 [FD 50 LT 90]
END
```

CORRECTING TYPING ERRORS

If you notice a typing mistake in a command before you have pressed RETURN, you can delete the incorrect characters. To delete the character to the left of the cursor you should press the key indicated below. Continue to press the key until the error is deleted and then re-type the correct characters.

MIT LOGO — The ESC key

APPLE LOGO — <— (left arrow key)

If you notice an error after RETURN has been pressed, type CTRL-G by holding down the CTRL key and pressing G. Then start again from the beginning of the procedure.

Correct any mistakes and proceed.

If you are using MIT LOGO, you must now type CTRL-C, by holding down the CTRL key and pressing C.

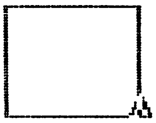
You have now given the computer a sequence of instructions it will obey when you type SQ. This is called a procedure. On the screen will appear

```
SQ DEFINED
?
```

Next to the ? type

```
SQ
```

and press the RETURN key. Now the procedure will be used (or executed) and the turtle should draw a square with side length of 50 turtle steps. The following should appear on the screen:



The square might be slightly distorted or flattened, as in the illustration above. The amount of flattening depends on the monitor screen you are using. A way of fixing this is shown in Chapter 4. Note the position of the turtle after SQ is finished. It is in the centre of the screen (called the home position) facing up.

If the turtle didn't draw a square for you, you can probably now see a message on your screen which says something like

MIT LOGO	APPLE LOGO
THERE IS NO PROCEDURE NAMED FD50	I DON'T KNOW HOW TO FD50

A typing error must have slipped through unnoticed. Type the following:

MIT LOGO	APPLE LOGO
ERASE SQ	ERASE "SQ

and press RETURN. Although the screen is not cleared, the procedure SQ is erased from the computer's memory.

Now type the procedure from the beginning again. (Later we will show you how to correct procedures without completely re-typing them.)

CLEARING THE SCREEN

Before you make further drawings you will probably want to clear the screen and return the turtle to the centre. The command which gets things ready for a new drawing is

MIT LOGO	APPLE LOGO
DRAW	CLEARSCREEN (abbreviated CS)

Type this command now. Remember to press the RETURN key.

Now type

```
SQ
```

(Did you remember RETURN?) The turtle should draw the square again.

Clear the screen by typing the appropriate command.

DRAWING A TRIANGLE

Let's make a triangle-drawing procedure. Type

```
TO TR
REPEAT 3 [FD 50 LT 120]
END
```

Use SHIFT-N and SHIFT-M for the square brackets. Remember to press RETURN at the end of each command.

If you are using MIT LOGO don't forget to press CTRL-C after END.

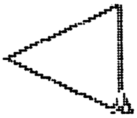
When the following appears

```
TR DEFINED
?
```

type

```
TR
```

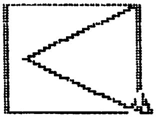
and watch your triangle come to life. The computer should respond



Notice again the flattening effect which makes the vertical side of the triangle appear slightly shorter than the other sides. If an error message appears, erase the TR procedure in the way you were shown earlier, and type it again.

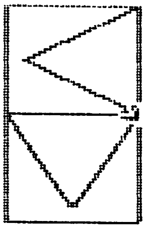
MAKING A PATTERN

Don't clear the screen. Leave the triangle there and type SQ. The computer should respond



Let's turn the turtle and make another square with a triangle inside it. Type (without clearing the screen first)

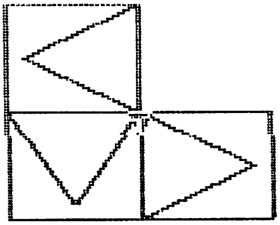
```
LT 90
SQ
TR
```



You might like to do this again. Type

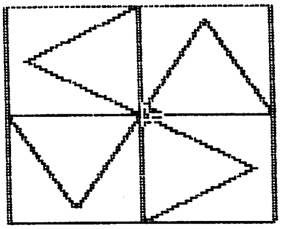
```
LT 90
SQ
TR
```

The computer should show



How many times can you do this before you are drawing entirely over old lines? Try it once more.

```
LT 90
SQ
TR
```



If your drawings go much below the centre of the screen, they will disappear into the text that appears at the bottom. Typing

```
FULLSCREEN
```

causes the text to disappear so the full screen can be used for drawing.

To get the text back, type

```
SPLITSCREEN
```

You can use these two commands at any time while a drawing is on the screen.

BIGGER SQUARES AND TRIANGLES

You could write new procedures changing the size of the square and the triangle.

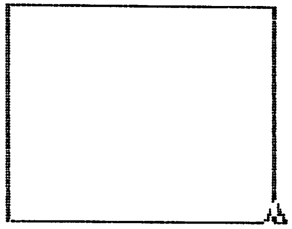
For a bigger square, type

```
TO SQ1
REPEAT 4 [FD 100 LT 90]
END
```

(If you are using MIT LOGO remember to press CTRL-C afterwards.) Then type

```
SQ1
```

The computer should respond



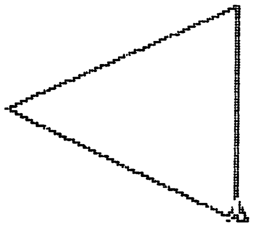
Now make a bigger triangle by typing

```
TO TR1
REPEAT 3 [FD 100 LT 120]
END
```

Type

```
TR1
```

The computer should respond



EXERCISE

1.1 Draw more squares and triangles of different sizes.

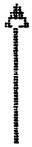
THE TURTLE STATE

We have written some procedures and seen how they work. Now we will go back and look at the individual commands that make up these procedures. Later we will see how these commands can be used by themselves outside procedures.

What is important about a turtle is its state. A turtle's state is determined by two things: its position (i.e. where it is), and its heading (i.e. the direction in which it is facing).

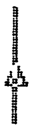
You can change the state of a turtle by changing either its position or its heading, or both. Its position is changed by LOGO commands such as FORWARD. Clear the screen and type

FORWARD 50 (abbreviated FD 50)



This causes the turtle to move forward 50 turtle steps (turtle steps are quite tiny). Or type

BACK 35 (abbreviated BK 35)

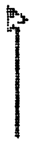


This causes it to move backward 35 turtle steps. Notice the position of the turtle.

The turtle's heading can be changed using LOGO commands such as LEFT and RIGHT.

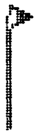
Now clear the screen and type

```
FD 50
LEFT 30        (abbreviated LT 30)
```



This causes the turtle to turn to its left on the spot, through an angle of 30 degrees (if you are not sure about degrees, see below). Now clear the screen and type

```
FD 50
RIGHT 90       (abbreviated RT 90)
```



This causes it to turn to its right through an angle of 90 degrees.

Degrees are used to measure angles or the amount of turning. Imagine yourself standing up, still facing the computer, and then turning right around once to face the computer again. You would turn through, or change your heading by, 360 degrees in making such a turn. Turning yourself - or the turtle - half way round, to face the direction opposite to where you started, would be changing the heading by 180 degrees. A quarter turn is 90 degrees, and so on.

EXERCISE

1.2 Clear the screen, and try drawing a square using the BACK (abbreviated BK) and RIGHT (abbreviated RT) commands.

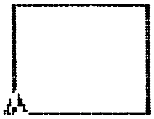
LOGO MODES

We have written some procedures and seen how they work. Let us now examine the two modes in which LOGO operates.

When the ? appears on the screen, we are in immediate mode. We can type commands such as FD and BK and they will be executed immediately. Clear the screen and type

```
REPEAT 4 [FD 50 RT 90]
```

Press RETURN, and this should be executed straight away. The computer should respond



Now type

```
TO SQ2
```

and press RETURN. The TO command is executed immediately. As a result, the computer leaves the immediate mode of operation and enters the defining mode. In the defining mode, you can type commands and they are not executed immediately. Note also that the question mark prompt is no longer there. Type

```
REPEAT 4 [FD 50 RT 90]
```

The REPEAT command is not executed. Now type

```
END
```

If you are using MIT LOGO press CTRL-C. The following should appear on the screen.

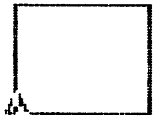
```
SQ2 DEFINED
?
```

You have left the defining mode and returned to immediate mode.

Notice the ? is back again. If you type

```
SQ2  (don't forget RETURN)
```

the commands within SQ2 will be executed immediately, and the computer should respond

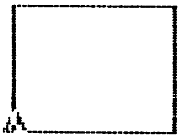


Remember when you use the TO command you enter a mode where you define a procedure for later use.

A SQUARE IN IMMEDIATE MODE

We can draw a square step by step in immediate mode, and watch the turtle turn and move as we type the commands. Type the following set of commands. Remember to press RETURN after each command, and clear the screen before you start.

```
FD 60
RT 90
FD 60
RT 90
FD 60
RT 90
FD 60
RT 90
```



(This square is larger than our first square and is drawn in the opposite direction.)

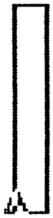
DRAWING RECTANGLES

Clear the screen and type the following set of commands.

```
FD 40
LT 90
FD 80
LT 90
FD 40
LT 90
FD 80
LT 90
```

This rectangle is wide but not very high. Clear the screen and draw another which is high and narrow.

```
FD 96
RT 90
FD 12
RT 90
FD 96
RT 90
FD 12
RT 90
```



EXERCISES

1.3 Draw some more rectangles. What is the width (in turtle steps) of the thinnest rectangle you can draw?

1.4 Experiment with very long rectangles. What happens when a rectangle is too long to fit on the screen?

ERROR MESSAGES IN IMMEDIATE MODE

What will happen if you make a mistake while working in immediate mode? In case you haven't found out already, type

```
FD50  (with no space between
       FD and 50)
```

Then press RETURN. You should see an error message on the screen:

```
THERE IS NO PROCEDURE NAMED FD50    I DON'T KNOW HOW TO FD50
```

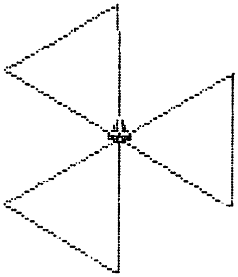
The incorrect command cannot be used by the computer, and is not recorded. All you have to do is re-type the command correctly and press RETURN. In fact this is generally how mistakes should be handled; once they are indicated, retype the command correctly and proceed.

EXERCISES

1.5 Draw an equilateral (equal-sided) triangle with a horizontal base. Hint: you will need to type another command before the triangle-drawing command.

1.6 Use FD, BK, RT and LT to draw equilateral triangles of different sizes.

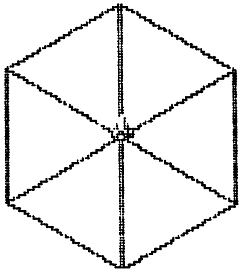
1.7 Draw three equilateral triangles arrayed around the turtle's central position, like this:



Hint: turn the turtle 120 degrees after each of the triangles.

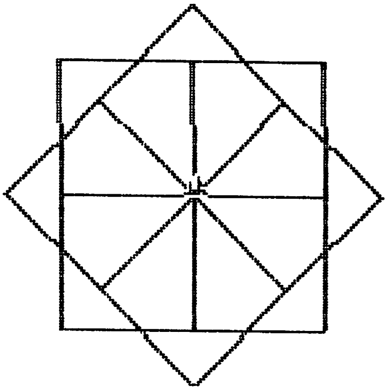
1.8 Use the FD and LT commands to explore the size of the screen. What happens when you command the turtle to go beyond the edge of the screen?

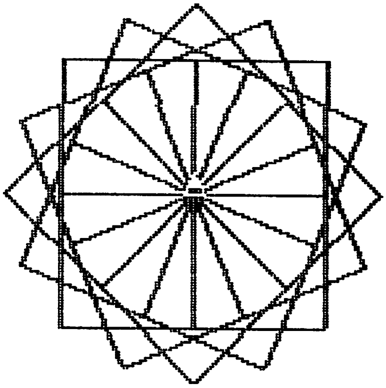
1.9 Draw a hexagon made up of six equilateral triangles, like this:



Hint: if 120 degrees gave 3 triangles in Exercise 1.7, what angle should give 6?

1.10 Spin some squares about the central position to make an interesting design, like these examples:





IDEAS INTRODUCED IN THIS CHAPTER

Use of the Apple computer.

Dealing with typing errors.

Drawing simple shapes.

Defining procedures.

Turtle state: position and heading.

Immediate mode and defining mode.

SUMMARY OF COMMANDS INTRODUCED IN THIS CHAPTER

MIT LOGO	APPLE LOGO	DESCRIPTION
ESC key	left arrow key	delete character to left
TO title	TO title	define new procedure
DRAW	CLEARSCREEN, CS	clear the screen and place the turtle at home
REPEAT	REPEAT	repeat commands
FORWARD, FD	FORWARD, FD	move the turtle forward
BACK, BK	BACK, BK	move the turtle back
LEFT, LT	LEFT, LT	turn the turtle left
RIGHT, RT	RIGHT, RT	turn the turtle right
ERASE title	ERASE "title	erase procedure
END	END	last line of procedure
CTRL-C	END	define new procedure
SPLITSCREEN	SPLITSCREEN	mixed text and drawings
FULLSCREEN	FULLSCREEN	full screen for drawings
CTRL-G	CTRL-G	stops current procedure

2 EDITING AND DEBUGGING PROCEDURES

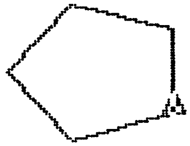
POLYGONS

We have written procedures to draw squares and triangles. Now let's try some other shapes. The following commands can be used to draw regular (equal-sided) shapes with more than four sides. Type them, one at a time, and study the shape each draws. Look for patterns in the commands.

First draw a pentagon

```
REPEAT 5 [FD 40 LT 72]
```

The computer should respond



This command has been written in immediate mode, and is used as soon as you press RETURN. If you write it again as a procedure, you can use it as many times as you like without re-typing it.

Type

```
TO PG
REPEAT 5 [FD 40 LT 72]
END
```

(Don't forget CTRL-C in MIT LOGO).

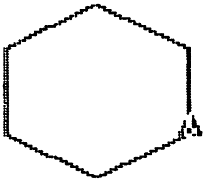
Clear the screen before each of the following drawings.
Type

```
PG
```

and the pentagon should be drawn.

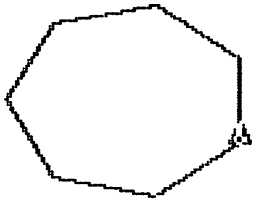
Next try a hexagon.

```
REPEAT 6 [FD 40 LT 60]
```



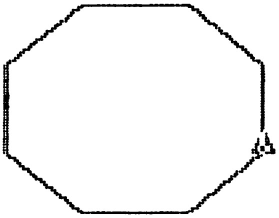
Now draw a heptagon.

```
REPEAT 7 [FD 40 LT 51.43]
```



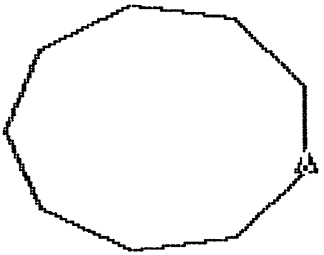
Try drawing an octagon.

```
REPEAT 8 [FD 40 LT 45]
```



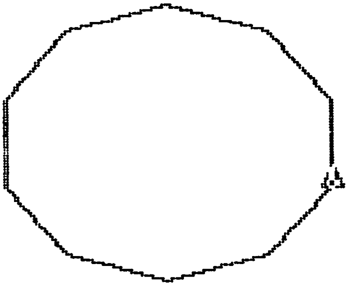
Have you remembered to clear the screen, or are you drawing one over the other? Will a nonagon be bigger or smaller than an octagon? Try it.

```
REPEAT 9 [FD 40 LT 40]
```



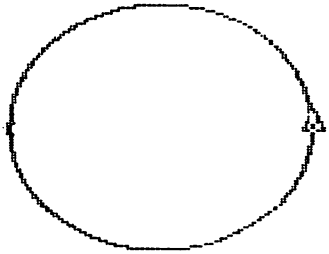
Now you can draw a decagon.

```
REPEAT 10 [FD 40 LT 36]
```



Finally, imagine a regular 360-sided shape, and draw that in immediate mode by typing

```
REPEAT 360 [FD 1 LT 1]
```



This last drawing looks almost like a circle. It is in fact distorted by the screen to make an oval shape. We will discuss how to overcome this distortion effect in Chapter 4.

STOPPING PROCEDURES

If you are using a procedure that is going on and on, and you wish to stop it, press CTRL-G.

EXERCISES

2.1 Write procedures to draw the following regular polygons: a hexagon, a heptagon, an octagon, a nonagon, a decagon and a circle. You can give these any titles you wish; make them easy to remember.

THE TOTAL TURTLE TRIP THEOREM

If you look closely, you will see a pattern in these commands. The product of the repeat number and the angle is always 360.

SHAPE	REPEAT	DEGREES	PRODUCT
Triangle	3	120	3 x 120 = 360
Square	4	90	4 x 90 = 360
Pentagon	5	72	5 x 72 = 360
Hexagon	6	60	6 x 60 = 360
Heptagon	7	51.43	7 x 51.43 = 360
Octagon	8	45	8 x 45 = 360
Nonagon	9	40	9 x 40 = 360
Decagon	10	36	10 x 36 = 360
Circle	360	1	360 x 1 = 360

For the turtle to go all the way around and finish with the same heading as it began with, it must turn through 360 degrees, or a multiple of 360, e.g. 720. This is regardless of the number of sides it draws.

This is called the Total Turtle Trip Theorem. We can say that when the turtle draws a figure and starts and ends in the same position, and is pointing in the same direction as when it started, it has turned through 360 degrees.

If we look at each figure, we will see that the number of sides to be drawn in each case is 360 divided by the angle.

INPUTS TO PROCEDURES

We have a pentagon-drawing procedure

```
TO PG
REPEAT 5 [FD 40 LT 72]
END
```

Each time we use this procedure, it will draw a pentagon of side length 40 turtle steps. A pentagon-drawing procedure would be more useful if it could draw pentagons of various sizes. We could specify the side length needed each time the procedure is used. Since this is a different procedure it must have a different name. Let's choose PENTAGON. For example

there must be a space before the colon, but not one after it

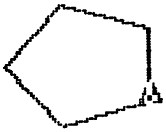
```
TO PENTAGON :SIDE
REPEAT 5 [FD :SIDE LT 72]
END
```

The word SIDE is called an input to the procedure. Notice how SIDE is used in the FD command, instead of a fixed number.

Using this procedure, we can now draw pentagons of different sizes. When the procedure has been defined, PENTAGON DEFINED will appear. Next to the ? type

```
PENTAGON 35
```

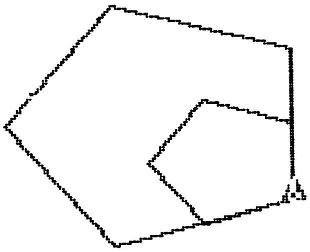
The computer should respond



Now try (don't clear the screen first)

```
PENTAGON 70
```

The computer should respond



The number that we used, i.e. 35 or 70, is substituted for :SIDE wherever it appears in the procedure.

There is nothing special about the word that is used for the input in PENTAGON, i.e. SIDE. Any word or letter can be used, but it must commence with a colon, and can have no spaces within it. It is better if the word is meaningful, for instance we used :SIDE in the pentagon procedure. Another meaningful word would be LENGTH, or maybe the letter L.

A FIRST POLY PROCEDURE

We can vary the angle as well as the length of the sides. If we use both these inputs together, we can make any number of regular polygons and in any sizes. The procedure would look like this

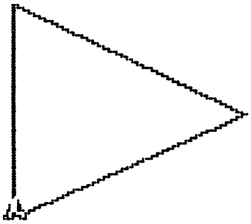
```
TO POLY :SIDE :ANGLE
REPEAT 360/:ANGLE [ FD :SIDE RT :ANGLE]
END
```

this means 360 divided by the angle

When the REPEAT statement is executed, the computer will calculate the number of times to repeat the commands by dividing 360 by the angle that we have input. Try the following

```
POLY 100 120
```

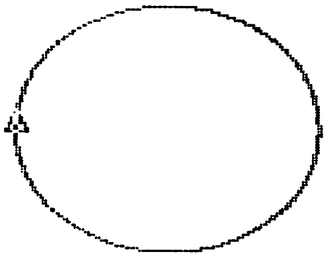
The computer should respond



This will draw a triangle, since the computer will repeat the commands in the brackets 360/120 = 3 times, and turn 120 degrees each time.

To draw a circle we can type

```
POLY 1 1
```



A circle can be thought of as having a lot of very small sides. Going forward 1 step, turning left 1 degree, and doing this 360 times, should draw us a circle.

Try playing "turtle" yourself.

EXERCISES

2.2 Using POLY, draw a pentagon, a hexagon and a circle.

2.3 Draw a circle of SIDE 2 and ANGLE 2. What is different from a circle of SIDE 1 and ANGLE 1?

2.4 Now draw a circle of SIDE 2 and ANGLE 4. What has happened?

2.5 Try a polygon of SIDE 10 and ANGLE 10.

2.6 What relationship is there between the side length of the polygon and the speed with which the turtle draws it?

2.7 Write a procedure SPINPOLY

```
TO SPINPOLY
REPEAT 4 [ POLY 50 90 LT 90]
END
```

Experiment using this procedure. Can you modify SPINPOLY, so that it has three inputs, a side and an angle to draw a particular polygon, and a spin to turn the polygon each time it is drawn?

COMMANDS FOR EDITING PROCEDURES

Type and define the step by step square procedure.

```
TO SQ3
FD 50
LT 90
FD 50
LT 90
FD 50
LT 90
FD 50
LT 90
END
```

If you have defined and used a procedure, and it is not as you wish, then you can change or edit it. Typing EDIT and the title of the procedure places the system in the edit mode (similar to the defining mode), and the procedure is listed on the screen. Then you can make changes. Type

MIT LOGO

```
EDIT SQ3
```

APPLE LOGO

```
EDIT "SQ3
```

The computer will enter edit mode and the SQ3 procedure will be listed

cursor

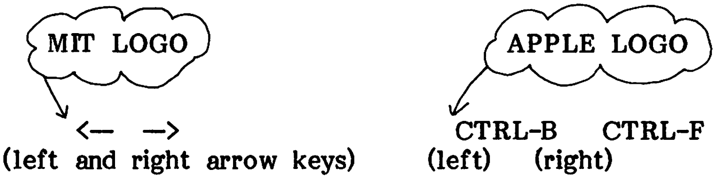
```
TO SQ3
FD 50
LT 90
FD 50
LT 90
FD 50
LT 90
FD 50
LT 90
END
```

The cursor is on the T at the top left of the screen.

The following commands can be used to correct errors and make alterations to procedures in edit mode. There are more, but we will only discuss about six of them as that is all we need at present.

Moving the Cursor Left and Right:

These keys enable you to move the cursor left and right along a line, without changing the characters on the line. At the end of a line, pressing the same key again moves the cursor to the next line. Try it and see what happens.



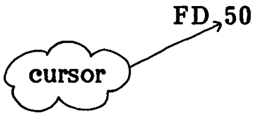
Use the right hand movement key to move the cursor to the first space after the LT 90. Now move the cursor with the left movement key back to the T in the left top corner of the screen.

Remember when you use the CTRL key you hold it down first, then press the other key mentioned (B in CTRL-B).

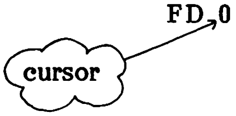
Deleting the Character Under the Cursor (CTRL-D)

Now use the right hand cursor movement key to move the cursor to cover the 5 of the first FD 50.

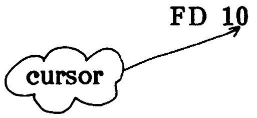
You can erase the letter under the cursor by holding down the CTRL key and pressing D



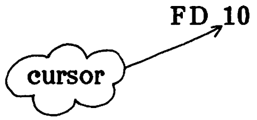
Now press CTRL-D and you will have



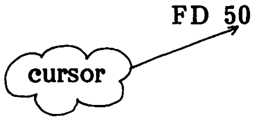
Now type 1



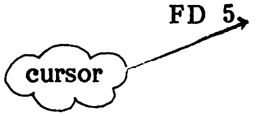
Move the cursor with the cursor control key back to the 1.



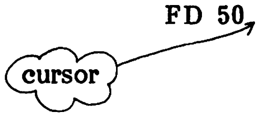
Press CTRL-D to erase the 1, then type 5 to replace it.



Holding the CTRL key down and pressing D several times, deletes one character each time. Now use CTRL-D to delete the 0, so you have



Now type 0 to get back the original line.

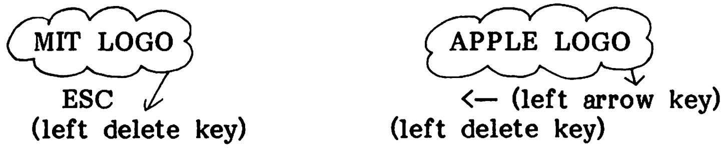


If You Are Really Messing It Up

If you have typed lots of things wrongly then don't despair; type CTRL-G. This stops everything without keeping your changes, and you will be back in immediate mode. You can type EDIT and the title of the procedure, and start again.

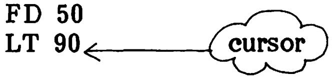
Deleting the Character to the Left of the Cursor

If you wish to delete a character to the left of the cursor, then you press the ESC key if you have MIT LOGO, and press the left arrow key if you have APPLE LOGO.



The cursor is on the first space after the 0 of FD 50. Now use the left delete key to remove the entire line, then retype the line (but don't type RETURN at the end of the line).

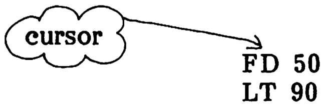
Now move the cursor with the right movement key until it is at the end of LT 90.



Moving the Cursor Up One Line (CTRL-P)

Holding the CTRL key down and pressing P, moves the cursor up one line.

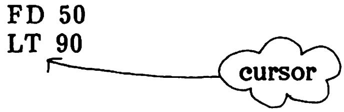
Now press CTRL-P to move the cursor up one line, and move it left until it is over the D.



Moving the Cursor Down One Line (CTRL-N)

Holding the CTRL key down and pressing N, moves the cursor down one line.

Press CTRL-N and you will have



Inserting in the Middle of a Line

You will have already inserted characters into a previously typed line when you changed FD 50 to FD 10. When inserting one or more characters into the middle of a line, the cursor should be moved to the position after that in which the character is to be put, and typing started from there.

Now type AAA and you will have

```
FD 50
LAAAT 90
```

Notice how the characters in front of the cursor move ahead of it.

Now move the cursor back to the first A and delete with CTRL-D three times to get rid of the A's.

Placing a New Line Between Existing Lines (CTRL-O)

To add a line in between two other lines, move the cursor to the beginning of the line below where you wish to insert the new line. Hold down the CTRL key and press O. A line space will appear.

Now use CTRL-P and the left movement key to position the cursor at the beginning of the first FD

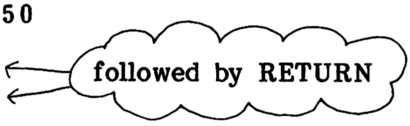
```
TO SQ
FD 50
```

Press CTRL-O,and a line will open

```
TO SQ

FD 50
```

Now type in

```
AA
BB
CC
```



Your procedure will look like this

```
TO SQ3
AA
BB
CC
FD 50
LT 90    etc.
```

Now use the left delete key continuously to remove AA, BB, CC. Alternatively use the left movement key to take the cursor back to the first A, then use CTRL-D continuously.

The REPT Key

The key marked REPT is used to repeat continuously any other key pressed. If you type F with one finger and at the same time put another finger on the REPT key, you should see

```
FFFFFFFFFFFFFFFFFFFFF
```

To stop the repeating, take your finger off the REPT key.

If you wish to move the cursor up several lines, hold down CTRL-P and the REPT key until the cursor moves up to the required line. The REPT key also works with the ESC and CTRL-D keys to delete more than one character at a time.

Now throw the edit away with CTRL-G so whatever you have done cannot change the original SQ3.

Going Over the End of a Line

The Apple screen only allows 40 characters on one line. If you are typing, you can go over the end of a line and on to the next one. It does not matter if the information goes beyond the end of the line; just keep typing. You use the RETURN key to indicate the end of the LOGO command.

Now type

FFFFFFFFFFFFFFFFFFFFFFFF (45 or so of them)

When you came to the end of the line you could just keep typing. If in edit mode LOGO places an exclamation mark "!" at the end of the line, just to tell you that you have gone to the next line, you can ignore this mark.

You can also place more than one command on a line, such as

```
FD 50 RT 90
```

Both will be executed, as if they were on different lines. A warning: it is harder to read a procedure and understand what it is doing if there are a lot of commands on one line.

SOME MEMORY AIDS

CTRL-C	Complete (define)
CTRL-D	Delete under cursor
CTRL-P	uP
CTRL-N	dowN
CTRL-G	Garbage, throw edit away
CTRL-O	Open a new line space

AN EDITING EXERCISE

Type

```
EDIT EX                         EDIT "EX
```

The computer will enter edit mode.

Press RETURN, then type these three lines exactly as they appear.

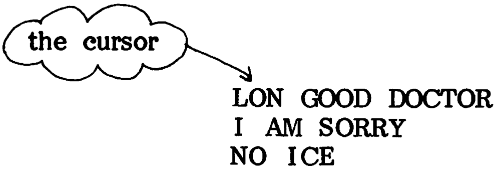
```
LEAD ON GOOD DOCTOR
I AM SORRY
NO ICE
```

Now define this procedure by pressing CTRL-C, and then EDIT EX again. The procedure will appear on the screen. (Remember "EX in APPLE LOGO)

Move the cursor down so it is over the first letter (L), and carry out the following instructions. If things go badly wrong press CTRL-G to throw your editing away, and type EDIT EX again.

1. Move the cursor until it is over the first E (of LEAD).

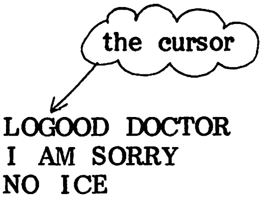
2. Use CTRL-D to delete characters until the next O is under the cursor (don't delete this). You should now have



```
LON GOOD DOCTOR
I AM SORRY
NO ICE
```

3. Move the cursor to the next character (N).

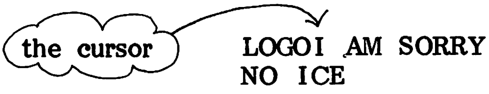
4. Use CTRL-D to delete the N and the space. You should now have



```
LOGOOD DOCTOR
I AM SORRY
NO ICE
```

5. Move the cursor until it is over the third O in the line.

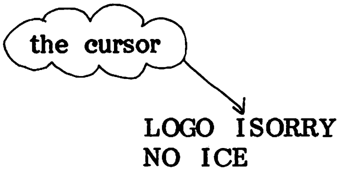
6. Now use CTRL-D to delete the next eight letters until the I is under the cursor. You will notice the next line comes to meet you. The message is now



```
LOGOI AM SORRY
NO ICE
```

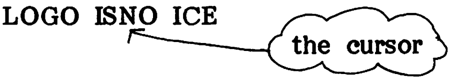
7. Press the space bar once, then move the cursor until it is on the space after the I.

8. Use CTRL-D to delete until the S is under the cursor, and you should have



```
LOGO ISORRY
NO ICE
```

9. Move the cursor until it is over the next O, then delete with CTRL-D until the N is under the cursor. The procedure will look like this



```
LOGO ISNO ICE
```

10. Now type one space and then move the cursor until it is over

the O. Now remove the O and the next space, and the I should end up under the cursor, like this

```
LOGO IS NICE
```

Notice how the characters come up from the next line automatically.

If you didn't get this result you should use CTRL-G, then type EDIT EX again and start from the beginning of the exercise.

The best way to become confident with editing is to do a little practice. Have a look at what happens if you press RETURN while the cursor is in the middle of a line. You can then use the left delete key to get you back to the original. Try some other editing for practice.

EXERCISE

2.8 Type your full name in a procedure:

```
TO MYNAME
CHRISTOPHER ANTHONY ADAMS
END
```

your own name of course

Define the procedure. Now edit it as follows.

Put each name on a separate line. Hint: move the cursor to the end of each name and press RETURN.

Put them all back onto one line. Hint: move the cursor to the first character on each line and use the left delete key.

Give yourself an additional middle name.

Delete that extra name - you didn't really like it much.

Reduce the whole lot to your initials.

Define the procedure in its new form, and use it to see what happens.

GETTING READY TO DRAW A HOUSE

Type in the following procedure:

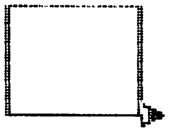
```
TO WALLS
FD 50
LT 90
FD 50
LT 90
FD 50
LT 90
FD 50
END
```

Now define the procedure (remember in MIT LOGO press CTRL-C).

Test that the procedure works by typing

```
WALLS
```

The computer should respond



Notice that the turtle is pointing to the right, not straight up (the direction it pointed in when it started). Adding another 90 degree turn before the end of the WALLS procedure now will save us some time later. To do this, type

```
EDIT WALLS          EDIT "WALLS
```

MIT LOGO

APPLE LOGO

The computer lists the procedure

```
TO WALLS
FD 50
LT 90
FD 50
LT 90
FD 50
LT 90
FD 50
END
```

Move the cursor down, using CTRL-N, until it is on the E of END. Then press CTRL-O to make a new line and type

```
LT 90
```

If you were now to define the new procedure, you would lose the old one. Do not do this just now, because we will need the original WALLS procedure later. Instead we will give the new procedure another title. This means we will not lose the old procedure when we define the new one. Press CTRL-P to move the cursor up to the line

```
TO WALLS
```

and with the right hand movement key, move the cursor across to the space past the S and type 1. The title of the new procedure is now WALLS1.

```
TO WALLS1
```

Don't forget to define the procedure, press CTRL-C. In the edit mode, you must press CTRL-C in both MIT LOGO and APPLE LOGO.

THE WORKSPACE

Once defined, a procedure is stored in the workspace in the computer. The workspace contains the procedures that you have defined in a LOGO session. You can use the procedures which you have defined, again and again. Try it. Type

```
DRAW                        CLEARSCREEN
WALLS                       WALLS
```

or

```
DRAW                        CLEARSCREEN
WALLS1                      WALLS1
```

Both procedures will be executed in turn because the procedures are all stored in the workspace as long as the power to the computer is on. Once the power is turned off, the procedures in the workspace are lost.

To check whether a particular procedure is presently in the workspace, you can type

```
     EDIT title                    EDIT "title
```

For example type

```
     EDIT WALLS1                   EDIT "WALLS1
```

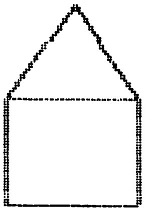
If the procedure is there, it will be listed, and typing CTRL-C will change the system back out of edit mode.

If the procedure is not in the workspace, the title TO WALLS1 will appear on the screen in edit mode with a blank screen so you can just type it in.

Note that edit mode can be used to type new procedures or to change old ones.

DRAWING A HOUSE

Now we are ready to make a slightly more complex drawing. We will try a house.

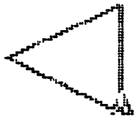


Complex pictures can often be thought of as a collection of simpler shapes. This house is made up of a square and a triangle.

We already have in the workspace a procedure for drawing a square, WALLS. We can now define a procedure to draw a triangle, called ROOF. Type

```
TO ROOF
REPEAT 3 [FD 50 LT 120]
END
```

If you are using MIT LOGO, remember to define the procedure with CTRL-C.



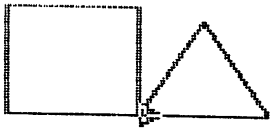
As a first try at writing at a house procedure, then, type

```
TO HOUSE
WALLS
ROOF
END
```

The house procedure contains the WALLS and ROOF procedures as sub-procedures. Type

```
HOUSE
```

The drawing looks like this



This is a start, but not quite what we want. When the turtle drew the square, it ended heading across the screen, horizontally. It started drawing the triangle from this position and this heading. Now you play "turtle" with pencil and a piece of paper. Draw out the procedure step by step with a pencil.

List the WALLS procedure again, using the EDIT command

```
     EDIT WALLS                    EDIT "WALLS
```

Note that it has four FD commands, but only three LT commands. Although the square has been completed, the turtle has finished with a heading different from that with which it started. This problem is overcome if we use the WALLS1 procedure. Type CRTL-C, then check that it is still in the workspace. Type

```
     EDIT WALLS1                   EDIT "WALLS1
```

then define the procedure with CTRL-C.

We will change the HOUSE procedure to include WALLS1 as a sub-procedure instead of WALLS. Type

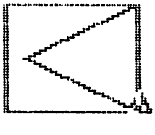
```
     EDIT HOUSE                    EDIT "HOUSE
```

Now change the line WALLS to WALLS1 by adding the 1. The HOUSE procedure now looks like this

```
TO HOUSE
WALLS1
ROOF
END
```

Define the new procedure with CTRL-C, and the old procedure is replaced. Now type

```
HOUSE
```



The drawing looks like this - it is still not correct.

DEBUGGING

Computer programs, like lots of things we do, very often do not work properly the first time. The skills of debugging, that is removing the "bugs" or errors from procedures, are important ones for any computer programmer to develop.

When a procedure doesn't produce the expected result

1. Look carefully at the result it does produce.

2. Decide exactly how this differs from the desired result.

3. Carefully read the commands which are at present in the procedure to work out what is wrong and what changes are needed. Playing "turtle" with a pencil and paper can often help.

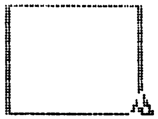
This is just what we are doing with our HOUSE procedure, so let's continue.

We have a main procedure, HOUSE.

```
TO HOUSE
WALLS1
ROOF
END
```

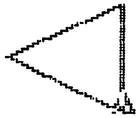
It contains two sub-procedures, WALLS1 and ROOF.

```
TO WALLS1
FD 50
LT 90
FD 50
LT 90
FD 50
LT 90
FD 50
LT 90
END
```

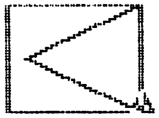


and

```
TO ROOF
FD 50
LT 120
FD 50
LT 120
FD 50
LT 120
END
```

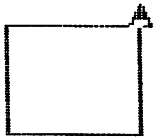


and the resulting drawing is as shown.



Play "turtle" and trace out the drawing step by step. The first sub-procedure in HOUSE is WALLS1, ending with the turtle in the central position and heading upwards.

Before it does the triangle sub-procedure, it should move up to the top of the square. It needs another FD 50.



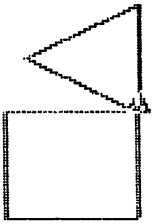
Change the HOUSE procedure to include this command. Refer to the section on editing procedures if you need to. Remember to define the new version of HOUSE with CTRL-C. It should now look like this.

```
TO HOUSE
WALLS1
FD 50
ROOF
END
```

Type

```
HOUSE
```

to use the procedure - and we are nearly there!



When the turtle begins the triangle it should not be heading upward. We need a LT command, to turn it to the correct heading. You might work out by trial and error the size of the turn required, or you might calculate that it is LT 30.

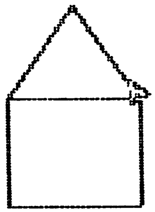
Change the HOUSE procedure once more to add LT 30 just before the triangle is drawn.

```
TO HOUSE
WALLS1
FD 50
LT 30
ROOF
END
```

Define the procedure with CTRL-C, and type

```
HOUSE
```

Presto!



PROCEDURES AND SUB-PROCEDURES

Once a procedure is defined it becomes part of the turtle's "vocabulary", like FD and LT. It can then be used either on its own in immediate mode, or as a command in another procedure. In the latter case, it is called a sub-procedure. Our HOUSE procedure contained two sub-procedures, WALLS1 and ROOF, as well as some primitive commands. A primitive command is a command which is part of the LOGO language, in this case FD and LT.

IDEAS INTRODUCED IN THIS CHAPTER

Total turtle trip theorem

Variable inputs to procedures

Edit mode

Editing procedures

The workspace

Drawings combining two procedures

Debugging

Procedures and sub-procedures

SUMMARY OF COMMANDS INTRODUCED IN THIS CHAPTER

MIT LOGO	APPLE LOGO	DESCRIPTION
EDIT title	EDIT "title	enter EDIT mode
Cursor movement keys:		
<—(left arrow)	CTRL-B	move cursor left
—>(right arrow)	CTRL-F	move cursor right
CTRL-P	CTRL-P	move cursor up one line
CTRL-N	CTRL-N	move cursor down one line
CTRL-O	CTRL-O	insert new line
Delete character keys:		
ESC (esc key)	<— (left arrow)	delete character to left of cursor
CTRL-D	CTRL-D	delete character under cursor
Defining procedures:		
CTRL-C	CTRL-C	define procedure in edit mode
CTRL-G	CTRL-G	discard edit

3 TURTLE PROJECTS

The projects in this chapter have been contributed by Charlie, Chris and Gareth.

HIDETURTLE AND SHOWTURTLE

You might prefer not to have the turtle itself show in your picture. This can be achieved using the HIDETURTLE (abbreviated HT) command. Let's try it.

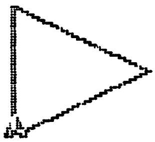
Draw a triangle. Type

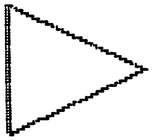
```
REPEAT 3 [FD 60 RT 120]
```

Press RETURN and the turtle will appear in the drawing.

Type

```
HT
```





Note that this time no turtle is shown. If you wish to make the turtle re-appear on the screen, use the SHOWTURTLE (abbreviated ST) command.

Type

```
ST
```

If you notice that the turtle is not on the screen at any time, remember just type ST and it will re-appear.

EXPERIMENTING WITH SIDES AND ANGLES

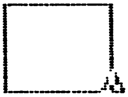
Let's write some more procedures and include sub-procedures to draw a variety of pictures. We will also explore some familiar procedures further, such as our square. This time let's use a nonsense title that has no relation to the drawing. Let's use EAT. This procedure will sometimes draw a square, but not always, depending on what number is typed in for the angle. Now type

```
TO EAT :S :A
REPEAT 4 [FD :S LT :A]
END
```

Define EAT and then type

```
EAT 40 90
```

We will have a square shape like this



This shape is a square because the angle is 90 and the number of repeats is 4. (Remember 90 X 4 = 360.) Now experiment with different inputs in the EAT procedure. Don't forget to clear the screen.

Type

```
EAT 40 180
```

or

```
EAT 40 45
```

and

```
EAT 100 20
```

WRAPPING

Do you still have a square shape? You will notice that the turtle has gone off the edge of the screen and come back on the other side. This is called wrapping.

ROTATING SHAPES

Draw a shape using EAT, and then move it around its own axis. We will call the procedure ROTATE and include EAT as a sub-procedure. Type

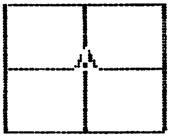
```
TO ROTATE :S :A
REPEAT 360/:A [EAT :S :A LT 90]
END
```

We need to type a number for the length and a number for the angle. The angle 90 will produce a square, so let's first try rotating a square.

Type

```
ROTATE 30 90
```

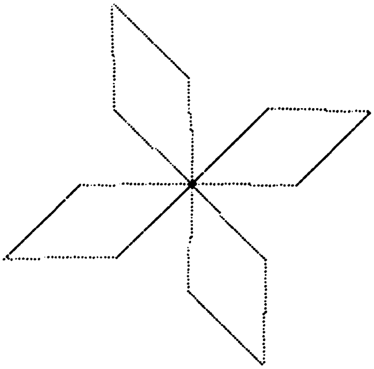
This is what should be on the screen



Now try a smaller angle but still a number which divides exactly into 360, such as 45. Type

```
ROTATE 30 45
```

This drawing should look like this

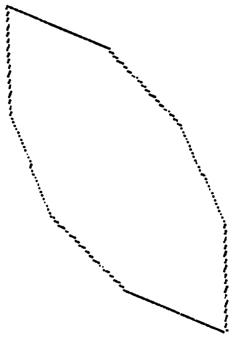


Try an even smaller angle, such as half of 45, which is 22.5. Note that the decimal point is typed by using the fullstop or period key, (.).

Type

```
ROTATE 30 22.5
```

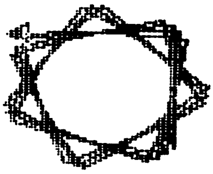
Did your drawing look like this?

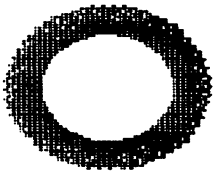


We can now try some very small angles. Type

```
ROTATE 15 3
```

The diagrams below show this shape partly drawn and then completed.

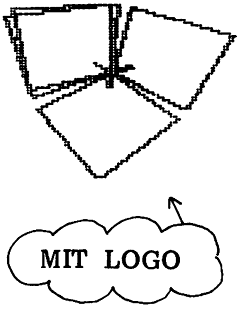


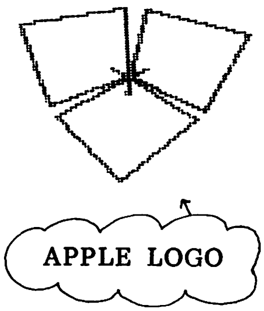


We have used angles which divide evenly into 360. Now we can be adventurous and try some different angles, such as

```
ROTATE 40 97
```

Is this how yours looks?

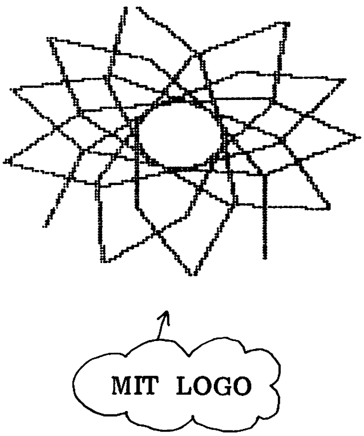


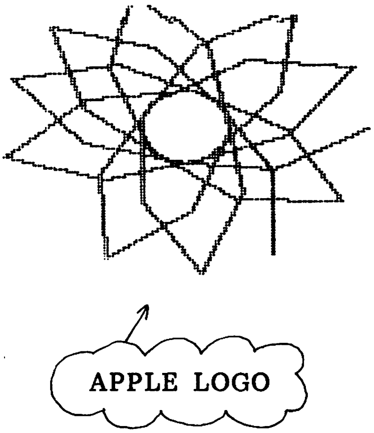


Here is another example. Type

```
ROTATE 40 33
```

This looks more interesting.

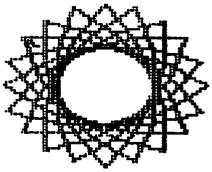




Notice how this one has what look like loose ends. Why do you think this is? Hint: remember the Total Turtle Trip Theorem in Chapter 2?

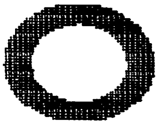
Here are two more. First type

```
ROTATE 20 18
```



Then type

```
ROTATE 10 1
```



This one is going on a bit isn't it? You can stop it if you wish to move on to other drawings. Use CTRL-G.

You could play with this for hours, even weeks. For example, what would happen if, when the turtle stops after completing ROTATE 40 97, you typed in again

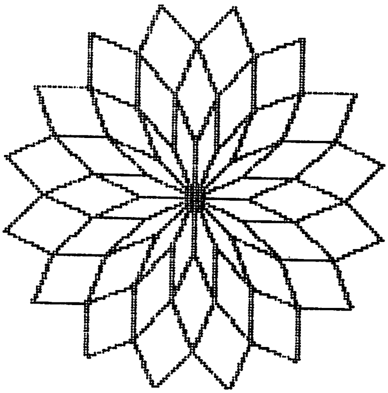
```
ROTATE 40 97
```

Let's type a procedure to rotate ROTATE like this. Type

```
TO ROTATE2 :S :A
REPEAT 360/:A [ROTATE :S :A LT :A]
END
```

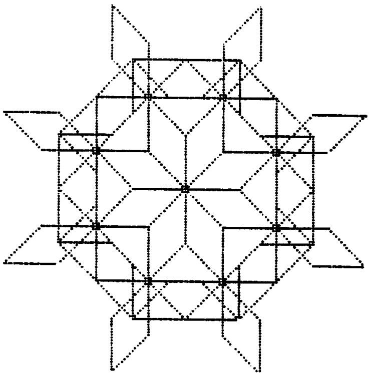
First try

```
ROTATE2 20 22.5
```



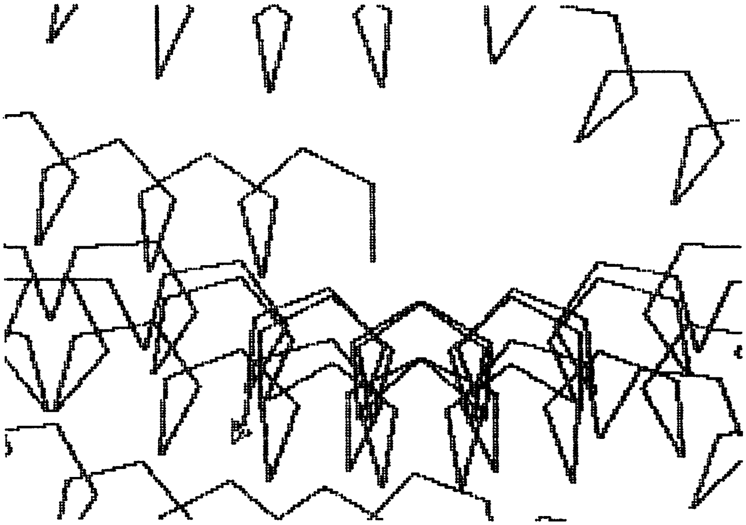
Now try

ROTATE2 20 45



Let's try something a bit different

ROTATE2 30 65



This is different, isn't it? You can continue with these for a long, long time. Experiment some more.

TYPING MORE THAN ONE PROCEDURE

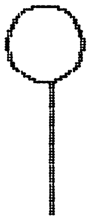
Now is the time to move on to other drawings. Do you feel hungry? Perhaps you would like a lollipop! Type

```
TO LOLLIPOP
FD 50
LT 90
POLY 5 20
END
```

There is a sub-procedure, POLY. That is no problem. If POLY is already in the workspace, define LOLLIPOP. If POLY is not in the workspace just continue typing POLY in the defining mode, and define the two procedures.

```
TO POLY :S :A
REPEAT 360/:A [FD :S RT :A]
END
```

Now define the procedures and type LOLLIPOP. This is how it should look.



Did you notice that the side and angle numbers for the POLY procedure were included in the LOLLIPOP procedure? What would happen if you changed these numbers in the LOLLIPOP procedure?

EXERCISES

3.1 Write a procedure to draw a flower with sixteen petals, similar to that drawn by ROTATE2 20 22.5. Do not draw over existing lines.

3.2 Try some angles and sides that haven't already been used in ROTATE, and ROTATE2. For example, use angles of 30, 15 and 7.5 degrees.

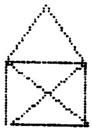
3.3 Rewrite LOLLIPOP (call it LOLLIPOP2), so it has inputs which vary the angle and side in LOLLIPOP.

DRAWING A HOUSE WITHOUT LIFTING THE PEN

Have you ever been challenged to draw a house without lifting the pen, and without drawing over existing lines? Let's type a procedure to do this for us, and title it CRAZY. Type

```
TO CRAZY
FD 30 LT 30 FD 30 LT 120 FD 30 LT 120
FD 30 RT 135 FD 40 LT 133 FD 30 LT 140
FD 43 LT 133 FD 30 HT
END
```

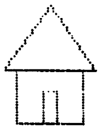
This is what CRAZY should look like.



Note that we have spaced the commands across rather than down the page in this procedure.

A COTTAGE WITH EAVES

We have drawn some houses, but we have not drawn a cottage with eaves. Such a cottage might look like this.



Type these procedures

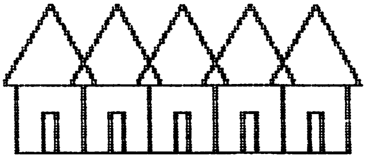
```
TO COTTAGE
SIDES LT 90 FD 10 RT 90 ENTRANCE
LT 90 FD 15 LT 90 FD 25 LT 90 EAVES
END

TO SIDES
REPEAT 4 [FD 25 LT 90]
END

TO ENTRANCE
FD 15 LT 90 FD 5 LT 90 FD 15
END
```

```
TO EAVES
FD 30 RT 120 FD 35 RT 120
FD 35 RT 120 FD 5
END
```

Now type

```
COTTAGE
```

How would you draw a terrace of cottages, that is a row of cottages with common walls? Hint: check the position of the turtle when each cottage is finished, before you draw the next one. This is how it could look.



PENUP AND PENDOWN

In all the pictures we have drawn so far, the turtle has left a visible trail. There are times when we do not wish to see the trail. We can type PENUP (abbreviated PU), then move the turtle to a new position using FD, BK, LT and RT commands, then type PENDOWN (abbreviated PD) and the procedure we wish to use. Let's try it. Type

```
FD 30
PU
FD 20
PD
FD 30
```

When we wish to move the turtle without leaving a trail, we often need to change the direction in which the turtle is heading as well. Type two procedures, to move to the left or the right, titled SLIDEL and SLIDER

```
TO SLIDEL :D
PU
LT 90
FD :D
RT 90
PD
END
```

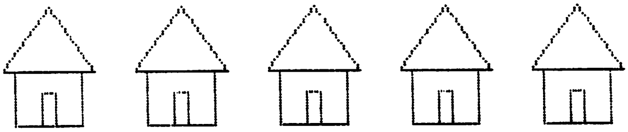
and

```
TO SLIDER :D
PU
RT 90
FD :D
LT 90
PD
END
```

We can make an avenue of cottages using the SLIDEL and SLIDER procedures. Type

```
TO AVENUE
SLIDEL 105
REPEAT 5 [COTTAGE LT 90 FD 25 LT 180 SLIDER 50]
END
```

Does it look like this? Could you draw several avenues?

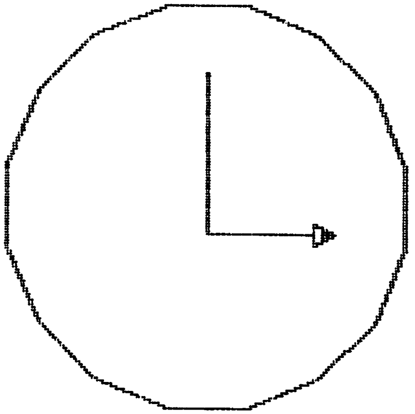


GOING HOME

The turtle has a home, which is in the centre of the screen heading upwards. Any time we wish the turtle to return to this position, we type HOME and the turtle returns. The turtle will leave a trail, unless we type PENUP before typing HOME.

DRAWING A CLOCK

We can use the PENUP, PENDOWN and HOME commands and also include the SLIDER procedure to draw a clock.



Type

```
TO TIME
SLIDER 90 CLOCK HANDS
END

TO CLOCK
REPEAT 16 [LT 22.5 FD 30 ]
END

TO HANDS
PU HOME BK 25 RT 90 FD 15 LT 90 PD
FD 60 BK 60 RT 90 FD 40
END
```

Define the above procedures and then type TIME. If your clock goes low on the screen where there is text, type FULLSCREEN to see the complete drawing. There will be no text on the screen when you do this. The turtle makes a nice point on the hour hand. If it is not shown, type ST. Remember you can type SPLITSCREEN if you want to see the text at the bottom of the screen again.

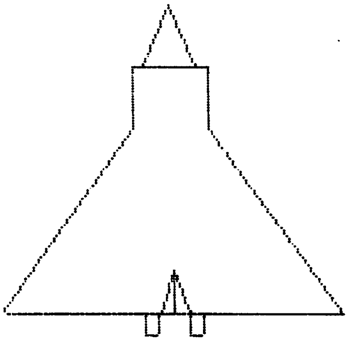
EXERCISES

3.4 Use SLIDEL and SLIDER with the square and triangle procedures, to move the drawings in lines round the screen. What happens if the direction in which the turtle is facing isn't straight up when you start the drawings?

3.5 This is a harder exercise that you might want to come back to later. Rewrite the clock drawing procedures so that you can use an input that sets the hour hand at a particular hour.

DRAWING A PLANE

We will type a longer procedure, titled PLANE. It uses many sub-procedures and includes the PENUP and PENDOWN commands. It looks like this.



Now type each procedure and define it as you go. If you make a typing error, remember the edit commands in Chapter 2.

```
TO NOSE
PU FD 20 PD RT 160 FD 30 BK 30
LT 320 FD 30 LT 110 FD 25
END
```

```
TO FUSELAGE
RT 90 FD 29 BK 29 RT 90
FD 29 LT 90 FD 29
END

TO WINGS
RT 30 FD 100 BK 100 PU LT 120 FD 29
PD RT 60 FD 100 RT 120 FD 127
END

TO TAIL
BK 63.5 RT 90 FD 20 LT 165 FD 21
BK 21 LT 30 FD 21 RT 15
END

TO TL
FD 10 LT 90 FD 5 LT 90 FD 10 LT 90
END

TO PLANE
PU FD 80 PD NOSE FUSELAGE WINGS
TAIL TL FD 22 LT 90 TL HT
END
```

When you have defined the sub-procedures and the procedure PLANE, type PLANE.

DRAWING A STICK FIGURE

We have a house, a plane and a clock, but no people to use these things. It is time to create some people. This procedure has a sub-procedure POLY. If POLY is in the workspace, do not type it again. If it is not in the workspace, then you will need to retype POLY when you come to it. We will title this procedure MAN. Type

```
TO BODY :S
FD :S
END
```

Define BODY and type BODY 50. Now continue. Type

```
TO HEAD
LT 90 POLY 1 5 RT 90 PU FD 3 LT 90 PD
FD 3 BK 6 FD 3 RT 90 PU FD 2 PD FD 5
PU FD 2 LT 90 FD 3 PD FD 1 PU BK 8 PD
FD 1 PU FD 3 LT 90 FD 12 PD
END
```

Type POLY now, if it is not in the workspace.

```
TO POLY :S :A
REPEAT 360/:A [FD :S RT :A]
END
```

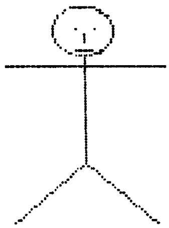
Does the HEAD procedure work? Now for the arms and legs.

```
TO ARMS
LT 90 FD 30 BK 60 FD 30 RT 90
END

TO LEGS
LT 135 FD 40 BK 40 LT 90
FD 40 BK 40 LT 135
END
```

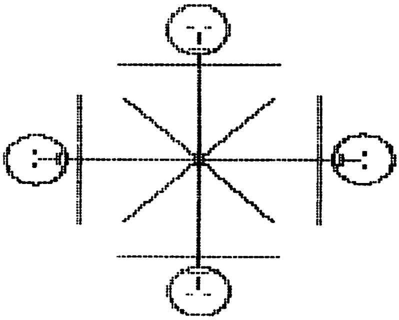
Let's put the man together. Type

```
TO MAN
BODY 50 HEAD BODY 5 ARMS
BODY 45 LT 180 LEGS HT
END
```

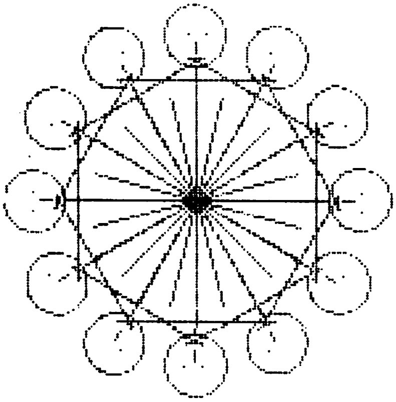
Don't forget to clear the screen. Type MAN and see if your drawing looks like this.

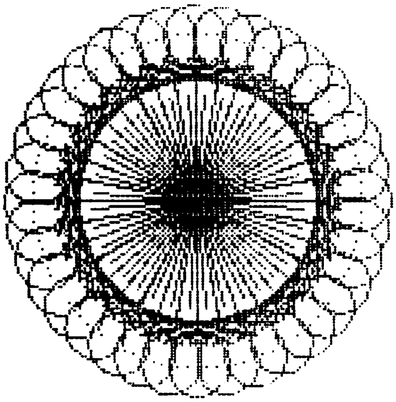


Turn the turtle LT 90 and type MAN again. If you do this twice more, the drawing should look like this.



Turn the turtle 30 degrees and fill in the gaps. Now try turning the turtle 10 degrees. Remember the REPEAT command.





EXERCISE

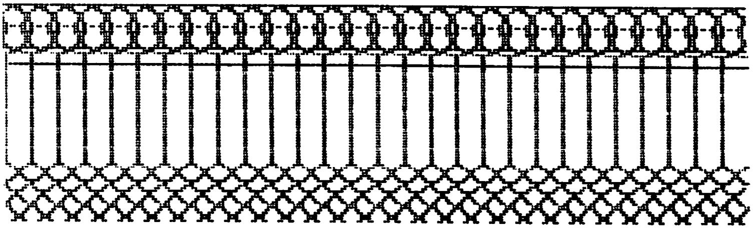
3.6 Write a procedure called SPINMAN, with an input that is the number of degrees to turn MAN each time.

CLEARING THE SCREEN WITHOUT MOVING THE TURTLE

We can move our man across the screen by typing a procedure titled MARCH, which includes the SLIDEL procedure. (If the SLIDEL procedure is not in the workspace, type it in now.)

```
TO MARCH
REPEAT 28 [MAN SLIDEL 10]
END
```

The man will march across the screen like this.



If you would like the man to march across the screen and not leave a drawing of himself each time, you can clear the screen without returning the turtle to the central position. Type

CLEARSCREEN
(abbreviated CS)

MIT LOGO

CLEAN

APPLE LOGO

Let's try it. Type

```
TO MARCH1
REPEAT 6 [MAN SLIDEL 20 CS]
END
```

```
TO MARCH1
REPEAT 6 [MAN SLIDEL 20 CLEAN]
END
```

Now type MARCH1 and watch the man move across the screen.

SETTING THE TURTLE'S HEADING AND CO-ORDINATES

Sometimes we may wish to set the turtle's direction to a particular angle, independently of where it was previously pointing. For example, heading 0 or heading 360 always points the turtle upwards, and heading 180 points it straight down.

Type

```
PU
HOME
```

The turtle will be in the centre of the screen facing straight up. Type

```
SETHEADING 180
```

The turtle should now be facing straight down.

Type

```
SETHEADING 180
```

and the turtle will not move because it is already pointing at that angle.

SETHEADING (abbreviated SETH) sets the turtle's heading to an absolute direction, as shown above.

Type

```
SETX 100
```

and the turtle should move 100 steps to the right of home.

Type

```
SETY -60
```

and the turtle should move 60 turtle steps straight down from its previous position (its x co-ordinate hasn't changed).

The HOME position has an x co-ordinate of 0. Positive x co-ordinates are on the right of the screen and negative x co-ordinates are on the left of the screen.

The HOME position has a y co-ordinate of 0. Positive y co-ordinates are on the top half of the screen and negative y co-ordinates are on the bottom half.

Type

```
SETX -50
SETY  70
```

The turtle should now be at the top left hand side of the screen.

If the pen were not up, the turtle would leave a trail. These commands with the pen up are very useful if you wish to start a drawing in a specific position on the screen.

The commands HEADING, XCOR and YCOR enable you to find out where the turtle is on the screen, and the direction in which it is pointing.

Type

```
(PRINT HEADING XCOR YCOR)
```

The computer should respond

```
180. -50.  70.
```

which is the turtle's current heading and position (providing that you haven't changed it meanwhile).

Type

```
SETHEADING HEADING-10
SETX XCOR+20
```

By looking at the screen you will see that the turtle's position and heading have changed. Type

```
(PRINT HEADING XCOR YCOR)
```

The computer should now respond with

```
170  -30 70
```

which are the turtle's new co-ordinates.

EXERCISE

3.7 What are the biggest x and y co-ordinates that you can have without the screen wrapping round?

MORE ON MARCHING MEN

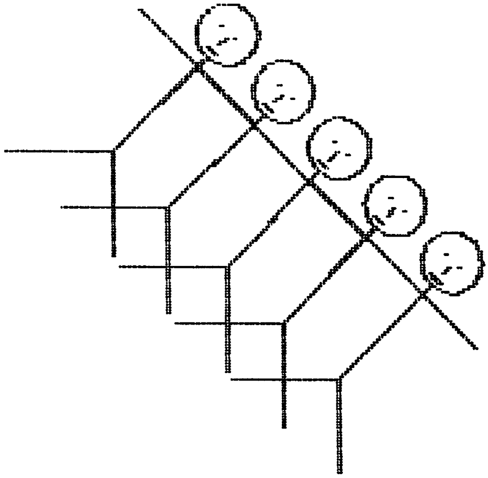
We could change the MARCH procedure as follows:

```
TO MARCH2
REPEAT 5 [MAN SLIDEL 30]
END
```

Now type

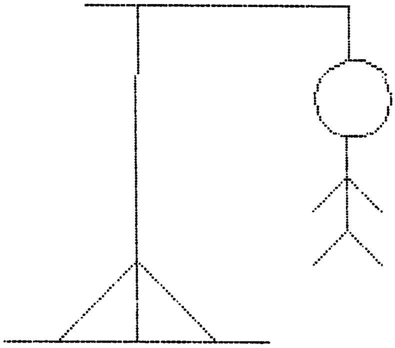
```
SETH 135
PU
SETY -60
SETH 45
PD
MARCH2
```

Use MARCH2 and the computer should respond



A HANGMAN PROJECT

If you have ever played "Hangman", you will enjoy the following drawing.



Type

```
TO A
PU LT 90 FD 25 RT 90 PD FD 100 BK 125
END
```

Define A and then type

```
TO B
RT 90 FD 50 BK 100 FD 50 LT 90 FD 125
END
```

Define B, and then use A and B to see what they draw. Then go on to the next procedure. Type

```
TO C
LT 90 FD 20 BK 100 FD 80 LT 90 FD 95
END
```

Define C, then use A, B and C. Continue in this way until all the sub-procedures have been completed.

```
TO D
RT 45 FD 43 BK 43
END

TO E
LT 90 FD 43 BK 43 LT 135
FD 95 RT 90 FD 80
END
```

```
TO F
RT 90 FD 20 LT 90
END

TO G
POLY 5 20 RT 90 PU FD 28 PD
END

TO H
FD 35 BK 20
END

TO I
LT 45 FD 20 BK 20
END

TO J
RT 90 FD 20 BK 20 LT 45 FD 20
END

TO K
LT 45 FD 20 BK 20
END

TO L
RT 90 FD 20 HT
END

TO POLY :S :A
REPEAT 360/:A [FD :S RT :A]
END
```

Finally let's put it all together. Type

```
TO HANGMAN
PU BK 20 PD A B C D E F G H I J K L
END
```

Now define the procedure and type HANGMAN.

EXERCISES

3.8 Draw a rocket.

3.9 Using the circle drawing version of POLY for the wheels, draw a car.

IDEAS INTRODUCED IN THIS CHAPTER

Experimenting with procedures

Larger projects in turtle geometry

A system of co-ordinates

SUMMARY OF COMMANDS INTRODUCED IN THIS CHAPTER

MIT LOGO	APPLE LOGO	DESCRIPTION
SHOWTURTLE,ST	SHOWTURTLE,ST	show turtle on screen
HIDETURTLE,HT	HIDETURTLE,HT	turtle figure hidden
CLEARSCREEN,CS	CLEAN	clears screen but doesn't move the turtle
HOME	HOME	set the turtle to the centre of the screen facing up
PENUP,PU	PENUP,PU	stop leaving turtle trail
PENDOWN,PD	PENDOWN,PD	start leaving trail
SETHEADING, SETH	SETHEADING, SETH	set turtle heading in absolute degrees
SETX	SETX	set turtle x coordinate
SETY	SETY	set turtle y coordinate
HEADING	HEADING	output current turtle heading
XCOR	XCOR	output current turtle x coordinate
YCOR	YCOR	output current turtle y coordinate

4 RECURSION AND MORE TURTLE PROJECTS

AN INTRODUCTION TO RECURSION - COUNTING BACKWARDS

The following procedure is designed to count backwards. Carefully type and define it. (Remember MIT LOGO is on the left and APPLE LOGO is on the right.)

```
TO BACKWARDS :NUMBER             TO BACKWARDS :NUMBER
IF :NUMBER = 0 STOP              IF :NUMBER = 0 [STOP]
PRINT :NUMBER                    PRINT :NUMBER
BACKWARDS :NUMBER - 1            BACKWARDS :NUMBER - 1
END                              END
```

Now use it to count backwards from 10.

```
BACKWARDS 10
```

The computer should respond

```
10
9
8
7
6
5
4
3
2
1
```

Simple enough! Now let's look carefully at how the procedure works.

The first line gives the procedure a title, BACKWARDS, and names an input, NUMBER, for the procedure. The second line tests whether the value of NUMBER is zero, and stops the procedure if it is. The third line prints the (non-zero) value of NUMBER.

The fourth line is the really interesting one. You will remember that procedures can use (or call) other procedures. It is also possible for a procedure to call itself, and this is what is happening here. This process is called recursion.

A good way to think about it is this:

> If a genie appears and offers you three wishes
> you should use your third wish to wish for
> three more wishes
>
> (Abelson, 1982, p.33).

At the fourth line of BACKWARDS, a new copy of the BACKWARDS procedure is made in the computer's memory. However this time it is used with an input whose value is 1 less than the original input. The second copy of the procedure is executed in the same way as the original, so at its fourth line a third copy is set up with a still smaller input value, and so on.

Eventually the newest copy will have an input of zero. This copy of the procedure will stop at its second line, as a result of the IF :NUMBER = 0 test being true, and no further numbers will be printed.

We could modify the BACKWARDS procedure to count back by 2's. Type and define a new procedure to do this, using BACKWARDS as a guide.

```
TO BACK2 :NUMBER              TO BACK2 :NUMBER
IF :NUMBER = 0 STOP           IF :NUMBER = 0 [STOP]
PRINT :NUMBER                 PRINT :NUMBER
BACK2 :NUMBER - 2             BACK2 :NUMBER - 2
END                           END
```

Use the new procedure to count back from 10.

```
BACK2 10
```

The computer should respond

```
10
8
6
4
2
```

Try the procedure with some other inputs including some odd numbers. It will only stop properly if you use even numbers. If you input an odd number you will need to stop the procedure yourself by pressing CTRL-G.

A RECURSIVE SPIRAL PROCEDURE

Now we will try a recursive procedure for drawing a spiral. Type and define

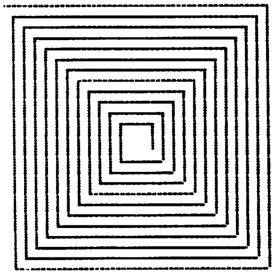
```
TO SPI :LENGTH :ANGLE :INC
IF :LENGTH > 100 STOP
FD :LENGTH
LT :ANGLE
SPI :LENGTH+:INC :ANGLE :INC
END
```

```
TO SPI :LENGTH :ANGLE :INC
IF :LENGTH > 100 [STOP]
FD :LENGTH
LT :ANGLE
SPI :LENGTH+:INC :ANGLE :INC
END
```

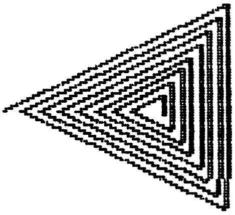
Note the recursive call in the fifth line of the SPI procedure. In this call the value of LENGTH is changed each time by the value of INC. First we will experiment with the procedure; then we will examine exactly how it works.

Use the procedure with sets of three inputs, for example

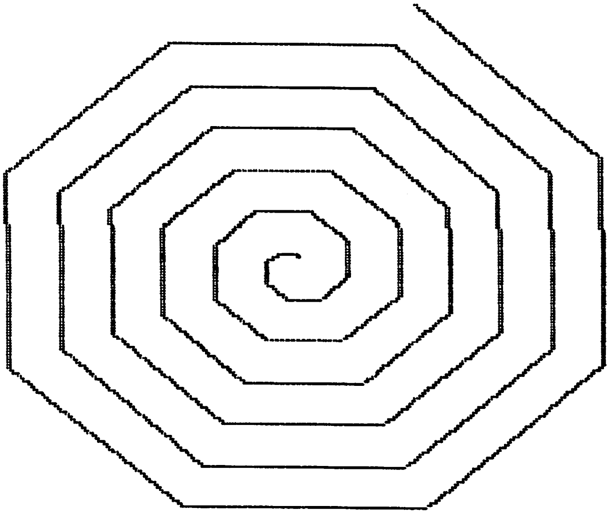
```
SPI 10 90 2
```



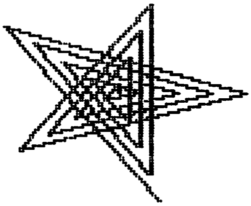
SPI 10 120 4



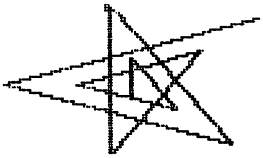
SPI 1 45 2



SPI 5 144 5



SPI 20 216 10



Now let us look in detail at how SPI works with inputs of 10, 90 and 2.

1. We use SPI with three inputs, LENGTH (10), ANGLE (90) and INC (2).

2. The IF command is used to test whether the value of LENGTH is greater than 100. If it is, the procedure stops. LENGTH has a value of 10, so the procedure goes on.

3. The turtle moves ahead LENGTH steps, so it goes forward 10 steps.

4. The turtle turns left ANGLE (90) degrees.

5. SPI is called again with a new value of LENGTH. LENGTH has the value of INC added to it, so the new value of the first input is 12.

6. LENGTH is tested to see whether it exceeds 100 yet. The new

value for LENGTH is 12, so it doesn't.

7.The turtle now moves forward 12 steps.

8. The turtle turns 90 degrees.

9. SPI is called again with new values of its inputs, they are now 14, 90 and 2. In this way, the sides of the figure get longer and longer and a spiral shape emerges. Finally, when LENGTH has a value exceeding 100, the turtle will stop.

BIGGER SPIRALS

The SPI procedure stops when the length of a side in the spiral is 100 turtle steps long. Change the procedure to draw bigger spirals. Make it keep drawing until the side length reaches 350 turtle steps.

```
TO SPI :LENGTH :ANGLE :INC          TO SPI :LENGTH :ANGLE :INC
IF :LENGTH > 350 STOP               IF :LENGTH > 350 [STOP]
FD :LENGTH                          FD :LENGTH
LT :ANGLE                           LT :ANGLE
SPI :LENGTH+:INC :ANGLE INC         SPI :LENGTH+:INC :ANGLE :INC
END                                 END
```

Use the procedure again with the inputs that you used with the earlier SPI procedure. Some of the spirals will look nicer if you eliminate the several text lines at the bottom of the screen and allow the drawing the full screen area. The command for this is

```
FULLSCREEN
```

Although you cannot see the commands you type after FULLSCREEN, continue as before. Any error messages will in fact appear at the bottom of the screen. When you want to restore the screen format with the text lines at the bottom, type the command

```
SPLITSCREEN
```

MORE ON SCREEN MODES

Both FULLSCREEN and SPLITSCREEN have abbreviations.

The abbreviation for SPLITSCREEN is CTRL-S, and the abbreviation for FULLSCREEN is

```
CTRL-F                    CTRL-L
```

FULLSCREEN and SPLITSCREEN are convenient to use within a program.

The control keys are easy to use from the keyboard since they can be typed at any time.

When LOGO finds an error in a procedure, it will stop and an error message will be printed on the screen. In fullscreen mode you may not see the message at all, and in splitscreen mode some of it may be obscured. A further mode called text mode allows the full screen to be devoted to text. Now use the CTRL key to get

```
CTRL-T
```

and the screen should go into text mode. Any error (or any other message) will be completely shown on the screen. If you had a drawing on the screen before you used CTRL-T, don't worry; all of these commands are reversible.

All of the above commands can be used in any combination to cycle between the various modes. No pictures or messages will be lost.

EXERCISES

4.1 Experiment with various combinations of side and angle in SPI.

4.2 Modify SPI by taking the line

```
IF :LENGTH >    etc.
```

out of the procedure. Call this procedure SPI2. Now use SPI2 with the various combinations of side and angle that have been suggested, and some of your own.

MORE RECURSIVE PROCEDURES

In Chapter 2 the POLY procedure was used.

```
TO POLY :SIDE :ANGLE
REPEAT 360/:ANGLE [FD :SIDE RT :ANGLE]
END
```

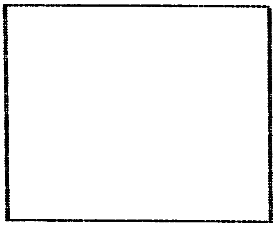
A more general form of POLY can be written using recursion. Call this procedure POLY1. Type

```
TO POLY1 :SIDE :ANGLE
FD :SIDE
RT :ANGLE
POLY1 :SIDE :ANGLE
END
```

Notice the similarity between this and the SPI procedure. Define POLY1 and type

```
POLY1 100 90
```

The computer should respond with a square (but to stop it you will need CTRL-G).

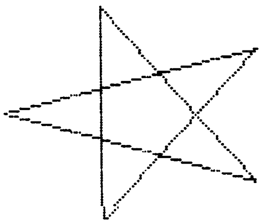


Likewise you can produce all the regular polygons. Side 100, angle 120 will produce a triangle. Side 1 and angle 1 will produce a circle; so will side 1, angle 2; so will side 10, angle 10 and so on.

Try

```
POLY1 100 144
```

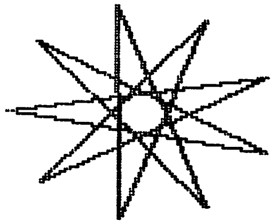
The computer should respond



Type

```
POLY1 100 160
```

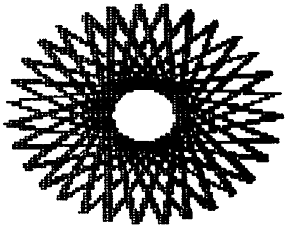
The computer should respond



Type

```
POLY1 100 151
```

The computer should respond



EXERCISES

4.3 Use POLY1 to produce some regular polygons such as octagons and nonagons.

4.4 Experiment with some sides and angles of your own. Use particularly some that don't divide evenly into 360 degrees or a multiple of 360 (such as 720 and 1080).

4.5 Explain what happens when the angle you use does divide exactly into one of these numbers, and what happens when the angle doesn't.

4.6 What is different about the figures drawn when the angle is greater than 90 degrees?

GETTING ROUND CIRCLES

On some monitor screens circular figures, squares, etc, will appear squashed up. A command exists for you to compensate for this if it is causing you problems.

```
.ASPECT 1.0                SETSCRUNCH 1.0
```

When you first load LOGO the computer is automatically set up as if you had typed

```
.ASPECT .8                 SETSCRUNCH .8
```

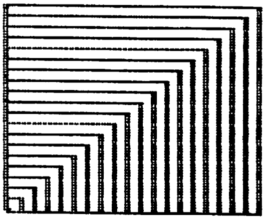
The use of 1.0 will make the turtle's vertical steps larger in relation to its horizontal steps, so a circle will be stretched upwards. You should choose a figure that suits your own monitor screen. This command must be used before, not after a drawing.

GROWING THINGS

Type these procedures to make a square grow.

```
TO GROWSQUARE :SIDE
SQUARE :SIDE
GROWSQUARE :SIDE+5
END

TO SQUARE :SIDE
REPEAT 4 [FD :SIDE RT 90]
END
```

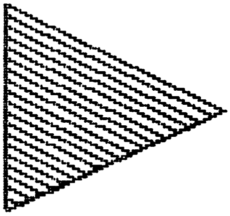
Define and use GROWSQUARE; the computer should respond with something like this



Now grow a triangle.

```
TO GROWTRIANGLE :SIDE
TRIANGLE :SIDE
GROWTRIANGLE :SIDE+5
END

TO TRIANGLE :SIDE
REPEAT 3 [FD :SIDE RT 120]
END
```

Use GROWTRIANGLE, and the computer should respond with something like this



STOPPING THESE GROWING THINGS

You will remember the way SPI was stopped with the IF command. Now modify GROWSQUARE and GROWTRIANGLE with a similar command in their second line.

```
IF :SIDE > 100 STOP          IF :SIDE > 100 [STOP]
```

You will probably want to experiment with the stopping value; try 350, 500 and 1000.

EXERCISES

4.7 GROWSQUARE and GROWTRIANGLE each grow by 5 turtle steps. Try them with other increments such as 1, 2 or 10.

4.8 Modify GROWSQUARE and GROWTRIANGLE so that the increment is a variable:

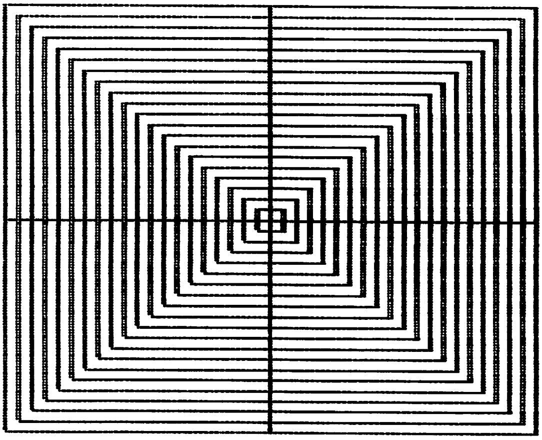
```
TO GROWSQUARE1 :SIDE :INC
```

SPINNING THINGS

Now we will spin the GROWSQUARE and GROWTRIANGLE drawings in various ways. Type

```
TO SPINGS
REPEAT 4 [GROWSQUARE 1 RT 90]
END
```

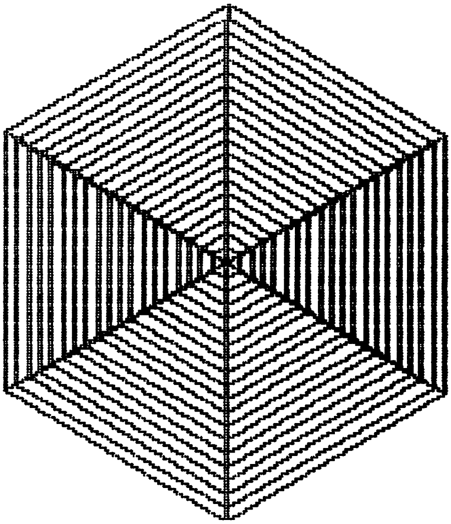
The computer should respond



Spin GROWTRIANGLE; try

```
TO SPINGT
REPEAT 6 [GROWTRIANGLE 1 RT 60]
END
```

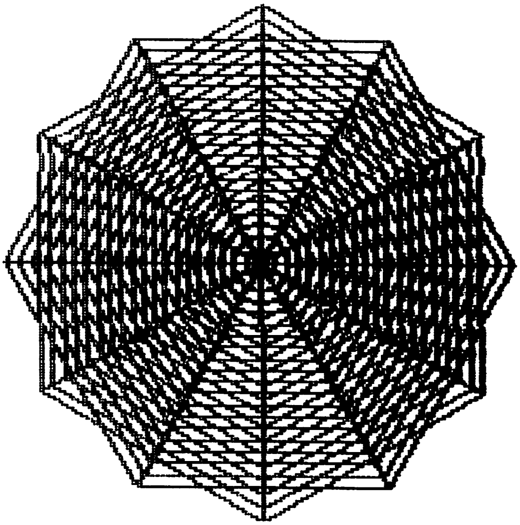
and the computer should respond



Now modify SPINGT to produce SPINGT1:

```
TO SPINGT1
REPEAT 12 [GROWTRIANGLE 1 RT 30]
END
```

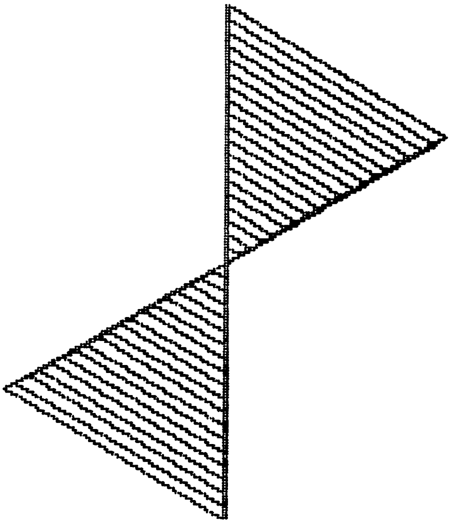
The computer should respond



Note that in the above examples the number in the REPEAT and the angle of the turn multiplied together give 360.

Now for a bow tie:

```
TO BOW
GROWTRIANGLE 1
RT 180
GROWTRIANGLE 1
END
```



SPINNING CIRCLES ROUND AND ROUND

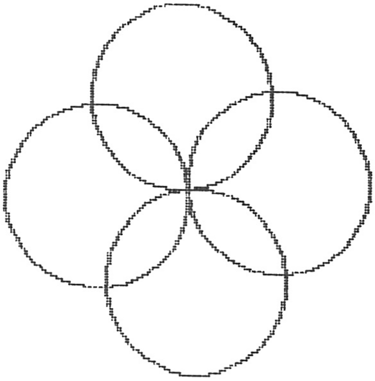
```
TO SPINCIRCLE :ANGLE
REPEAT 360/:ANGLE[CIRCLE RIGHT :ANGLE]
END

TO CIRCLE
REPEAT 36 [FD 6 RT 10]
END
```

Notice that CIRCLE has unusual forward and right amounts. The FD 6 is so the turtle will draw a fairly fast circle, and the right 10 will keep it small. (You could use a circle procedure with side 1 and angle 1 but this would be larger and take longer to print.)

Now type

```
SPINCIRCLE 90
```

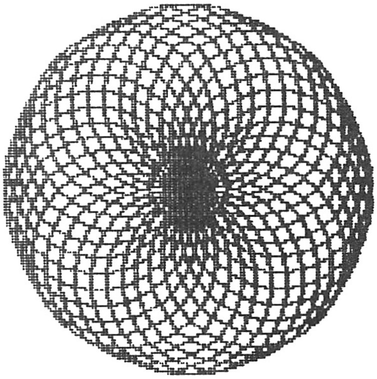
The computer should respond

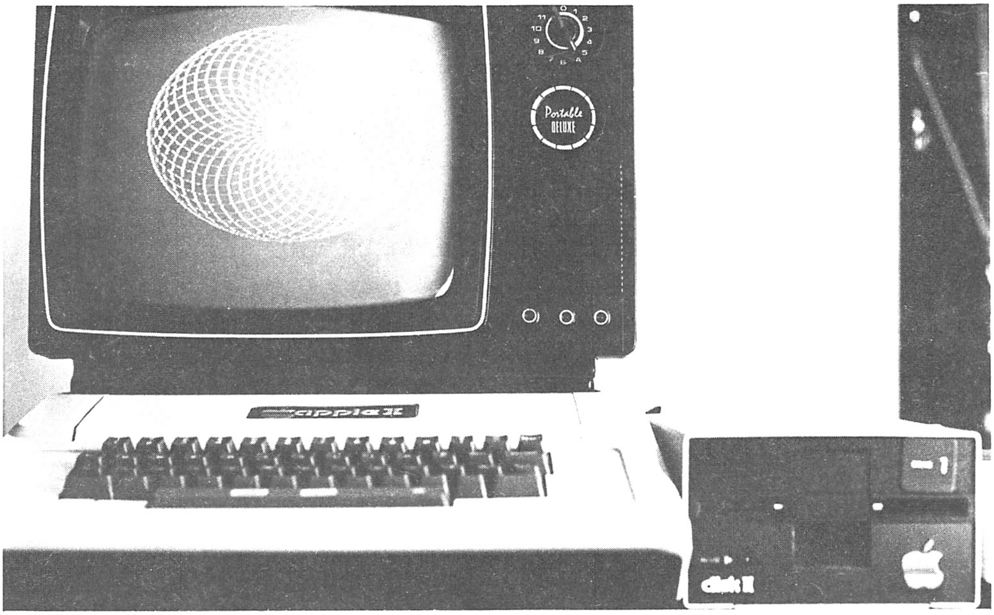


Try

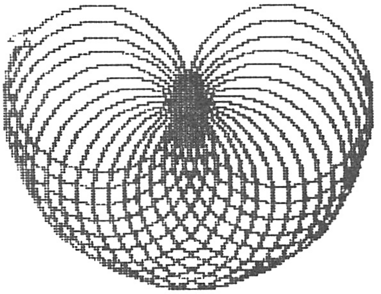
SPINCIRCLE 10

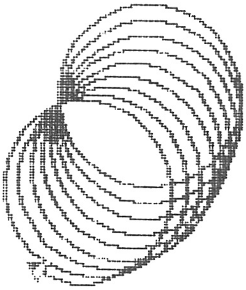
The computer should respond





If we stop SPINCIRCLE (using CTRL-G) at different stages before. it is complete we will get some unusual results.





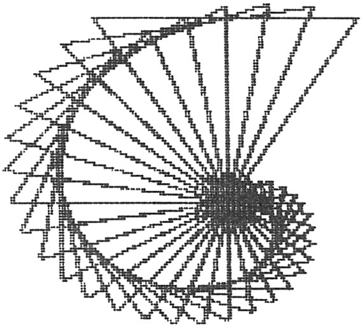
SPINNING SPIRALLING THINGS

You will need the TRIANGLE and SQUARE procedures in your workspace for the next projects. Type

```
TO SPITRI :SIDE :ANGLE :INC
TRIANGLE :SIDE
RT :ANGLE
SPITRI :SIDE+:INC :ANGLE :INC
END
```

Now try

```
SPITRI 1 10 3
```



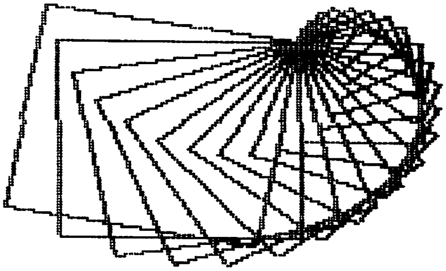
In SPITRI the variable ANGLE is the amount the triangle is turned after it is drawn, and INC is the amount that the triangle grows between drawings.

We can spin a square in the same way.

```
TO SPISQUARE :SIDE :ANGLE :INC
SQUARE :SIDE
RT :ANGLE
SPISQUARE :SIDE+:INC :ANGLE :INC
END
```

Type

```
SPISQUARE 1 10 5
```



EXERCISES

4.9 Place a stop into SPISQUARE and SPITRI after 100 turtle steps.

4.10 Try other inputs with SPISQUARE and SPITRI.

4.11 Write a set of procedures to spin any polygon in a similar way to SPINCIRCLE. HINT: you will need the form of POLY that was in Chapter 2, and another variable (not ANGLE in POLY) to turn between uses of POLY.

MAKING TUNNELS

This set of procedures starts off drawing a large circle; then it increments the angle in the circle procedure to draw a smaller circle. This process goes on until the procedure stops. The result is like a tunnel.

Type

```
    TO TUNNEL :ANGLE                 TO TUNNEL :ANGLE
    IF :ANGLE > 9 STOP               IF :ANGLE > 9 [STOP]
    RCIRCLE :ANGLE                   RCIRCLE :ANGLE
    TUNNEL :ANGLE+1                  TUNNEL :ANGLE+1
    END                              END
```

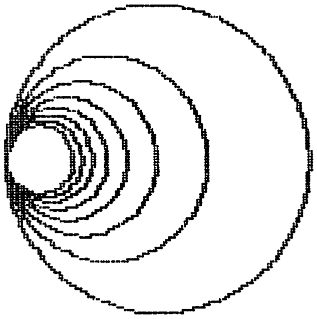
```
TO RCIRCLE :ANGLE
REPEAT 360/:ANGLE [FD 2 RT :ANGLE]
END
```

both LOGO'S

Type

```
TUNNEL 2
```

and the computer should respond

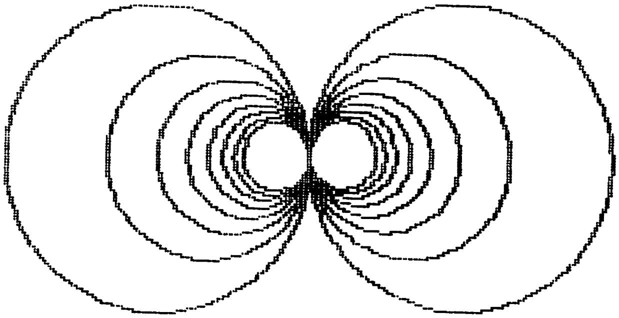


MAKING EYES

We will join two tunnels together so they look like eyes. Type

```
TO EYES
TUNNEL 2
RT 180
TUNNEL 2
END
```

Now use EYES and the computer should respond



MORE ABOUT THE STOP COMMAND

Whenever we have used STOP, the computer has responded with the ? prompt and we have gone on to the next thing we want to do. STOP doesn't stop the computer. It stops the procedure that is being used, and returns the computer to the next line of the procedure that called the procedure that is stopped. In the example above, this means that when TUNNEL stops, the computer goes back to the next line of EYES, which is RT 180. It then carries out the RT 180 command. When TUNNEL next stops the computer also returns to EYES at the next line, but this is END which also has the effect of returning the computer to the next line of whichever procedure called EYES. No procedure called EYES so the ? prompt is printed and the computer waits for the next command.

TESTING WITH THE IF COMMAND

In the procedure TUNNEL and various others in this chapter we have used an IF command. The examples that have been used show that when a condition is true, the command that is part of the IF is obeyed; otherwise the computer goes on to the next line of the procedure. Look at the TUNNEL procedure.

```
TO TUNNEL :ANGLE              TO TUNNEL :ANGLE
IF :ANGLE > 9 STOP            IF :ANGLE > 9 [STOP]
RCIRCLE :ANGLE                RCIRCLE :ANGLE
TUNNEL :ANGLE +1              TUNNEL :ANGLE +1
END                           END
```

When we use TUNNEL 2, the first time through the procedure ANGLE has the value 2. The IF tests whether ANGLE is greater than 9. The result is FALSE (it is not greater than 9) so the computer goes to the next line. Finally TUNNEL will be called with ANGLE having a value of 9 and the result of the test will be TRUE, so the LOGO command on the same line as the IF will be executed, in this case STOP.

Another way of writing TUNNEL would be

```
TO TUNNEL :ANGLE             TO TUNNEL :ANGLE
TEST :ANGLE > 9              TEST :ANGLE > 9
IFTRUE STOP                  IFTRUE [STOP]
IFFALSE RCIRCLE :ANGLE       IFFALSE [RCIRCLE :ANGLE]
TUNNEL :ANGLE + 1            TUNNEL :ANGLE + 1
END                          END
```

This works the same way as the IF but it is divided into two parts. The TEST returns either TRUE or FALSE. The command on the line IFTRUE will be executed if TEST returned TRUE, and the line IFFALSE will be executed if the result of the test was FALSE. IFTRUE can be abbreviated IFT and IFFALSE can be abbreviated IFF.

Let us now look at another example.

```
TO SELECT :NUMB              TO SELECT :NUMB
IF :NUMB=1 CIRCLE            IF :NUMB=1 [CIRCLE]
IF :NUMB=2 SQUARE            IF :NUMB=2 [SQUARE]
END                          END
```

If we type SELECT 1, the procedure CIRCLE will be used. When CIRCLE is finished, the computer will return to the next line of SELECT. Since NUMB has the value 1, the next line will be ignored and SELECT will end. Let us now examine a slightly altered form of SELECT.

```
TO SELECT :NUMB              TO SELECT :NUMB
IF :NUMB=1 CIRCLE            IF :NUMB=1 [CIRCLE]
SQUARE                       SQUARE
END                          END
```

This time we have left out the second IF. If we say SELECT 1, the CIRCLE will be printed and the computer will return to SELECT at the next statement, so SQUARE will be printed (not what we wanted). If we say SELECT 2 then the SQUARE will be printed, which is what we wanted. A procedure that only sometimes does what we want is not very useful.

A better SELECT procedure would be

```
TO SELECT :NUMB                TO SELECT :NUMB
IF :NUMB=1 CIRCLE STOP         IF :NUMB=1 [CIRCLE STOP]
IF :NUMB=2 SQUARE STOP         IF :NUMB=2 [SQUARE STOP]
PRINT [ERROR IN SELECT]        PRINT [ERROR IN SELECT]
END                            END
```

This version is the best of the lot because

1. It stops after the selection is finished.

2. Only one selection is printed.

3. It prints an error if you make a wrong selection. The PRINT command is examined in detail in Chapter 5. For now we can say that it prints the message in square brackets on the screen, so now you have a way of printing your own error (or any other) messages.

4. It will only get to PRINT if an incorrect selection is made.

More than one command can be part of the IF (or IFTRUE or IFFALSE). In MIT LOGO everything until the next RETURN is part of the IF (even if it goes over the line). In APPLE LOGO everything in the square brackets is part of the IF (you cannot have RETURN in the brackets).

We can use equals (=), greater than (>) or less than (<) in IF or TEST statements

Throughout the text, both IF and TEST with IFTRUE and IFFALSE will be used, so you should get used to both ways of testing.

EXERCISES

4.12 Rewrite and use the final form of SELECT with TEST. Make sure you have suitable square and circle procedures.

4.13 Add POLY1 to SELECT, and verify that it works with all possibilities.

CREATING WALLPAPER PATTERNS

When the turtle goes over the edge of the screen it returns at the opposite point and continues drawing. We saw this effect with earlier spiralling patterns. It can be used to produce patterns similar to wallpaper or textile designs.

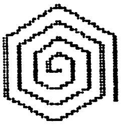
Type

```
TO SPIR :SIDE :ANGLE :INC          TO SPIR :SIDE :ANGLE :INC
FD :SIDE                           FD :SIDE
LT :ANGLE                          LT :ANGLE
IF :SIDE < 0 STOP                  IF :SIDE < 0 [STOP]
SPIR :SIDE-:INC :ANGLE :INC        SPIR :SIDE-:INC :ANGLE :INC
END                                END
```

Define and use the procedure. Try

```
SPIR 25 60 1
```

The computer should respond



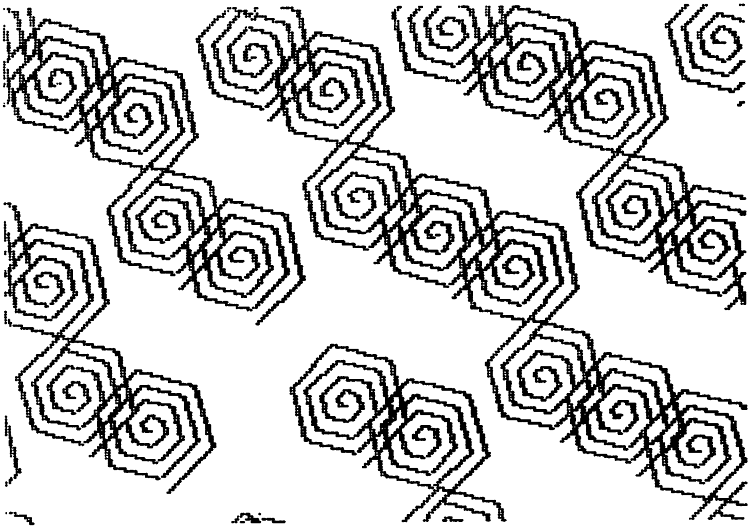
Notice that this procedure spirals in, not out. It can be used as the basis of a pattern. (So could a circle, a square, a spiral that spiralled out, or just about any figure). Type

```
TO WALLPAPER :INC                TO WALLPAPER :INC
IF :INC=7 RT 45                  IF :INC=7 [RT 45]
SPIR 25 60 1                     SPIR 25 60 1
PU                               PU
SETHEADING 0                     SETHEADING 0
RT 45                            RT 45
FD 55                            FD 55
PD                               PD
IF :INC < 0 STOP                 IF :INC <0 [STOP]
WALLPAPER :INC-1                 WALLPAPER :INC-1
END                              END
```

Define the procedure and type

```
WALLPAPER 7
```

The computer should respond with something like this.



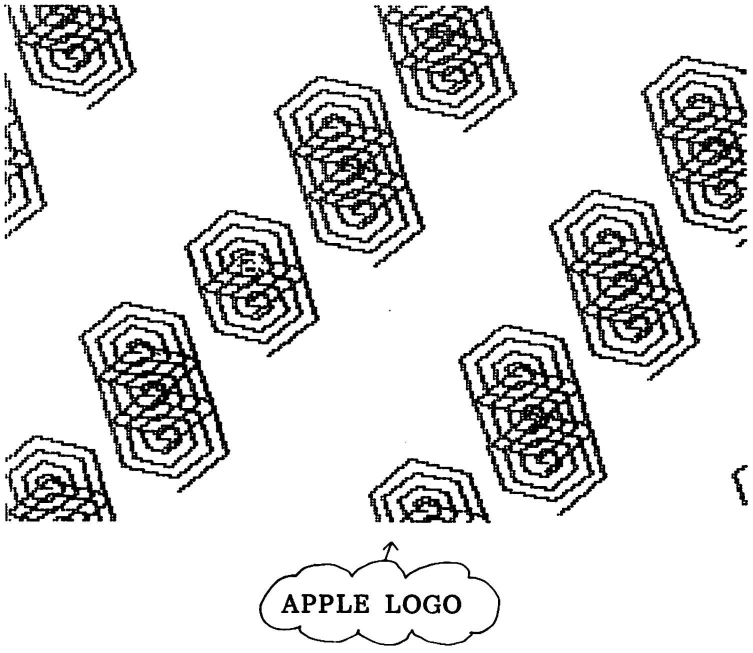
MIT LOGO

If we now edit WALLPAPER to WALLPAPER1 with this change to the second line

```
IF :INC=21 RT 45                    IF :INC=21 [RT 45]
```

the computer should produce the following drawing if you type

```
WALLPAPER1 21
```



EXERCISES

4.13 Try altering the angle and number of steps in WALLPAPER to get different effects.

4.14 Make your own patterns with squares and triangles.

4.15 Patience and experimentation will produce some beautiful and interesting patterns, limited only by your imagination. Try various stars, reduce the size of the windmill below, change the windmill pattern so it resembles a flower, and so on.

SPINNING WINDMILLS

Type

```
TO WINDMILL :ANGLE
MAKE "TURN 0
REPEAT 360/:ANGLE [BLADE :ANGLE]
END
```

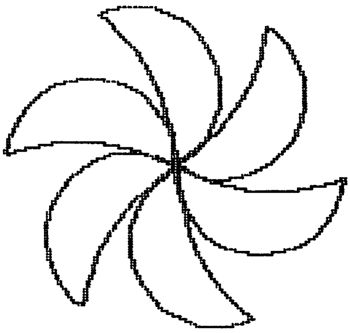
the MAKE command is discussed in Chapter 5

```
TO BLADE :ANGLE
CRESCENT
SETHEADING 0
MAKE "TURN :TURN+:ANGLE
RT :TURN
END

TO CRESCENT
REPEAT 20 [FD 5 LT 9]
LT 110
REPEAT 13 [FD 5 RT 4]
END
```

Define these procedures and type

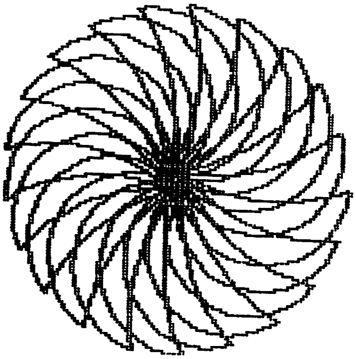
```
WINDMILL 60
```

The computer should respond



Now try

WINDMILL 17

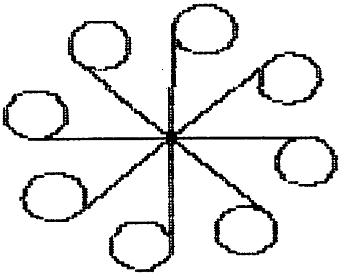


EXERCISES

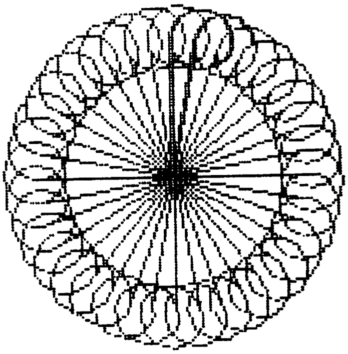
4.15 Try some other inputs to WINDMILL: 1, 90, 5, and so on .

4.16 Write a procedure to draw this (call it WHEEL).

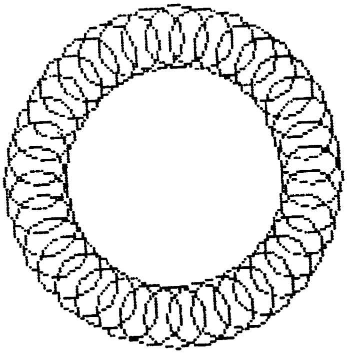
Hint: step forward 50 steps, use RCIRCLE 10, then go back 50 steps. Now use a repeat line like that in SPINCIRCLE to spin this. The above example is WHEEL 45.



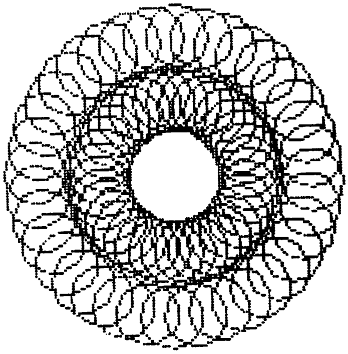
4.17. Use WHEEL 10 to get this:



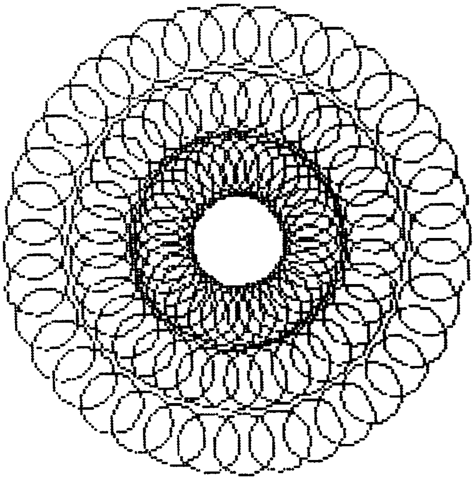
4.18 Use PENUP to remove the line each time; call this new procedure WHEEL1. Don't forget to put the pen down before using RCIRCLE.



4.19 Now go forward 25 steps, use RCIRCLE 10, go forward another 25 steps and use RCIRCLE 10 again; then go back 50 steps. Spin it as before. Call this new procedure WHEEL2.



4.20 Work this one out for yourself.



IDEAS INTRODUCED IN THIS CHAPTER

Recursion

Stopping recursive procedures

Testing variables in procedures

Screen modes

SUMMARY OF COMMANDS INTRODUCED IN THIS CHAPTER

MIT LOGO	APPLE LOGO	DESCRIPTION
FULLSCREEN	FULLSCREEN	full screen drawings
SPLITSCREEN	SPLITSCREEN	split drawing and text
CTRL-F	CTRL-L	full screen from keyboard
CTRL-S	CTRL-S	split screen from keyboard
CTRL-T	CTRL-T	text screen only
PRINT	PRINT	print data on screen (see Chapter 5)
IF	IF	test and carry out if true
TEST	TEST	test true or false
IFTRUE, IFT	IFTRUE, IFT	carry out if test true
IFFALSE, IFF	IFFALSE, IFF	carry out if test false
STOP	STOP	stop procedure and return to calling procedure
> < =	> < =	used in IF and TEST for greater, less and equals
MAKE	MAKE	assign name to variable (see Chapter 5).
.ASPECT	SETSCRUNCH	set vertical aspect ratio

5 NAMING THINGS AND DOING ARITHMETIC

WORKING WITH NUMBERS

LOGO allows you to use the computer like a very powerful pocket calculator. Try it. Type

```
PRINT 219.305 + 41762.9
```

The response should be

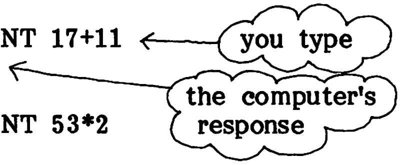
```
41982.2
```

With LOGO you can carry out the following mathematical operations:

```
+ addition,
- subtraction,
* multiplication,
/ division.
```

Try the following examples

```
PRINT 17+11
28

PRINT 53*2
106

PRINT 8/4
2.

PRINT 8/3
2.66667
```



LOGO can only display the result of a calculation to seven decimal places.

If we multiply 888888*888888, the answer will be too large to display in the normal way. Try it. Type

```
PRINT 888888*888888
7.90121E11
```

This answer is correct, but it is displayed in a new way. The number before the E is called the mantissa (i.e. 7.09121 in the example). The number after is called the exponent (i.e. 11 in the example). To find out what the number is in normal notation, you move the decimal point in the mantissa to the right, the number of places shown in the exponent. The number

```
7.90121E11
```

becomes

```
790121000000.0
```

a very large number indeed.

A similar thing happens for very small numbers.

```
0.0000130021
```

becomes

```
1.30021N05
```

when printed by the computer.

The N or negative exponent indicates that the decimal point in the mantissa is to be moved to the left. The number of places is indicated by the exponent (i.e. 5 in the example).

More than one arithmetic operation can be carried out on one line.

```
PRINT 3*3*3
27
```

Additions, subtractions, multiplications, and divisions can be mixed together.

```
PRINT 3*3+6*2/3+1
14.
```

Before attempting any further examples you should understand the way in which the computer works out a problem like the one above.

First, all the multiplication and division operations are carried out, working from left to right. Then all the additions and subtractions are done, also working from left to right.

Step 1. Multiply 3 by 3 to get 9.
Step 2. Multiply 6 by 2 to get 12.
Step 3. Divide 12 by 3 to get 4.
Step 4. Add the result of step 1 and step 3 to get 13.
Step 5. Add 1 to the result of step 4 to get 14.

Brackets can be used to assist in making the order of operations clearer. Type

```
PRINT ((3*3)+((6*2)/2)+1)
```

The computer should respond

```
16.
```

Suppose you wanted to divide 10 by the sum of 2 and 3. The answer you would expect is 2. Try typing

```
PRINT 10/2+3
```

The computer responds

```
8.
```

This is not what we expected. The problem is that, working from left to right, the division is carried out first. This gives a result of 5. The 3 is added to this to give a final result of 8.

The correct way to write the problem would have been

```
PRINT 10/(2+3)
2.
```

In this case, the calculation in the bracket is carried out first. The result is 5, which is subsequently divided into 10 to produce a final result of 2.

Now try

```
PRINT 5*(6+4/2)
40.
```

In this example, the division inside the brackets is carried out first, then its result added to 6, and the sum is multiplied by 5 to produce the final result. You should note that the multiplication sign (*), must be typed before the bracket. This is not the case when similar statements are written without a computer.

PRINTING TEXT

A more interesting use for PRINT is to display text on the screen.

Type

```
PRINT [THIS IS A TEST]
```

The computer should respond

```
THIS IS A TEST
```

The result of the PRINT command is that the message in the square brackets is printed out.

Anything inside square brackets like this is called a list.

```
PRINT [TODAY IS A GREAT DAY]
TODAY IS A GREAT DAY

PRINT [HELLO BETTY I AM YOUR COMPUTER]
HELLO BETTY I AM YOUR COMPUTER
```

It would be nice to print several different things on one line, for example:

```
PRINT [THE RESULT IS] 4*4
```

The answer we would like to get is

```
THE RESULT IS 16
```

Type

```
PRINT [THE RESULT IS] 4*4
```

and see what the computer response is. The outcome is unexpected and takes two lines.

To get the correct response, we must place brackets right around the PRINT statement. This is necessary whenever more than one item is to be printed. Type

```
(PRINT [THE RESULT  IS] 4*4)
```

The response should be

```
THE RESULT IS 16
```

The PRINT command can also be used to print out the values of variables.

```
MAKE "ANSWER 4
PRINT :ANSWER
4

MAKE "ANSWER "WOOW
PRINT :ANSWER
WOOW

MAKE "ANSWER  4
(PRINT [RESULT IS] :ANSWER)
RESULT IS 4
```

The PRINT statement can also be used to print blank lines. This is sometimes useful when you want to provide gaps between different parts of a printout.

```
PRINT []
```

This statement prints a list that contains nothing, and has the effect of printing a blank line.

NAMING THINGS

Naming is a very important part of LOGO. We have already seen that the titles we give to procedures become new commands in the language. For example, once a square-drawing procedure, titled SQUARE, was defined, we could use it by typing the command SQUARE. This enables us to build our own versions of the language, containing, as well as the original LOGO primitives, commands chosen and designed by us.

We have also given names to inputs for procedures. To enable the square-drawing procedure to make squares of various sizes, we might use, for example, an input named SIDELENGTH in the procedure definition.

As well as these two kinds of naming, LOGO enables us to give names to items of data in our programming. These data items might be numbers, words or lists.

We will begin by giving a name to a number. Imagine we have a procedure in which we want to handle only numbers less than 20. We want to give the name UPPERLIMIT to the number 20. Type

```
MAKE "UPPERLIMIT 20
```

The number 20 is stored in the computer, and the name UPPERLIMIT is stored in association with that value.

The number 20 is called the THING of the name UPPERLIMIT. Type

```
PRINT THING "UPPERLIMIT
```

The computer should respond

```
20
```

Now let us name a thing which is itself a word. Type

```
MAKE "COLOR "GREEN
```

This gives the name COLOR to the word GREEN. Type

```
PRINT THING "COLOR
```

and the computer should respond

```
GREEN
```

Similarly we can name a list. Type

```
MAKE "MESSAGE [THREE BLIND MICE]
```

The list [THREE BLIND MICE] now has the name MESSAGE. Typing

```
PRINT THING "MESSAGE
```

should result in

```
THREE BLIND MICE
```

being printed. Note that if you type

```
PRINT "MESSAGE
```

the computer response is

```
MESSAGE
```

THING can be abbreviated using a colon sign. (You will remember using this notation in defining procedures with inputs.) Thus THING "MESSAGE and :MESSAGE are equivalent. Type

```
PRINT :MESSAGE
```

and the response should be

```
THREE BLIND MICE
```

Now type

```
MAKE "COLOR "BLUE
PRINT "COLOR
```

The computer should respond

```
COLOR
```

This is because we have asked it to print the name COLOR.

Now type

```
PRINT THING "COLOR
```

The computer should respond

```
BLUE
```

This is because BLUE is the THING or value associated with the name COLOR. If we type

```
PRINT :COLOR
```

then the computer will respond

```
BLUE
```

At any time we can change the THING (or value) of a name by using another MAKE statement, such as

```
MAKE "COLOR "5
```

Not only have we changed the value of the THING of COLOR, but now the value is a number.

The MAKE statement is not the only way of changing the value of the THING of a name. The following two procedures are equivalent.

```
TO POLYSPI :SIDE :ANGLE
FORWARD :SIDE
RIGHT :ANGLE
POLYSPI :SIDE+2 :ANGLE
END
```

```
TO POLYSPI :SIDE :ANGLE
FORWARD :SIDE
RIGHT :ANGLE
MAKE "SIDE :SIDE+2
POLYSPI :SIDE :ANGLE
END
```

The differences are in the ways in which the value of SIDE (its THING) is changed, but the effect is the same. In the following chapters both methods are used.

MORE ABOUT THE MAKE COMMAND

We can carry out complex calculations with MAKE, using the rules that we have already applied for arithmetic.

Type

```
MAKE "RESULT  4*(3+2)/(3+2)+1
PRINT :RESULT
```

The result should be

```
5.
```

Now type

```
MAKE "AREA  3*3
MAKE "VOL :AREA * 3
(PRINT [VOLUME IS ] :VOL
```

Whoops; we haven't put in the closing bracket in the PRINT statement, and we will get an error message. We can fix that up by retyping the line

```
(PRINT [VOLUME IS ] :VOL )
```

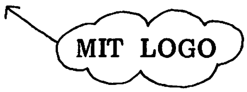
We should now get the correct answer.

```
VOLUME IS 27
```

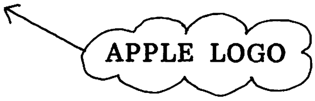
PRINTING ON THE SAME LINE

The PRINT command causes a new line to be printed each time. Sometimes you may want to print several results on the same line. For this you use

PRINT1



TYPE



Type

```
TO PRINTIT                       TO PRINTIT
PRINT1 [TODAY IS TUESDAY]        TYPE [TODAY IS TUESDAY]
PRINT [AND IT IS SUNNY]          PRINT [AND IT IS SUNNY]
END                              END
```

Now define and use the procedure

```
TODAY IS TUESDAYAND IT IS SUNNY
```

Note the lack of a space between the two former lists. We will discuss a solution for this problem in Chapter 6.

After the PRINT1 (or TYPE) command, the computer does not go to the next line. After the PRINT command it will go to the next line.

GETTING DATA INTO THE COMPUTER

Data values can be entered directly from the keyboard.

Type

```
MAKE "MYNAME REQUEST          MAKE "MYNAME READLIST
```

The cursor should blink on and off asking you to enter data. You can then type some data. Finish the data that you have typed by pressing the RETURN key. The data will be stored with the name "MYNAME. We could then display on the screen anything that was typed in. For example, type

```
PRINT :MYNAME
```

The result should be

```
IRVING
```

which is the data that you typed.

We could also have used REQUEST (or READLIST) in a PRINT statement.

```
(PRINT [HELLO I'M] REQUEST)    (PRINT [HELLO I'M] READLIST)
```

The result should be

```
HELLO I'M JENNIFER
```

which is the name that you typed.

The REQUEST (or READLIST) statement passes on (outputs) whatever is typed in, to the operation using it (say a MAKE or PRINT). The data that REQUEST (or READLIST) gives to any operation is in the form of a list (i.e. it has square brackets around it, even though you didn't type them).

What this means is that if you want to type numbers and use them in calculations, you must use the following kind of statement. (Chapter 6 discusses the use of lists in greater detail.)

```
MAKE "NUMBER FIRST REQUEST      MAKE "NUMBER FIRST READLIST
```

Now type

```
45
```

Then type

```
(PRINT [DOUBLE IS] :NUMBER*2)
```

The computer should respond

```
DOUBLE IS 90
```

The FIRST command has the effect of removing from the list the number that you entered. You will have to use this type of statement whenever you want to enter numbers into the computer with REQUEST (or READLIST) and use them in calculations. (The use of FIRST is explained in more detail in Chapter 6.)

One way of inputting numbers without having to use FIRST each time is to write a new procedure, READNUM. Type

```
TO READNUM                      TO READNUM
OUTPUT FIRST REQUEST            OUTPUT FIRST READLIST
END                             END
```

We can now use this procedure instead of REQUEST or READLIST whenever we want to read in a number from the keyboard. The OUTPUT statement causes the result of FIRST REQUEST (or FIRST READLIST) to be passed back to the procedure or statement that called READNUM. Now type

```
MAKE "NUMBER READNUM
```

Then type

```
45

(PRINT [DOUBLE IS] :NUMBER*2)
```

The computer should respond

```
DOUBLE IS 90
```

We will use READNUM whenever we wish to enter a number from the keyboard. The use of READNUM illustrates how in LOGO we can define new commands to suit our own purposes.

USING CALCULATIONS IN PROCEDURES

We can write procedures to carry out arithmetic just as we can use them for turtle drawings.

Type

```
TO ADD :NUMBER
PRINT :NUMBER
ADD :NUMBER+1
END
```

When we have written and defined the procedure, we can use it.

Now type

```
ADD 1
```

The result should be

```
1
2
3
4
5
etc.
```

This procedure will run for a very long time, and you will probably want to stop it with a CTRL-G command.

ADD works by starting with an input, which in our case was 1. The first line of ADD prints this input (1). The second line calls ADD again but with a new input (:NUMBER+1) which will have the value 2. This will be printed and ADD will be called again with a new input (:NUMBER+1) which will have the value 3, and so on.

The ADD procedure is similar in many ways to the BACKWARDS procedure we used for counting backwards in Chapter 4. Notice though that ADD doesn't have within it a stopping command as BACKWARDS does.

A DOUBLING PROCEDURE

A procedure to double numbers could be written as follows:

```
TO DOUBLE :NUMBER
MAKE "NUMBER 2*:NUMBER
PRINT :NUMBER
DOUBLE :NUMBER
END
```

After the first time through the procedure, we call DOUBLE again, but this time with twice the value that we started with. You may be surprised to find that after having doubled the number 128 times (in about 15 seconds), the computer will signal that the result is too large and the procedure will stop.

EXERCISES

5.1 Try the doubling procedure and see what happens.

5.2 Change the procedure so that it multiplies the number by itself each time and see what happens. Call this procedure MULT. Try MULT with inputs 0, then 1, then 2. What do the different results mean?

ANOTHER DOUBLING PROCEDURE

Another variation would be to write a procedure which requests the numbers to be doubled, one at a time, from the keyboard.

```
TO DOUBLE1
PRINT [INPUT THE NEXT NUMBER]
MAKE "NUMBER READNUM
(PRINT [DOUBLE IS] 2*:NUMBER)
DOUBLE1
END
```

When you use DOUBLE1, the following dialogue should take place:

You type

```
DOUBLE1
```

The computer should respond

```
INPUT THE NEXT NUMBER
```

You type

```
4
```

The computer responds

```
DOUBLE IS 8
INPUT THE NEXT NUMBER
```

and so on.

The trouble with this procedure is that it never stops. It will keep asking for the next number until you stop it with CTRL-G, or take the drastic step of turning the computer off. Let us arrange that if you input a zero you want the procedure to stop. Edit DOUBLE1 to produce this new version. (Remember that MIT LOGO is on the left and APPLE LOGO is on the right.)

```
TO DOUBLE1                        TO DOUBLE1
PRINT [INPUT THE NEXT NUMBER]     PRINT [INPUT THE NEXT NUMBER]
PRINT [TO STOP ENTER 0]           PRINT[TO STOP ENTER 0]
MAKE "NUMBER READNUM              MAKE "NUMBER READNUM
TEST :NUMBER =0                   TEST :NUMBER=0
IFTRUE PRINT [FINISHED] STOP      IFTRUE [PRINT [FINISHED] STOP]
(PRINT [DOUBLE IS] 2*:NUMBER)     (PRINT [DOUBLE IS] 2*:NUMBER)
DOUBLE1                           DOUBLE1
END                               END
```

The new procedure DOUBLE1 will go through the following steps.

1. The first two PRINT statements will cause the following messages to be displayed on the screen.

```
INPUT THE NEXT NUMBER
TO STOP ENTER 0
```

2. The system will then wait until you enter a number to double and press RETURN.

3. The data that you have input will be given the name NUMBER.

4. The TEST statement will then test the value of the name NUMBER for zero.

5. If it is zero the IFTRUE statement will be obeyed, FINISHED will be displayed on the screen and the procedure will stop.

6. If the data weren't zero then the answer will be displayed on the screen.

7. The call to procedure DOUBLE1 will be obeyed, and the whole operation will commence again.

8. Note that the brackets around the third PRINT statement are there because we are trying to print DOUBLE IS and the value of 2*:NUMBER both on the same line.

If you don't put in a number at all, but instead try and double something non-numeric like BETTY, you will get an error message, and the procedure will stop. Try it. The computer will respond with a message like

```
* DOESN'T LIKE BETTY AS INPUT
```

A LOGO STOPWATCH

We can write in LOGO a procedure that counts seconds. We could slow it down to run at the speed of a clock. We would then have a stopwatch that could be started and stopped to time events. Type

```
TO STOPWATCH :COUNT
PRINT :COUNT
MAKE "COUNT :COUNT+1
STOPWATCH :COUNT
END
```

STOPWATCH could also have been written

```
TO STOPWATCH :COUNT
PRINT :COUNT
STOPWATCH :COUNT+1
END
```

Both procedures are identical in the way they work, the only difference being the way that :COUNT is modified. Now define the procedure with CTRL-C. Type

STOPWATCH 0 but don't type RETURN yet.

Keep your finger on the RETURN key and depress it the instant you want the clock to start.

To stop the procedure, keep one finger on CTRL and another one poised to press G the instant you want to stop the clock.

When you run STOPWATCH you will find three obvious faults, and perhaps some others as well.

1. STOPWATCH goes too fast.

2. When it gets to 60 seconds it just goes on to 61 seconds, rather than counting in minutes and seconds.

3. Having the numbers run up the screen is annoying.

The first problem can be solved by putting in a delay each time we go through the procedure. If the time for the computer to go through the procedure once, plus the delay we introduce, adds up to 1 second then our stopwatch will be calibrated correctly.

A suitable method for introducing the delay is to use a REPEAT statement.

```
REPEAT 100[]
```

This will repeat what is in the square brackets 100 times. The fact that there is nothing in them won't worry us since it will simply create a small delay each time. The number of repeats that we have chosen might however be wrong; it might be too small or too large.

Now edit the procedure to get:

```
TO STOPWATCH :COUNT
MAKE "COUNT :COUNT+1
REPEAT 100 []
PRINT :COUNT
STOPWATCH :COUNT
END
```

You will now have to run STOPWATCH a number of times, checking it with your wristwatch and changing the value in REPEAT each time until you have it operating accurately.

USING RANDOM NUMBERS

When you throw a die there is an equal chance that any one of the faces will come up. The same occurs with a coin; if you toss a coin it has the same chance of coming up heads or tails. If you carry out, say, 50 tosses you will see that the number of heads will be nearly the same as the number of tails. Computers can produce numbers which simulate the random nature of throwing dice and coins.

To use the random number generator in LOGO we can type

```
PRINT RANDOM 100
```

The response might be

```
32
```

What is happening is that the computer is producing a number chosen at random between 0 and 99 inclusive. Each number between 0 and 99 has the same chance of being chosen. Try using RANDOM 100 several times to see what happens.

```
PRINT RANDOM 100
29

PRINT RANDOM 100
81

PRINT RANDOM 100
5
```

If we used RANDOM 6, then we would get numbers between 0 and 5.

Computer generated random numbers can be used in games and simulations. For example, we can write a procedure that simulates the tossing of a coin. This procedure will tell us whether we have thrown a head or tail.

```
TO TOSS                          TO TOSS
MAKE "FLIP RANDOM 2              MAKE "FLIP RANDOM 2
TEST :FLIP=0                     TEST :FLIP=0
IFTRUE  PRINT [HEADS] STOP       IFTRUE  [PRINT [HEADS] STOP]
PRINT [TAILS] STOP               PRINT [TAILS] STOP
END                              END
```

When we use the procedure TOSS we will get either a HEAD or a TAIL. It runs in the following fashion:

1. RANDOM is asked to pass on to the MAKE statement a number which is either 0 or 1. We shall call 0 a HEAD and 1 a TAIL.

2. The name FLIP is assigned to the random number produced from RANDOM.

3. We test FLIP to see if it is 0.

4. If it is TRUE that FLIP is 0, the computer prints out that it is a HEAD, and stops.

5. If it is FALSE that it is a 0, then TAILS is printed and the procedure then stops. Note that we don't have to specifically ask IFFALSE, because what is not true must be false.

RANDOMIZING RANDOM NUMBERS
(APPLE LOGO USERS SKIP THIS SECTION)

For a single session with MIT LOGO, RANDOM will produce randomly chosen numbers. Each time you start MIT LOGO, though, you will get the same sequence of numbers. To get a different sequence each time you must type RANDOMIZE before you use RANDOM the first time.

```
RANDOMIZE
```

COUNTING HEADS AND TAILS

We might want to write a set of procedures to count the numbers of heads and tails that occur in, say, 100 tosses. We would need to carry out the following steps:

1. Set a counter to zero for the running total of heads.

2. Do the same thing for counting tails.

3. Toss the coin (generate the random number), 100 times.

For each toss the procedure would have to determine whether the outcome was a head or a tail, and add 1 to either the heads sub-total or to the tails sub-total.

4. Following the 100 tosses, the computer would print out the numbers of heads and tails.

Type

```
TO COUNTTOSS
MAKE "HEADS 0
MAKE "TAILS 0
REPEAT 100 [ADDUP]
(PRINT [THE NUMBER OF HEADS WAS] :HEADS)
(PRINT [THE NUMBER OF TAILS WAS] :TAILS)
END
```

Now type the ADDUP procedure. (Remember MIT LOGO is on the left and APPLE LOGO is on the right.)

```
TO ADDUP                            TO ADDUP
MAKE "FLIP RANDOM 2                 MAKE "FLIP RANDOM 2
TEST :FLIP = 0                      TEST :FLIP = 0
IFTRUE MAKE "HEADS :HEADS+1         IFTRUE [MAKE "HEADS :HEADS+1]
IFFALSE MAKE "TAILS :TAILS+1        IFFALSE [MAKE "TAILS :TAILS+1]
END                                 END
```

Define the procedure and use it.

```
COUNTTOSS
```

The computer will respond with an answer that may be different each time you run COUNTTOSS.

For example

```
THE NUMBER OF HEADS WAS 53
THE NUMBER OF TAILS WAS 47
```

1. The procedure COUNTTOSS reflects the earlier outline of the problem. We could write COUNTTOSS without worrying about the details of ADDUP until we came to write it later.

2. The LOGO segment:

```
TEST :FLIP=0                      TEST :FLIP=0
IFTRUE MAKE "HEADS :HEADS+1       IFTRUE [MAKE "HEADS :HEADS+1]
IFFALSE MAKE "TAILS :TAILS+1      IFFALSE[MAKE "TAILS :TAILS+1]
END                               END
```

could have been replaced by:

MIT LOGO

```
TEST :FLIP=0
IFTRUE MAKE "HEADS :HEADS + 1 STOP
MAKE "TAILS :TAILS+1
END
```

APPLE LOGO

```
TEST :FLIP = 0
IFTRUE [MAKE "HEADS :HEADS + 1 STOP]
MAKE "TAILS :TAILS + 1
END
```

The STOP statement stops the current procedure and returns to the calling procedure (COUNTTOSS).

EXERCISES

5.3 Modify the stopwatch procedure so that it counts in minutes and seconds.

5.4 Write a set of procedures to throw two dice and to print out the two numbers that come up.

5.5 Modify exercise 5.4 so that it will throw the dice 100 times and print out how many times each of the faces 1 through 6 comes up. Hint: you should be able to use the procedure you wrote in exercise 5.4.

STANDARD NUMERIC PROCEDURES

The following procedures are provided as part of the LOGO system for carrying out numeric operations. If you don't know about trigonometry and square roots then you should skip to the end of the chapter.

The Cosine of an Angle:

This procedure outputs the cosine of an angle expressed in degrees.

```
COS :ANGLE
```

Now type

```
PRINT COS 45
```

The computer will respond

```
.707107
```

The Sine of an Angle:

SIN outputs the sine of an angle expressed in degrees.

Type

```
PRINT SIN 90
```

The computer will respond

```
1.
```

The Tangent of an Angle:

The Tangent of an angle can be found from the procedure:

```
TO TAN :ANGLE
PRINT (SIN :ANGLE)/(COS :ANGLE)
END
```

Getting the Integer Part from a Real Number:

This procedure outputs the whole number part of a real number; in other words, everything after the decimal point is lost.

```
INTEGER :NUMBER            INT :NUMBER
```

Now type

```
PRINT INTEGER 10.1         PRINT INT 10.1
```

The result should be

```
10                         10
```

Also try typing

```
PRINT INTEGER -10.9        PRINT INT -10.9
```

The result should be

```
     -10                          -10
```

Finding the Square Root:

The SQRT procedure outputs the square root of its input. Type

```
PRINT SQRT 4
```

The result should be

```
2.
```

Now type

```
PRINT SQRT 5
```

The result should be

```
2.23607
```

The radius of a circle can be found if we know its area. A procedure for doing this follows.

```
TO GETRADIUS :AREA
OUTPUT SQRT :AREA/3.14159
END
```

OUTPUT causes the result to be passed back to the PRINT command

To use the procedure we could type

```
PRINT GETRADIUS 10
```

The computer should respond with the radius

```
1.78412
```

The Arctangent of an Angle:

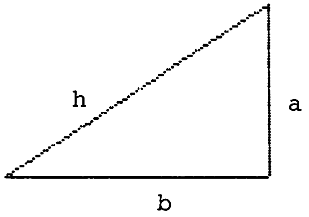
This command outputs the arctangent of two sides of a triangle in degrees.

```
     ATAN :A :B                   ARCTAN :A/:B
```

Note that a complete list of mathematical functions is contained in Appendix A.

A MATHEMATICAL PROJECT

Pythagoras showed us that the length of the longest side of a right-angled triangle can be found by:



$$h^2 = a^2 + b^2$$

A LOGO procedure to calculate the length of the longest side of a right-angled triangle could be written this way. Type it.

```
TO PYTHAG1 :A :B
MAKE "H SQRT :A*:A + :B*:B
PRINT :H
END
```

To use the procedure type

```
PYTHAG1 3 4
```

The computer should respond

```
5.
```

Another way of writing this procedure would be to type

```
TO PYTHAG2 :A :B
OUTPUT SQRT :A*:A + :B*:B
END
```

PYTHAG2 is more general than PYTHAG1 since, because of the OUTPUT command, the result of PYTHAG2 can be passed to another procedure (such as PRINT) and used. This is demonstrated in the next section.

A MORE ELABORATE TRIANGLE CALCULATION PROCEDURE

We will write a procedure called CALCSIDE which reads the lengths of two sides of a right-angled triangle from the keyboard, then calculates and prints the length of the hypotenuse. CALCSIDE will keep doing this until a side of zero length is read for the first number.

```
TO CALCSIDE                          TO CALCSIDE
PRINT1 [FIRST SIDE?]                 TYPE [FIRST SIDE?]
MAKE "A READNUM                      MAKE "A READNUM
TEST :A =0                           TEST :A =0
IFTRUE PRINT [END] STOP              IFTRUE [PRINT [END] STOP]
PRINT1 [SECOND SIDE?]                TYPE [SECOND SIDE?]
MAKE "B READNUM                      MAKE "B READNUM
PRINT1 [HYPOTENUSE]                  TYPE  [HYPOTENUSE]
PRINT PYTHAG2 :A :B                  PRINT PYTHAG2 :A :B
CALCSIDE                             CALCSIDE
END                                  END
```

Now type

```
CALCSIDE
```

The computer should respond

```
FIRST SIDE ?3
SECOND SIDE ?4
```

the data that you typed

The computer will now calculate the length of the third side

```
HYPOTENUSE5
FIRST SIDE?0
END
```

you type

DRAWING OUR TRIANGLE

The real power of using OUTPUT to return a result to a calling procedure is that we can use the result any way we want to.

If we modify CALCSIDE by including a call to a new procedure TRIDRAW then we can draw the resulting triangle on the screen. Edit CALCSIDE to produce:

```
TO CALCSIDE1                       TO CALCSIDE1
PRINT1 [FIRST SIDE?]               TYPE [FIRST SIDE?]
MAKE "A READNUM                    MAKE "A READNUM
TEST :A=0                          TEST :A=0
IFTRUE PRINT [END] STOP            IFTRUE [PRINT [END] STOP]
PRINT1 [SECOND SIDE?]              TYPE [SECOND SIDE?]
MAKE "B READNUM                    MAKE "B READNUM
MAKE "H PYTHAG2 :A :B              MAKE "H PYTHAG2 :A :B
DRAW                               CLEARSCREEN
(PRINT [HYPOTENUSE] :H)            (PRINT [HYPOTENUSE] :H)
RT 90                              RT 90
TRIDRAW :A :B :H                   TRIDRAW :A :B :H
END                                END
```

Now type:

```
TO TRIDRAW :A :B :H                TO TRIDRAW :A :B :H
FD :A                              FD :A
LT 90                              LT 90
FD :B                              FD :B
LT 180-ATAN :A :B                  LT 180-ARCTAN :A/:B
FD :H                              FD :H
END                                END
```

TRIDRAW works in the following way:

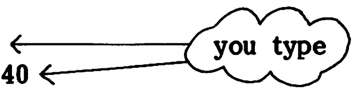
1. It takes three inputs (the sides of the triangle).

2. It draws the first two sides with the right angle between them.

3. To draw the final side it must turn through the outside angle. The inside angle is found by arctan :A/:B, therefore the outside angle is 180-arctan :A/:B.

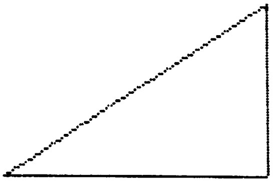
Now type

```
CALCSIDE1
FIRST SIDE?30
SECOND SIDE?40
```



HYPOTENUSE 50

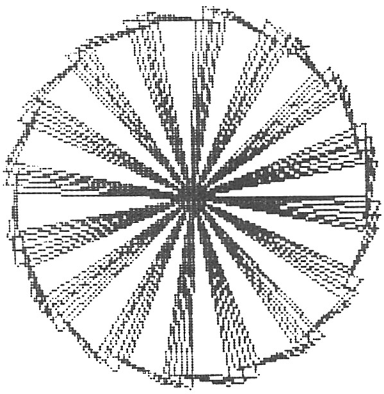
The computer should now draw the triangle.

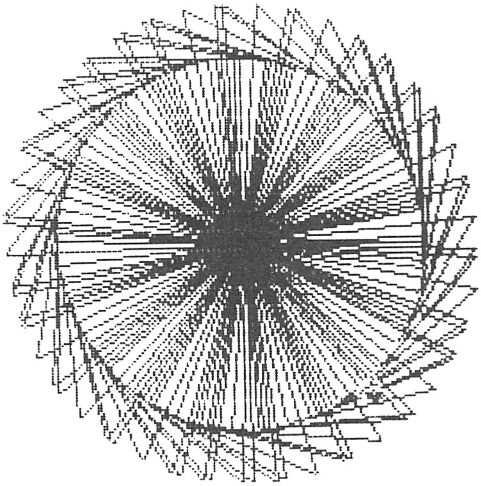


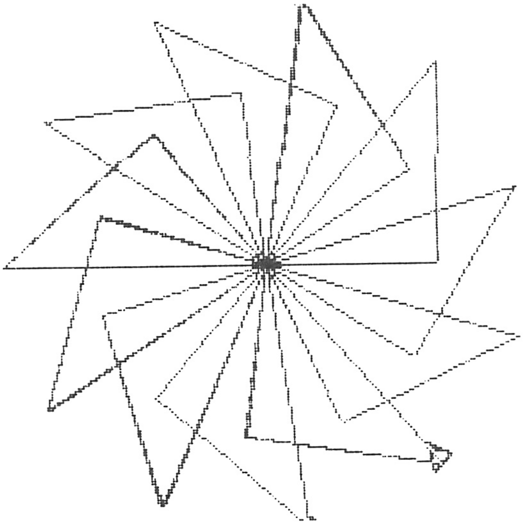
Change TRIDRAW as shown below to make some interesting drawings.

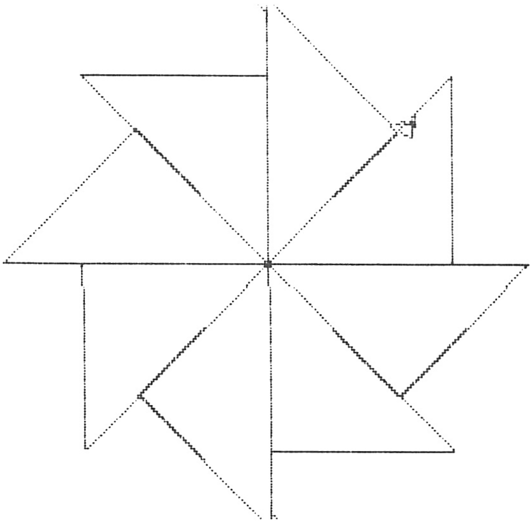
```
TO TRIDRAW :A :B :H          TO TRIDRAW :A :B :H
FD :A                        FD :A
LT 90                        LT 90
FD :B                        FD :B
LT 180-ATAN :A :B            LT 180-ARCTAN :A/:B
FD :H                        FD :A
TRIDRAW :A :B :H             TRIDRAW :A :B :H
END                          END
```

Now that you have made this change to TRIDRAW, use the CALCSIDE1 procedure and see what happens. The following pictures were made by trying various changes to CALCSIDE1 and TRIDRAW.









EXERCISES

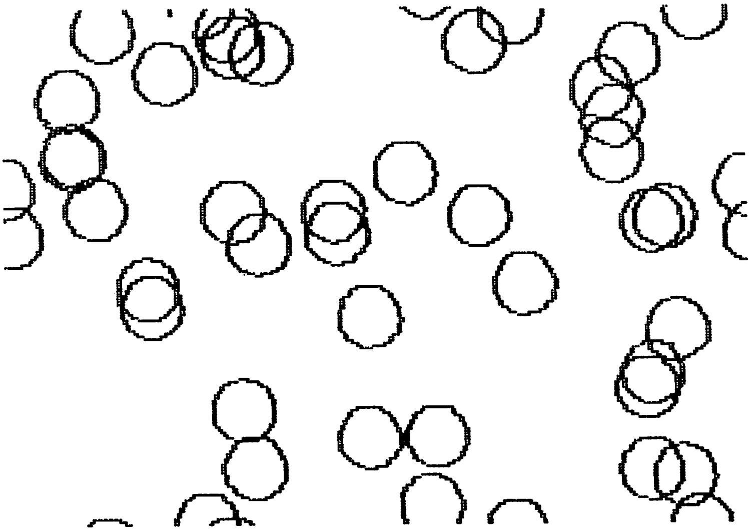
5.6 Write a procedure that outputs the area of a circle given the formula

AREA=3.1416*RADIUS*RADIUS

The procedure is to be called AREACIRCLE and should work as follows.

```
PRINT AREACIRCLE 1
THE AREA OF A CIRCLE RADIUS 1 IS 3.1416
```

5.7 Write a procedure called BUBBLES that draws small circles randomly around the screen like this.



Hint: use RCIRCLE from Chapter 4 with an input of 10 to draw the circles. BUBBLES will work as follows.

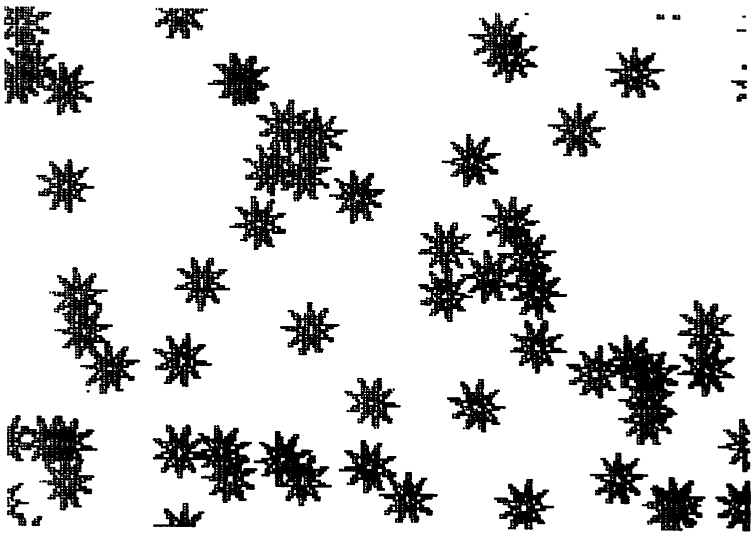
1. Draw a circle.

2. Turn the turtle right through an angle of between 0 and 360 degrees (RT RANDOM 361)

3. Move the turtle forward a random number of steps between 0 and 200 (FD RANDOM 201)

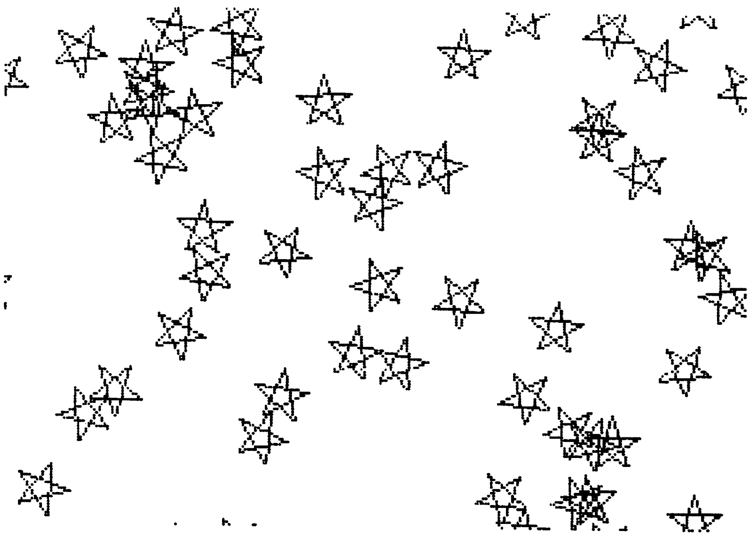
4. Make a recursive call to BUBBLES again.
BUBBLES will go on forever, so you will want to stop it with CTRL-G.

5.8 Modify BUBBLES and call it FLOWERS to produce something like this.

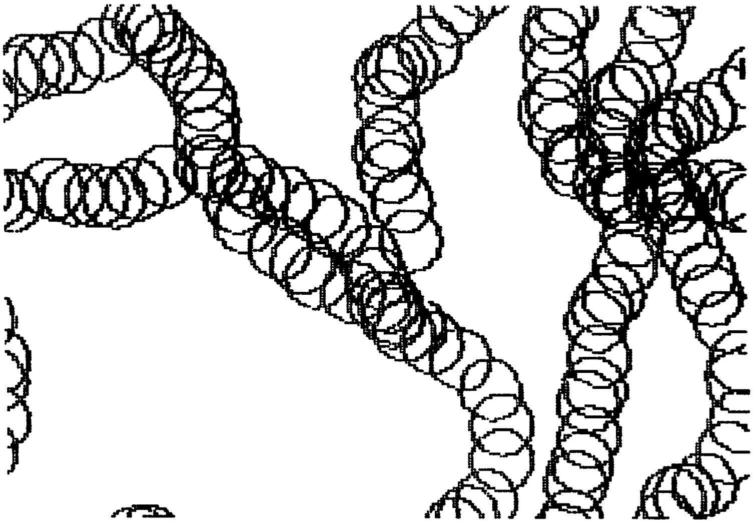


Hint: you will need to replace RCIRCLE with another procedure (a poly) that looks like a flower (have a look at Chapter 4). The poly must be one that stops when its shape is complete.

5.9 Now modify FLOWERS to produce STARS. Hint: try a different poly.



5.10 Now write SNAKE.



SNAKE works as follows.

1. Draw a circle (use RCIRCLE 10).

2. Select a random number between 0 and 1. If it is 0 then turn the turtle to the right a random number of degrees between 0 and 35. If it is 1 then turn the turtle to the left in the same way.

3. Move the turtle forward 10 steps.

4. Recursively call SNAKE.

5.11 Modify SNAKE to use small squares.

5.12 For the more mathematically inclined. A quadratic equation of the form

$$ax^2 + bx + c$$

can be solved by the formula

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Write a procedure to output the solution of a quadratic equation.

5.13 For the more adventurous. Write a set of procedures that allow you to test a student's arithmetic ability. Read from the keyboard the number of additions that the student requests. Then for each addition select two random numbers to add. Ask the student what the result is, and give him or her three tries to get it right. When the specified number of additions have been completed, print out the percentage of correct answers.

5.14 Modify 5.13 so that the student is asked whether he or she wants additions, subtractions, multiplications or divisions.

IDEAS INTRODUCED IN THIS CHAPTER

Arithmetic operations in LOGO

Exponential notation

Printing text and numbers

Naming things

Data input from the keyboard

Delays to control timing in procedures

Random numbers

Standard mathematical procedures in LOGO

Passing results from a sub-procedure back to its calling procedure

SUMMARY OF COMMANDS INTRODUCED IN THIS CHAPTER

MIT LOGO	APPLE LOGO	DESCRIPTION
PRINT,PR	PRINT,PR	print list, word or number
MAKE	MAKE	assign name to value
THING, :	THING, :	value of name
"	"	name (e.g. "YELLOW)
REQUEST	READLIST	read a list from the keyboard, terminate with RETURN
PRINT1	TYPE	same as print but remains on same line
OUTPUT	OUTPUT	stops procedure and returns value to calling procedure
+-/*	+-/*	arithmetic operations
RANDOM	RANDOM	output random number
COS	COS	cosine
SIN	SIN	sine
INTEGER	INT	output whole number
SQRT	SQRT	square root
ATAN	ARCTAN	arctangent

6 RECURSION AND LISTS

PROGRAMMING WITH LISTS

If we wished to display the message

WHAT IS TODAY'S DATE?

on the screen we could use a statement like

```
PRINT [WHAT IS TODAY'S DATE?]
```

The sentence enclosed in the square brackets is called a list.

The list is made up of elements. Each element may be a word, such as WHAT in the above example, a number or another list. Each of the elements of a list must be separated from the others by a space.

[17.3 21 [THIS IS A LIST] 27]

The list above consists of four elements. Three of these are numbers (17.3, 21, 27) and the fourth is another list (THIS IS A LIST), which itself contains four elements.

Now type

```
PRINT [17.3 21 [THIS IS A LIST] 27]
```

The computer should respond

```
17.3 21 [THIS IS A LIST] 27
```

LOGO contains a number of statements to allow us to examine and modify lists. Lists are particularly useful if we want to process English language text.

JOINING LISTS TOGETHER

The SENTENCE command can be used to join two lists. Type

```
     PRINT SENTENCE [TODAY'S DATE IS][THE THIRD OF JUNE]
```

SENTENCE outputs a single list, so we could have statements like

```
     MAKE "A SENTENCE [TODAY'S DATE IS][THE THIRD OF JUNE]
     PRINT :A
```

The computer should respond

```
                    TODAY'S DATE IS THE THIRD OF JUNE
```

Now type

```
                    MAKE "A [WHAT IS TODAY'S DATE?]
                    MAKE "B [THE THIRD OF JUNE]
                    PRINT SENTENCE :A :B
```

The computer should respond

```
                    WHAT IS TODAY'S DATE? THE THIRD OF JUNE
```

A more realistic example might involve having the date read from the keyboard each time we use the procedure. Type

```
TO PRINTDATE                       TO PRINTDATE
PRINT1 [INPUT THE DATE]            TYPE [INPUT THE DATE]
MAKE "DATE REQUEST                 MAKE "DATE READLIST
PR SENTENCE[THE DATE IS]:DATE PR SENTENCE[THE DATE IS]:DATE
END                                END
```

PRINTDATE can now be used by typing

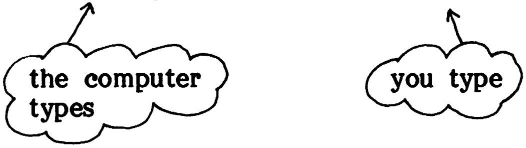
```
                    PRINTDATE
```

The computer will respond

```
                    INPUT THE DATE
```

You can now enter the date from the keyboard on the same line, and terminate it with RETURN; for example:

```
INPUT THE DATE THE THIRD OF JANUARY
```



The computer should respond

```
THE DATE IS THE THIRD OF JANUARY
```

JOINING MORE THAN TWO LISTS TOGETHER WITH SENTENCE

SENTENCE can join more than two lists if you enclose it and the lists in brackets. SENTENCE can also be abbreviated SE. Type

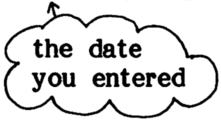
```
MAKE "DATE REQUEST          MAKE "DATE READLIST
```

Enter the date, terminated by RETURN
This can now be printed out

```
PR (SE [TODAY'S DATE IS] :DATE [AND IT'S WET])
```

The computer should respond

```
TODAY'S DATE IS THE FIRST OF FEBRUARY AND IT'S WET
```



LOOKING INTO LISTS

The statements FIRST, LAST, BUTFIRST and BUTLAST allow the elements of a list to be examined individually.

Type

```
MAKE "A [TODAY IS SUNNY]
PRINT FIRST :A
```

The computer should respond

```
TODAY
```

Type

```
PRINT LAST :A
```

The computer should respond

```
SUNNY
```

Type

```
PRINT BUTFIRST :A
```

The computer should respond

```
IS SUNNY
```

Type

```
PRINT BUTLAST :A
```

The computer should respond

```
TODAY IS
```

FIRST, LAST, BUTFIRST and BUTLAST by themselves don't allow us to look at the middle of a list. To do this we must write a procedure.

Type

```
TO EXAMINELIST :A              TO EXAMINELIST :A
TEST :A=[]                     TEST :A=[]
IFTRUE STOP                    IFTRUE [STOP]
PRINT FIRST :A                 PRINT FIRST :A
EXAMINELIST BUTFIRST :A        EXAMINELIST BUTFIRST :A
END                            END
```

To use this procedure, type

```
EXAMINELIST [TODAY IS SUNNY]
```

The computer should respond

```
TODAY
IS
SUNNY
```

Let us look at how this procedure works. Once you understand this you will be a long way towards understanding lists and how to use them.

1. The procedure begins with a list as input [TODAY IS SUNNY].

2. The first and second statements check whether the list is empty. This is done by checking the list that has been input against the empty list []. The first time through, the list that has been input will not be empty; it contains three elements [TODAY IS SUNNY].

3. The next statement prints out the first element of the list, TODAY.

4. We then call EXAMINELIST again but with a new input. The input is BUTFIRST :A. BUTFIRST outputs everything but the first element of a list, in this case :A. What happens then is that EXAMINELIST is called with a new list [IS SUNNY].

5. This will be continued. Each time EXAMINELIST will be called with everything but the first element it was called with last time. Finally EXAMINELIST will be called with an empty list []. This will be detected by TEST, and the procedure will STOP.

Instead of using FIRST and BUTFIRST we could have used LAST and BUTLAST. Type

```
TO EXAMINELIST1 :A              TO EXAMINELIST1 :A
TEST :A=[ ]                     TEST :A=[ ]
IFTRUE STOP                     IFTRUE [STOP]
PRINT LAST :A                   PRINT LAST :A
EXAMINELIST1 BUTLAST :A         EXAMINELIST1 BUTLAST :A
END                             END
```

Then use this procedure with the input [TODAY IS SUNNY]. Type

```
EXAMINELIST1 [TODAY IS SUNNY]
```

The computer prints

```
SUNNY
IS
TODAY
```

LAST takes the last element of a list, and BUTLAST takes everything but the last element of a list. Using these four statements (FIRST, LAST, BUTFIRST, BUTLAST), some interesting effects can be produced.

Type this

```
TO PRETTYLIST :A
TEST :A=[]
IFTRUE STOP
PRINT :A
PRETTYLIST BUTFIRST :A
END
```

```
TO PRETTYLIST :A
TEST :A=[]
IFTRUE [STOP]
PRINT :A
PRETTYLIST BUTFIRST :A
END
```

Now use this by typing

```
PRETTYLIST [TODAY IS SUNNY]
```

The computer prints

```
TODAY IS SUNNY
IS SUNNY
SUNNY
```

This procedure works in a similar way to EXAMINELIST.

1. The procedure starts off by seeing whether the list that has been input is empty.

2. If it's not empty the procedure prints the list that has been input. The first time through TODAY IS SUNNY will be printed.

3. A new copy of PRETTYLIST is called with everything but the first element,[IS SUNNY]. This will be printed and then PRETTYLIST will be called again with everything but the first element of the present list,[SUNNY].

4. The next time BUTFIRST [SUNNY] will cause PRETTYLIST to be called with the empty list []. This will be detected and the process will STOP.

EXERCISES

6.1 Using the same list [TODAY IS SUNNY], write down what would be the effect of replacing BUTFIRST with BUTLAST in PRETTYLIST. Now edit PRETTYLIST with this change and see what happens.

6.2 Try PRETTYLIST out with different lists, including some that go over one line.

(Remember that when you type a line that goes over the edge of the screen, you just keep typing. Don't use the RETURN key until the statement is finished. At the end of a screen line APPLE

LOGO places a ! character. This is to tell you that your statement has wrapped around to the next line on the screen.)

6.3 Edit PRETTYLIST and EXAMINELIST with different combinations of LAST, FIRST, BUTLAST and BUTFIRST, to see what happens.

A MORE RECURSIVE PROCEDURE

If we add a second PRINT statement to the PRETTYLIST procedure we get rather a different result. Edit PRETTYLIST so that it becomes.

```
TO PRETTYLIST :A                  TO PRETTYLIST :A
TEST :A=[]                        TEST :A=[]
IFTRUE STOP                       IFTRUE [STOP]
PRINT :A                          PRINT :A
PRETTYLIST BUTFIRST :A            PRETTYLIST BUTFIRST :A
PRINT :A                          PRINT :A
END                               END
```

the new statement

Now use PRETTYLIST. Type

```
PRETTYLIST [TODAY IS SUNNY]
```

The computer should respond

```
TODAY IS SUNNY
IS SUNNY
SUNNY
SUNNY
IS SUNNY
TODAY IS SUNNY
```

To explain what happens we must understand the principle of recursion. You should trace this explanation with Figure 6.1.

```
        PRETTYLIST (TODAY IS SUNNY)

 1      TO PRETTYLIST :A              :A IS (TODAY IS SUNNY)

 2      TEST :A = ( )                 FALSE
 3      IFTRUE STOP
 4      PRINT :A                      LINE 1                    TODAY IS SUNNY
 5      PRETTYLIST BUTFIRST :A        BUTFIRST :A IS (IS SUNNY)
 6      PRINT :A                      LINE 6                    TODAY IS SUNNY
 7      END

 8      TO PRETTYLIST :A              :A IS (IS SUNNY)
 9      TEST :A = ( )                 FALSE
10      IFTRUE STOP
11      PRINT :A                      LINE 2                    IS SUNNY
12      PRETTYLIST BUTFIRST :A        BUTFIRST :A IS (SUNNY)
13      PRINT :A                      LINE 5                    IS SUNNY
14      END

15      TO PRETTYLIST :A              :A IS (SUNNY)
16      TEST :A = ( )                 FALSE
17      IFTRUE STOP
18      PRINT :A                      LINE 3                    SUNNY
19      PRETTYLIST BUTFIRST :A        BUTFIRST :A IS ( )
20      PRINT :A                      LINE 4                    SUNNY
21      END

22      TO PRETTYLIST :A              :A IS ( )
23      TEST :A = ( )                 TRUE
24      IFTRUE STOP
25      PRINT :A
26      PRETTYLIST BUTFIRST :A
27      PRINT :A
28      END
```

Figure 6.1 PRETTYLIST

a) We call PRETTYLIST with an input [TODAY IS SUNNY]. The value of A is [TODAY IS SUNNY].

b) The value of A is not [] (empty) so the test is FALSE.

c) TODAY IS SUNNY is printed.

d) PRETTYLIST is called again, but with BUTFIRST :A , [IS SUNNY]

e) We now proceed to step 8, to a new copy of PRETTYLIST, with an input [IS SUNNY]. The value of A in this copy is [IS SUNNY].

f) Steps 9, 10, and 11 determine that the value of A is not [], so line 2 is printed IS SUNNY.

g) We now call a third copy of PRETTYLIST with an input of BUTFIRST :A, [SUNNY].

h) In this third copy the value of the input (A) is [SUNNY].

i) Steps 16, 17, and 18 as before test for an empty A and print the value of the current A. SUNNY is printed.

j) A fourth copy of PRETTYLIST is called with BUTFIRST :A, [].

k) In this fourth copy the value of its input (A) is []. The test at step 23 determines that this is true and line 24 stops the fourth copy of PRETTYLIST.

l) A STOP statement terminates the current procedure (copy four of PRETTYLIST), and returns to the calling procedure (copy 3 of PRETTYLIST), at step 20 (the one after the call to the fourth copy of PRETTYLIST).

m) The next statement (step 20) is PRINT :A. The value of A in copy three is [SUNNY], so SUNNY is printed.

n) The next statement (step 21) is END. This acts like STOP and returns us to the calling procedure (copy two of PRETTYLIST). Both STOP and END have the effect of returning to the calling procedure, at the next statement to be executed in that procedure (the statement after the call). See that step 20 returns to step 13.

o) The next statement executed is step 13 (PRINT :A). The value of A in this copy of PRETTYLIST is [IS SUNNY] so this is printed.

p) The END statement in line fourteen returns us to line 6 in the original copy of PRETTYLIST. The value to be printed here is TODAY IS SUNNY.

q) The END at line 7 returns us to the command level and the process is complete.

When we use recursion we only use our original procedure. LOGO handles the creation of new copies automatically, and destroys them when they are no longer of use. You should also remember that in the new copy of a procedure all the values associated with the names are new as well. Note that in PRETTYLIST the value of A is that in the current copy of the procedure.

The original version of PRETTYLIST printed the following.

```
TODAY IS SUNNY
IS SUNNY
SUNNY
```

and then finished. This can be explained in the following way. The first three lines are printed exactly as in our previous example. When A has a value of [] (the list is empty), the STOP statement returns us to the calling procedure (copy three). The next statement in this procedure is END. This returns us to the END in copy two, which returns us to the END in copy one, and back to the command level. Nothing more is printed after the third line; the procedure simply unwinds back to the beginning.

FINDING AN ELEMENT OF A LIST

Using recursion we can write procedures that manipulate lists in different ways. For example we can write a procedure to find out whether a particular number or word is contained in a list.

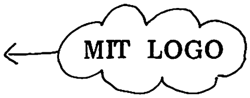
To set up the list we can use a MAKE statement.

Type

```
MAKE "OURLIST [1 7 THREE SUNNY MONDAY]
```

To write a set of procedures that will find out whether a number or word that we have read from the keyboard is an element in a list, type

```
TO FIND
PRINT [YOUR GUESS ?]
MAKE "GUESS REQUEST
TEST :GUESS =[]
IFT STOP
TEST SEARCH :GUESS :OURLIST="TRUE
IFT PRINT [IN THE LIST]
IFF PRINT [NOT IN THE LIST]
PRINT []
FIND
END
```



APPLE LOGO

```
TO FIND
PRINT [YOUR GUESS ?]
MAKE "GUESS READLIST
TEST :GUESS = []
IFT [STOP]
TEST SEARCH :GUESS :OURLIST = "TRUE
IFT [PRINT [IN THE LIST] ]
IFF [PRINT [NOT IN THE LIST] ]
PRINT[]
FIND
END
```

At this stage we haven't yet defined the procedure SEARCH. What we have done is to define our top level procedure FIND, in which we have said exactly what we want SEARCH to do. We have also indicated what inputs SEARCH will have, and what it will OUTPUT to the calling procedure FIND. This approach enables us to state what we want to do in a procedure (such as SEARCH) before it has been written.

```
TO SEARCH :A :B
TEST :B=[]
IFTRUE OUTPUT "FALSE
TEST FIRST :A=FIRST :B
IFTRUE OUTPUT "TRUE
OP SEARCH :A BF :B
END
```

```
TO SEARCH :A :B
TEST :B=[]
IFTRUE [OUTPUT "FALSE]
TEST FIRST :A=FIRST :B
IFTRUE [OUTPUT "TRUE]
OP SEARCH :A BF :B
END
```

Note the abbreviations for OUTPUT and BUTFIRST

Now type

```
FIND
```

The computer should respond

```
YOUR GUESS ?
FIVE
```

your response

The computer should now respond

```
NOT IN THE LIST

YOUR GUESS ?
```

Try this procedure with other values. We will now look at how these procedures work.

1. Firstly the computer reads in our guess. If we have pressed RETURN without entering any data, then it will read an empty list []. The first TEST statement will detect this and the procedure will stop. If we have entered data to process, the computer will use the TEST statement to determine whether the data is in the list. It tests the output from the SEARCH procedure. SEARCH will output either TRUE or FALSE, depending on whether the element is in the list or not. If the output from SEARCH is TRUE, the message IN THE LIST is printed by the FIND procedure. If the output from SEARCH is FALSE, the message NOT IN THE LIST is printed by FIND. Then the computer will call FIND again to get another guess.

2. SEARCH is called in the TEST statement with two inputs. The first is our guess and the second is the list the computer is going to search.

3. SEARCH works by testing our guess against the first element. If it does not match, the procedure goes back and does it again with everything but the first element, OUTPUT SEARCH :A BUTFIRST :B. Note that when we are using recursion and the procedure outputs a value to a calling procedure, then the recursive call itself should include OUTPUT, as in this case. This is because when the last copy finally ends it must output values all the way back to the beginning (see figure 6.1).

4. So the first thing that SEARCH does is to test for an empty list, []. If the list is empty, then we haven't found the element and SEARCH terminates and outputs FALSE.

5. If the guess is found, the search terminates and outputs TRUE.

AN ALTERNATIVE FIND PROCEDURE

In our previous FIND procedure the computer read the guess from the keyboard, and the procedure ran until it read in a null list (a line with only a RETURN in it.) We will write a new procedure (FIND1) which will be called with an input (the guess).

Type

```
TO FIND1 :GUESS
TEST :GUESS=[]
IFT STOP
TEST SEARCH :GUESS :OURLIST="TRUE
IFT (PR :GUESS [IN LIST])
IFF (PR :GUESS [NOT IN LIST])
END
```

MIT LOGO

```
TO FIND1 :GUESS
TEST :GUESS =[]
IFT [STOP]
TEST SEARCH :GUESS :OURLIST="TRUE
IFT [(PR :GUESS [IN LIST])]
IFF [(PR :GUESS [NOT IN LIST])]
END
```

APPLE LOGO

Notice the use of abbreviations, IFF for IFFALSE, IFT for IFTRUE and PR for PRINT.

Now type

```
FIND1[SUNNY]
```

The computer should respond

```
SUNNY IN LIST
```

EXERCISES

6.4 Build a new list with the name MYLIST. Run FIND and FIND1 so they use :MYLIST instead of :OURLIST.

6.5 Modify FIND and FIND1 so they will use any list that you name. Hint: pass the name of the list as an input to the procedure.

BUILDING LISTS

A way of extending lists is to use the SENTENCE statement. Type

```
PRINT :OURLIST
```

The computer should respond

```
1 7 THREE SUNNY MONDAY
```

Now type

```
MAKE "OURLIST SENTENCE :OURLIST [WET TUESDAY]

PRINT :OURLIST
```

The computer should respond

```
1 7 THREE SUNNY MONDAY WET TUESDAY
```

The SENTENCE (abbreviated SE) command can be used to combine two lists into a longer list.

Another way of building a list would be to write a procedure designed to do it. Type

```
TO BUILDLIST :NAME               TO BUILDLIST :NAME
PRINT1[NEXT ELEMENT>>]           TYPE[NEXT ELEMENT>>]
MAKE "NEXT REQUEST               MAKE "NEXT READLIST
TEST :NEXT=[]                    TEST :NEXT=[]
IFTRUE OUTPUT :NAME              IFTRUE[OUTPUT :NAME]
MAKE "NAME SE :NAME :NEXT        MAKE "NAME SE :NAME :NEXT
OUTPUT BUILDLIST :NAME           OUTPUT BUILDLIST :NAME
END                              END
```

The procedure BUILDLIST will add onto an existing list. If you wish to start with an empty (or new) list, type

```
MAKE "THISLIST []
```

prior to using BUILDLIST. This can be used with any method of building lists (such as SENTENCE). The name of the empty list we have just created is "THISLIST. Type

```
MAKE "THISLIST BUILDLIST :THISLIST
```

The computer should respond

```
NEXT ELEMENT>>
```

You now enter the elements, one at a time, and after each one press RETURN. When you are finished press RETURN without typing anything else. Then type

```
PRINT :THISLIST
```

The list that you have created will be printed. You can use any name you like for the new list.

EXERCISES

6.6 Modify BUILDLIST so that it will only add an element if it is not already present in the list. Hint: you will need a new procedure FIND2. Only if the result from FIND2 is FALSE do you add the element to the list. If the result is TRUE the element is already in the list. You will also need to use a version of FIND2 that will accept the name of the list that you are using.

EXPLORING WORDS

In an earlier chapter we met the idea of a word. For example we can type

```
PRINT "HELLO
```

The computer should respond by printing the word

```
HELLO
```

We could type

```
PRINT [HELLO]
```

The computer will still respond

```
HELLO
```

In both cases HELLO is a word. In the second example it is the only element of a list. Notice that when a word is used in a list we do not have to type the double quote character ("). A word can be made up of alphabetic or numeric characters or a mixture of both. A word cannot contain a space.

These are words:

```
"ABC
"GOODBYE
"123
```

Words that are numbers do not have to have quotes in front of them.

```
123
```

is the same as

```
"123
```

Now type

```
PRINT 10+"10
```

The computer should respond

```
20
```

Just as with lists we had a SENTENCE command, for handling words we have WORD. Type

```
PRINT WORD "NO "TIME
```

The computer should respond

```
NOTIME
```

Two separate words have been joined into a single word.

The statements FIRST, LAST, BUTFIRST and BUTLAST also work on words. Type

```
PRINT FIRST "BILL
```

The computer should respond

```
B
```

Type

```
PRINT LAST "BILL
```

The computer should respond

```
L
```

We can write procedures to manipulate words. These work in an almost identical fashion to those we used with lists. Type

```
TO EXAMINEWORD :A               TO EXAMINEWORD :A
TEST :A="                       TEST :A="
IFTRUE STOP                     IFTRUE [STOP]
PRINT FIRST :A                  PRINT FIRST :A
EXAMINEWORD BUTFIRST :A         EXAMINEWORD BUTFIRST :A
END                             END
```

We use the procedure with an input that is a word. Type

```
EXAMINEWORD "FRIDAY
```

The computer should respond

```
F
R
I
D
A
Y
```

The only difference in the form of the procedure from that of our list processing procedures is that we have tested for the empty word (") instead of the empty list ([]).

EXERCISES

6.7 Modify PRETTYLIST so that it works with words rather than lists. Call the new procedure PRETTYWORD. Test both forms of PRETTYLIST.

6.8 Modify FIND and SEARCH so that they work with words. Call these new procedures FINDW, and SEARCHW.

6.9 Modify EXAMINEWORD to use various combinations of LAST, BUTLAST, FIRST and BUTFIRST.

GETTING A NUMBER FROM A LIST

In Chapter 5 we used the statement

```
MAKE "NUMB FIRST REQUEST   MAKE "NUMB FIRST READLIST
```

When we use the REQUEST or READLIST statement, a list is read from the keyboard. If we are inputting numbers from the keyboard, it is necessary to extract the number from the list before we use it in a calculation, so the MAKE statement shown above may be used. It would also be possible to use the following procedure whenever you want to enter a number from the keyboard. Type

```
TO READNUM                   TO READNUM
OUTPUT FIRST REQUEST         OUTPUT FIRST READLIST
END                          END
```

The READNUM procedure can now be used whenever you wish to read a number from the keyboard. Type

```
PRINT 1 + READNUM
```

If you enter 2 followed by RETURN, the computer should respond

```
                  3
```

EXERCISES

6.10 Modify READNUM so that if an empty list is input (a RETURN only) it outputs zero.

6.11 Modify READNUM so that it checks whether all the characters input are numeric. If any of them are not, return to the command level after printing an error message. You return to immediate mode by the command

```
    TOPLEVEL                    THROW "TOPLEVEL
```

WRITING PROCEDURES FOR BOTH WORDS AND LISTS

It would be convenient to be able to write procedures that can be used whether the input is a word or a list. Type

```
TO EXAMINE :A                 TO EXAMINE :A
TEST EMPTYP :A="TRUE          TEST EMPTYP :A="TRUE
IFTRUE STOP                   IFTRUE [STOP]
PRINT FIRST :A                PRINT FIRST :A
EXAMINE BUTFIRST :A           EXAMINE BUTFIRST :A
END                           END
```

APPLE LOGO has a procedure called EMPTYP which enables us to achieve this. If you are using MIT LOGO type

```
TO EMPTYP :ELEMENT                      For APPLE LOGO
TEST LIST? :ELEMENT = "TRUE             don't type anything
IFTRUE OP CHECKLIST :ELEMENT            here
IFFALSE OP CHECKWORD :ELEMENT
END


TO CHECKLIST :ELEMENT
TEST :ELEMENT=[ ]
IFTRUE OUTPUT "TRUE
OUTPUT "FALSE
END

TO CHECKWORD :ELEMENT
TEST :ELEMENT="
IFTRUE OUTPUT "TRUE
OUTPUT "FALSE
END
```

Now type (everybody)

```
EXAMINE "HELLO
```

The computer should respond

```
H
E
L
L
O
```

Type

```
EXAMINE [THIS IS A TEST]
```

The computer should respond

```
THIS
IS
A
TEST
```

The LIST? command in MIT LOGO allows you to determine whether an input is a list or a word.

LOCAL AND GLOBAL VARIABLES

Type this procedure

```
TO ADD1 :B
MAKE "A :A+1
PRINT :A
MAKE "B :B+1
PRINT :B
END
```

Define the procedure and type

```
MAKE "A 1
MAKE "B 1
ADD1 :B
```

The computer should respond

```
2
2
```

which are the values of A and B printed by the procedure.

The value of B has been passed into the procedure ADD1 because it is an input to ADD1. Therefore the value 1 is passed in for B, one is added to it and the result 2 printed. Now type the following.

```
PRINT :A
```

The computer will respond

```
2
```

Type

```
PRINT :B
```

The computer will respond

```
1
```

This is rather an interesting result. A has the new value from ADD1, but B has not. No, the computer has not made a mistake and it does know how to add. The reason is that B is an input into ADD1. This means that the variable B used in the procedure is a new variable not known outside ADD1. Even though it has the same name as the variable B outside the procedure, it is not the same variable.

In other words it is called a local variable. Local variables (inputs) in a procedure have no values outside the procedure itself, so when we print out the value of B after ADD1 has finished the value has not been changed by ADD1.

On the other hand, A is called a global variable. Its value is added in ADD1 even though it is not an input, because a variable is known to all procedures that are called by the procedure it is in, if it is not used as an input. At the immediate mode level (with the question mark) a variable value is known by all procedures, providing it is not used as an input.

The advantage of using inputs is that you can use any name you like for them within a procedure; their existence is entirely local to that procedure and any procedures called by that procedure.

EXERCISES

6.12 Modify FIND and SEARCH so they work with either a word or a list.

6.13 Write a procedure that prints out the vowels it finds in a sentence (a list).

6.14 Write procedures to reverse a list (or word) that has been typed in.

FRIDAY becomes YADIRF
THIS LIST becomes LIST THIS

IDEAS INTRODUCED IN THIS CHAPTER

Lists

Manipulating lists

Simple and complex recursion

Building lists

Words

Manipulating words

Local and global variables

SUMMARY OF COMMANDS INTRODUCED IN THIS CHAPTER

MIT LOGO	APPLE LOGO	DESCRIPTION
SENTENCE,SE	SENTENCE,SE	combines input lists into single list
WORD	WORD	combines input words into single word
FIRST	FIRST	outputs first element of list or first character of word
BUTFIRST,BF	BUTFIRST,BF	outputs all but first element of list, or all but first character of word
LAST	LAST	outputs last element of list or last character of word
BUTLAST,BL	BUTLAST,BL	outputs all but last element of list, or all but last character of word

7 SECRET CODES

PLAYING WITH THE ENGLISH LANGUAGE

The list processing features of LOGO can be used to write procedures that do interesting things with the English language. As a simple example let us start by writing a procedure that recognizes whether or not a letter is a vowel. (A vowel is one of the letters A,E,I,O,U.) The procedure needs to check whether or not the letter we give it as input is one of these letters. If it is the procedure should output TRUE, if not it should output FALSE. Type

```
TO VOWELQ :LETTER                  TO VOWELQ :LETTER
IF :LETTER="A OUTPUT "TRUE         IF :LETTER="A [OUTPUT "TRUE]
IF :LETTER="E OUTPUT "TRUE         IF :LETTER="E [OUTPUT "TRUE]
IF :LETTER="I OUTPUT "TRUE         IF :LETTER="I [OUTPUT "TRUE]
IF :LETTER="O OUTPUT "TRUE         IF :LETTER="O [OUTPUT "TRUE]
IF :LETTER="U OUTPUT "TRUE         IF :LETTER="U [OUTPUT "TRUE]
OUTPUT "FALSE                      OUTPUT "FALSE
END                                END
```

(Remember that MIT LOGO is on the left hand side of the page and APPLE LOGO is on the right.)

When you have defined the procedure, use it by typing

```
PRINT VOWELQ "A
```

The computer should respond

```
TRUE
```

Test this procedure by trying various inputs to make sure that it outputs TRUE for all the vowels, and FALSE for everything else. If you type

```
PRINT VOWELQ "Z
```

the computer should respond

```
FALSE
```

Having written VOWELQ we can now write other procedures that make use of it.

PRINTING A SENTENCE WITHOUT VOWELS

Would we be able to understand a sentence if all the vowels were omitted? Let us now write a set of procedures that enable us to input one or more sentences as a list, and to print them without vowels. Type

```
TO PRINTNOVOWEL :SENT               TO PRINTNOVOWEL :SENT
IF :SENT=[ ] STOP                   IF :SENT=[ ] [STOP]
REMOVEVOWEL FIRST :SENT             REMOVEVOWEL FIRST :SENT
PRINTNOVOWEL BUTFIRST :SENT         PRINTNOVOWEL BUTFIRST :SENT
END                                 END
```

(REMOVEVOWEL hasn't been defined yet.)

This is our high level controlling procedure. We want to be able to execute it as follows (but don't type this yet).

```
PRINTNOVOWEL [THIS IS A TEST]
```

We expect the computer to print

```
THSSTST
```

How does the procedure produce this result ?

1. First it has a list [THIS IS A TEST] as input. This list will be our sentence or group of sentences.

2. It then tests to see whether the input list is empty. The first time through the procedure the list will not be empty.

3. The procedure REMOVEVOWEL is then called with an input of FIRST :SENT. In our example the word THIS is input to REMOVEVOWEL.

4. We haven't written REMOVEVOWEL yet. It will print the characters that are not vowels and skip those that are.

5. When REMOVEVOWEL is finished it will have printed THS and returned to the next statement of PRINTNOVOWEL.

6. The next statement of PRINTNOVOWEL is a call to PRINTNOVOWEL BUTFIRST :SENT. Since the computer has dealt with THIS it now calls PRINTNOVOWEL with everything but THIS, which is [IS A TEST].

7. It will keep going through this cycle until it has no more data; the list is then empty [] and the procedure will stop.

Let us now write the lower level procedure REMOVEVOWEL. Type

MIT LOGO:

```
TO REMOVEVOWEL :NEXTWORD
IF :NEXTWORD=" STOP
IF VOWELQ FIRST :NEXTWORD="FALSE PRINT1 FIRST :NEXTWORD
REMOVEVOWEL BUTFIRST :NEXTWORD
END
```

APPLE LOGO:

```
TO REMOVEVOWEL :NEXTWORD
IF :NEXTWORD=" [STOP]
IF VOWELQ FIRST :NEXTWORD="FALSE [TYPE FIRST :NEXTWORD]
REMOVEVOWEL BUTFIRST :NEXTWORD
END
```

Now try REMOVEVOWEL to see if it works. Type

```
REMOVEVOWEL "THIS
```

The computer should respond

```
THS
```

Try REMOVEVOWEL with different words. REMOVEVOWEL works on single words, so you can't use as input several words with spaces between them.

UNDERSTANDING REMOVEVOWEL

If you understand how REMOVEVOWEL works then skip this section.

REMOVEVOWEL works in a very similar way to PRINTNOVOWEL.

1. PRINTNOVOWEL sends REMOVEVOWEL a word, THIS.

2. REMOVEVOWEL checks whether its input is empty. Notice that its input is a word, not a list, so it checks for the empty word.

3. The first time through REMOVEVOWEL the word is not empty; it is THIS.

4. The procedure then checks the first character of the input to see if it is a vowel, using the previously defined procedure VOWELQ.

5. If it is not a vowel REMOVEVOWEL prints the letter with the form of the print statement that doesn't give a line feed (PRINT1 or TYPE).

6. REMOVEVOWEL is then called again with everything but the first character (BUTFIRST :NEXTWORD).

7. When the empty list is found in step 2, REMOVEVOWEL stops and returns to PRINTNOVOWEL.

In summary, PRINTNOVOWEL takes the next word on its list and sends it to REMOVEVOWEL. When REMOVEVOWEL has processed this word it returns to PRINTNOVOWEL which sends it the next word, until there are no words left.

All the component parts of PRINTNOVOWEL have been tested. Now type

```
PRINTNOVOWEL [THIS IS A TEST]
```

The computer should respond

```
THSSTST
```

Now use PRINTNOVOWEL with your own sentences. Try some very long ones.

EXERCISES

7.1 Use the procedure VOWELQ to write a set of procedures that count how many times each vowel occurs in a sentence. For example when you type

```
COUNTVOWELS [TRY THIS SENTENCE TODAY]
```

the computer should respond

```
A OCCURS 1 TIMES
E OCCURS 2 TIMES
I OCCURS 1 TIMES
O OCCURS 1 TIMES
U OCCURS 0 TIMES
```

7.2 Modify PRINTNOVOWEL so it becomes PRINTVOWEL and only prints the vowels. Can you understand the sentence after only the vowels have been printed ?

7.3 Write a new procedure that calls PRINTNOVOWEL with a list to be processed. The new procedure will ask to have the sentence entered from the keyboard.

PRINTING SENTENCES BACKWARDS

Using similar ideas to those in PRINTNOVOWEL, we can write a procedure that prints the words in a sentence in reverse order. For example:

```
REVERSELIST [THIS IS A TEST]
```

should give the following computer response

```
TEST A IS THIS
```

Now type

```
TO REVERSELIST :SENT          TO REVERSELIST :SENT
IF :SENT=[] STOP              IF :SENT=[] [STOP]
PRINT1 LAST :SENT             TYPE LAST :SENT
REVERSELIST BUTLAST :SENT     REVERSELIST BUTLAST :SENT
END                           END
```

Type

```
REVERSELIST [THIS IS A TEST]
```

The computer should respond

```
TESTAISTHIS
```

This has reversed the list but has joined the words. We will need to modify REVERSELIST, but let us first examine how it works.

1. The input to REVERSELIST is a list [THIS IS A TEST]

2. The first statement checks to see if the list is empty. If it is, then REVERSELIST is finished so the procedure stops.

3. The last word in the input is printed.

4. REVERSELIST is called again, but this time with everything but the last element of the input.

PRINTING SPACES IN A LINE

A word in LOGO cannot contain spaces, so a special sequence of characters must be input to print a space.

Let us examine the statement used in REVERSELIST to print out the result.

(Remember MIT LOGO is on the left of the page and APPLE LOGO is on the right.)

```
PRINT1 LAST :SENT            TYPE LAST :SENT
```

If this line is edited in the following way, spaces will be printed separating the words in the list.

```
(PRINT1 LAST :SENT "' ')   (TYPE LAST :SENT "\ )
```

type a double quote followed by two single quotes with a space between them

type a double quote followed by CTRL-Q and a space. This will be printed as \

The effect of these character sequences is to print a blank space. The brackets around the entire statement are necessary because two inputs are being used (:SENT and the blank space). Edit REVERSELIST with this new line.

Test REVERSELIST to see if it prints the spaces. Type

```
REVERSELIST [TODAY IS THE DAY AFTER YESTERDAY]
```

The computer should respond

```
YESTERDAY AFTER DAY THE IS TODAY
```

Now try this with some sentences of your own. Try and find some that make sense either way round, or are the same either way.

EXERCISES

7.4 Write a new procedure REVERSEALL which not only reverses the order of the words, but also prints the characters in the words in reverse order. For example:

REVERSALL [TODAY IS MONDAY]

should produce

YADNOM SI YADOT

Hint: change REVERSELIST so instead of printing out the last word of the list, it calls a new procedure WORDBACK which takes the last word of the list as input and prints it backwards.

WORDBACK should do the following:

1. Test for the empty word. If the word is empty then stop.

2. Take the last character of the input word and print it.

3. Go back and do WORDBACK recursively, but with everything but the last character of the input word.

4. When the empty word is found in step 1, print out a blank space.

Now modify REVERSELIST and write WORDBACK.

7.5 Change REVERSELIST so the words are printed in the original order, but each word is spelt backwards with WORDBACK.

CODING A MESSAGE

This project will be to write a set of procedures to code and uncode messages. We want a procedure which works as follows:

SCRAMBLE [TRY THIS AND SEE]

The computer will print out a coded message, such as:

ULPU123ODB3CC

We also want another procedure

UNSCRAMBLE[ULPU123ODB3CC]

which will print the original message

TRYTHISANDSEE

At this stage we have not bothered about the spaces between the words in the original message because it makes the code just a little bit harder to crack without the LOGO procedures.

SETTING UP THE CODES

To begin with we will construct two lists. The first is:

[A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3
4 5 6 7 8 9]

This is the list of alphabetic and numeric characters to be coded. The second list contains the codes for each of the characters in the original list.

[O A Y B C E 0 1 2 F T 6 I D J H G L 3 U 4 5 W X P R Q S Z 7 9
8 K M N V]

This second list is made by placing each alphabetic or numeric character into an unordered position in the list. The code for A is O , the code for B is A, and so on. The choice of where to place the characters in the second list is completely arbitrary. Each character in the first list appears only once in the second list. To enter these lists into the computer, type

```
MAKE "CLEARLIST [A B C  etc
```

and

```
MAKE "CODELIST [O A Y etc
```

SCRAMBLE will be a procedure very much like PRINTNOVOWEL and REVERSELIST. Type

```
TO SCRAMBLE :MESSAGE                TO SCRAMBLE :MESSAGE
IF :MESSAGE=[ ]STOP                 IF :MESSAGE=[ ][STOP]
SCRAMBLEWORD FIRST :MESSAGE         SCRAMBLEWORD FIRST :MESSAGE
SCRAMBLE BUTFIRST :MESSAGE          SCRAMBLE BUTFIRST :MESSAGE
END                                 END
```

SCRAMBLE works by taking the first word of the list and sending it to SCRAMBLEWORD. SCRAMBLEWORD carries out the coding of this word, and returns to SCRAMBLE. SCRAMBLE then calls itself with everything but the first word, and the next word is coded. When the input list is empty SCRAMBLE stops.

If you feel confident about programming with lists, try writing SCRAMBLEWORD yourself without looking at the next section.

CODING SCRAMBLEWORD

SCRAMBLEWORD will work in the following way.

1. The input to SCRAMBLEWORD is the next word in the list.

2. Test for the empty word; if it is empty return to SCRAMBLE (STOP).

3. Take the next character of the input word and see what position it has in CLEARLIST. For example A is in position 1 of CLEARLIST, B is in position 2, and so on. We will write a procedure called GETPOS to do this.

4. Then go to CODELIST and find the character in the corresponding position of that. GETCHAR will be the procedure that does this. If C is the character we wish to code, then we will find it in the third position of CLEARLIST. Now go to the third position of CODELIST and we find Y, so Y becomes the code for C.

5. Print this coded character.

6. Call SCRAMBLEWORD again, but with everything but the character just processed.

Type

MIT LOGO

```
TO SCRAMBLEWORD :NEXT
IF :NEXT=" STOP
MAKE "NUMB GETPOS FIRST :NEXT :CLEARLIST 1
PRINT1 GETCHAR :NUMB :CODELIST
SCRAMBLEWORD BUTFIRST :NEXT
END
```

APPLE LOGO

```
TO SCRAMBLEWORD :NEXT
IF :NEXT=" [STOP]
MAKE "NUMB GETPOS FIRST :NEXT :CLEARLIST 1
TYPE GETCHAR :NUMB :CODELIST
SCRAMBLEWORD BUTFIRST :NEXT
END
```

Now type GETPOS

MIT LOGO

```
TO GETPOS :CHR :LNAME :COUNT
IF :LNAME=[] OUTPUT 0
IF :CHR=FIRST :LNAME OUTPUT :COUNT
OUTPUT GETPOS :CHR BUTFIRST :LNAME :COUNT+1
END
```

APPLE LOGO

```
TO GETPOS :CHR :LNAME :COUNT
IF :LNAME=[] [OUTPUT 0]
IF :CHR=FIRST :LNAME [OUTPUT :COUNT]
OUTPUT GETPOS :CHR BUTFIRST :LNAME :COUNT+1
END
```

GETPOS works in the following way

1. It is called with the character to be found (:CHR), the name of the list to search (:LNAME), and a counter (:COUNT) that will give the position of the character. :COUNT will start off at 1.

2. If the list it is searching is found to be empty, then it has come to the end of the search without finding the character, so it outputs zero.

3. The character being searched for is tested against the first element of the list being searched. If they are the same we have come to the end of the search so we output the value of :COUNT. :COUNT contains the position number of the character.

4. If the character being searched for is not the same as the first element in the list, GETPOS is called again with everything but the first element of the list, and with :COUNT increased by 1.

Now type GETCHAR

MIT LOGO

```
TO GETCHAR :NUMB :LNAME
IF :LNAME=[] STOP
IF :NUMB=1 OUTPUT FIRST :LNAME
OUTPUT GETCHAR :NUMB-1 BUTFIRST :LNAME
END
```

APPLE LOGO

```
TO GETCHAR :NUMB :LNAME
IF :LNAME=[][STOP]
IF :NUMB=1 [OUTPUT FIRST :LNAME]
OUTPUT GETCHAR :NUMB-1 BUTFIRST :LNAME
END
```

GETCHAR is very similar to GETPOS and works in the following way.

1. The first input to GETCHAR is the position of the character in the list. This is the result from GETPOS. The second is the name of the list.

2. GETCHAR is called with the position number and the name of the list. Each time it calls itself with the next character in the list and counts down :NUMB by 1. When :NUMB is 1, then the first element of the remaining list is the one we are looking for. This character is output.

Note that GETPOS and GETCHAR output their results to the calling procedure SCRAMBLEWORD, which then uses them. Now try SCRAMBLE with different inputs. If you type

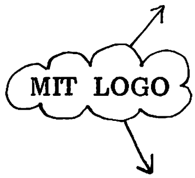
```
SCRAMBLE [TRY THIS AND SEE]
```

the computer should respond

```
ULPU123ODB3CC
```

DECODING THE MESSAGE

A procedure UNSCRAMBLE is necessary to return our coded message back to clear English. UNSCRAMBLE is the reverse of SCRAMBLE, so GETPOS and GETCHAR will still be used. Using GETPOS we will take the next character in the coded message and find its position in :CODELIST. Once we have this, we use GETCHAR to find the character at the corresponding position of :CLEARLIST. If we keep doing this until the coded message is empty then we have decoded our message. Type



```
TO UNSCRAMBLE :MESSAGE
IF :MESSAGE =[] STOP
UNSCRAMBLEWORD FIRST :MESSAGE
UNSCRAMBLE BUTFIRST :MESSAGE
END

TO UNSCRAMBLEWORD :NEXT
IF :NEXT=" STOP
MAKE "NUMB GETPOS FIRST :NEXT :CODELIST 1
PRINT1 GETCHAR :NUMB :CLEARLIST
UNSCRAMBLEWORD BUTFIRST :NEXT
END
```

```
TO UNSCRAMBLE :MESSAGE
IF :MESSAGE =[][STOP]
UNSCRAMBLEWORD FIRST :MESSAGE
UNSCRAMBLE BUTFIRST :MESSAGE
END
```

APPLE LOGO

```
TO UNSCRAMBLEWORD :NEXT
IF :NEXT="  [STOP]
MAKE "NUMB GETPOS FIRST :NEXT :CODELIST 1
TYPE GETCHAR :NUMB :CLEARLIST
UNSCRAMBLEWORD BUTFIRST :NEXT
END
```

EXERCISES

7.6 Using the same coding scheme, try double coding every letter. After you have found a code, use that code to find a new code. Modify SCRAMBLE and UNSCRAMBLE to handle this.

IDEAS INTRODUCED IN THIS CHAPTER

Projects manipulating words and sentences

* Recognizing and removing vowels

* Reversing words and sentences

* Coding and decoding messages

8 CREATING A COMPUTER POET

MAKING SENTENCES WITH LOGO

In this chapter we are going to experiment with writing procedures that enable the computer to write poetry (with our help). The method we will use is similar to that developed by Mike Sharples at the University of Edinburgh.

To begin with we will establish a dictionary of words. The dictionary will be divided into separate lists for nouns, verbs, adjectives, adverbs, articles, and so on.

Type

```
MAKE "NOUN [DOG HOUSE PLANE CHILD CAR]
MAKE "VERB [RUNS SMELLS SINGS FLIES SLEEPS]
MAKE "ADJ [TINY FRIENDLY MISERABLE BLUE QUICK]
MAKE "ADV [HAPPILY EASILY LOUDLY SLOWLY GRACEFULLY]
MAKE "ART [THE A ]
```

Don't worry if you reach the end of the screen line. Just keep typing, and only press the RETURN key when you have entered the complete LOGO command. If you make an error just retype the MAKE command from the beginning.

We can print simple sentences such as the following. Type

```
(PRINT FIRST :ADJ FIRST :NOUN FIRST :VERB)
```

The computer should respond

```
TINY DOG RUNS
```

Now type

```
(PRINT FIRST :ART FIRST BUTFIRST :NOUN LAST :VERB)
```

The computer should respond

```
THE HOUSE SLEEPS
```

This is a clumsy way of selecting words from our dictionary. What we need is a set of procedures that will randomly select words from our lists, for example (don't type this yet):

```
PRINT RAND :NOUN
PLANE

PRINT RAND :NOUN
HOUSE

PRINT RAND :VERB
FLIES
```

the computer's response

Using RAND we could build up random sentences.

(Because of the length of some of the statements, the MIT LOGO version will be presented before the APPLE LOGO version.)

Type

MIT LOGO

```
TO RAND :DICT
MAKE "NUMB 1+RANDOM 5
OUTPUT GETRANDOM :DICT :NUMB
END

TO GETRANDOM :DICT :NUMB
IF :DICT=[] OUTPUT []
IF :NUMB=1 THEN OUTPUT FIRST :DICT
OUTPUT GETRANDOM BUTFIRST :DICT :NUMB-1
END
```

APPLE LOGO

```
TO RAND :DICT
MAKE "NUMB 1+RANDOM 5
OUTPUT GETRANDOM :DICT :NUMB
END

TO GETRANDOM :DICT :NUMB
IF :DICT=[][OUTPUT []]
IF :NUMB=1 [OUTPUT FIRST :DICT]
OUTPUT GETRANDOM BUTFIRST :DICT :NUMB-1
END
```

Now type

```
(PRINT RAND :NOUN RAND :VERB RAND :ADV)
```

The computer should respond with something like

```
HOUSE SMELLS GRACEFULLY
```

RAND works by calling another procedure , GETRANDOM, which uses the procedure RANDOM to select a word from the chosen list. RAND works well but we are restricted to lists with only 5 words in them. What we need is a procedure LENGTH that can calculate how many words are in a list.

Type

MIT LOGO

```
TO LENGTH :DICT :COUNT
IF :DICT=[] OUTPUT :COUNT
OUTPUT LENGTH BUTFIRST :DICT :COUNT+1
END
```

APPLE LOGO

```
TO LENGTH :DICT :COUNT
IF :DICT=[] [OUTPUT :COUNT]
OUTPUT LENGTH BUTFIRST :DICT :COUNT+1
END
```

Now edit RAND. Delete the line

```
MAKE "NUMB 1+RANDOM 5
```

and replace it with

```
MAKE "NUMB 1+RANDOM LENGTH :DICT 0
```

We can now add new words to the dictionary. Type

```
MAKE "NOUN SENTENCE :NOUN [LIZARD SKY]
```

This can also be written using the abbreviation for SENTENCE.

```
MAKE "NOUN SE :NOUN [LIZARD SKY]
```

Using the same method, further words can be added to the other lists.

EXERCISES

8.1 Add five more words to each of NOUN, VERB, ADJ, ADV.

8.2 Print some sentences using the following statement

```
(PRINT RAND :NOUN RAND :VERB RAND :ADV)
```

8.3 Make up your own sentence-printing commands.

A MORE REFINED SENTENCE GENERATOR

The method of writing sentences is still very clumsy. We need to be able to create a template which shows the order of what we want to produce, and we want to minimize the amount of typing we have to do. A way of doing it is as follows (but don't type it yet).

```
WRITE [ART ADJ NOUN VERB]
```

which might produce

```
THE FRIENDLY DOG FLIES
```

Now type

MIT LOGO

```
TO WRITE :TEMPLATE
IF :TEMPLATE =[] PRINT [] STOP
(PRINT1 RAND THING FIRST :TEMPLATE "'   ')
WRITE BUTFIRST :TEMPLATE
END
```

Remember that "' 'allows MIT LOGO to print a space character.

APPLE LOGO

```
TO WRITE :TEMPLATE
IF :TEMPLATE =[][PRINT[] STOP]
(TYPE RAND THING FIRST :TEMPLATE "\ )
WRITE BUTFIRST :TEMPLATE
END
```

Remember that CTRL-Q allows APPLE LOGO to print a space character. When CTRL-Q is typed it appears as \.

Now use this procedure. Type

```
WRITE [ART ADJ ADJ NOUN VERB]
```

The computer might respond

```
THE MISERABLE BLUE LIZARD RUNS
```

EXERCISES

8.4 Explore different combinations of word types in templates, to see whether they make sensible sentences.

AN EVEN MORE REFINED SENTENCE GENERATOR

To write longer sentences, and to join them together in various ways, we need to be able to start new lines, and to have full stops (periods) and commas included. It would also be desirable if we could select words ourselves, as well as having the computer do it.

The following new version of WRITE will achieve these objectives.

Type

MIT LOGO

```
TO WRITE :TEMPLATE
 IF :TEMPLATE =[ ] PRINT [ ] STOP
 MAKE "WRD FIRST :TEMPLATE
 IF FIRST :WRD ="'"'( PRINT1 BUTFIRST :WRD "' ')
   ELSE ( PRINT1 RAND THING FIRST :TEMPLATE "' ')
 IF FIRST :TEMPLATE = "NL PRINT [ ]
 WRITE BUTFIRST :TEMPLATE
END
```

type as one line

APPLE LOGO

```
TO WRITE :TEMPLATE
 IF :TEMPLATE = [ ][PRINT [ ] STOP]
 MAKE "WRD FIRST :TEMPLATE
 IF FIRST :WRD = "" [(TYPE BUTFIRST :WRD " )]
   [(TYPE RAND THING FIRST :TEMPLATE " )]
 IF FIRST :TEMPLATE = "NL [PRINT[ ]]
 WRITE BUTFIRST :TEMPLATE
END
```

type as one line

Also type

```
MAKE "NL[]
MAKE "FS[.]
MAKE "CM[,]
```

Whenever NL is used a new line will be begun. FS will produce a full stop (period) and CM will cause a comma to be printed. If we type a double quote immediately in front of a word then the word itself will be printed.

```
"THE   will print  THE
"GREAT  will print  GREAT
```

Type

```
WRITE["THE ADJ NOUN VERB ADV NL "TODAY "WAS "A ADJ "DAY FS]
```

The computer might print

```
THE BIG HOUSE FLIES EASILY
TODAY WAS A FRIENDLY DAY.
```

USING THE WRITE PROCEDURE TO WRITE POETRY

A simple form of poetry was developed in Japan in ancient times. It is called Haiku. Haiku poems are free verse (they don't rhyme), and they consist of just three lines with a total of seventeen syllables. Haiku poems are written about nature and its seasons.

An example of a Haiku poem is

LATE COOL SHOWERS FALL.
TINY BLOSSOMS OPEN AND
GREET THE NEW WARM SUN.

We won't restrict ourselves to seventeen syllable poems because WRITE doesn't handle syllables. However our computer-written poems will have a similar style.

PRODUCING A TEMPLATE

To use WRITE we need first to categorize words into nouns, verbs, and so on. We may also want to print some words exactly as they appear in the poem; these we won't categorize.

```
LATE  COOL  SHOWERS FALL.
      ADJ   NOUN    VERB

TINY BLOSSOMS OPEN AND
ADJ  NOUN     VERB

GREET THE NEW WARM SUN.
VERB  ART ADJ ADJ  NOUN
```

By categorizing the words we have produced a template that we can input to WRITE to produce a similar computer poem.

We could type

```
WRITE["LATE ADJ NOUN VERB FS NL ADJ NOUN VERB "AND!
NL VERB ART ADJ ADJ NOUN FS]
```

This would work but it would mean that every time we wanted to use the template we would have to type it in again. A much more sensible approach would be to give the template a name, and use this each time.

Now type

```
    MAKE "HAIKU1 ["LATE ADJ NOUN VERB FS NL ADJ NOUN!
    VERB "AND NL VERB ART ADJ ADJ NOUN FS]
```

(Remember just keep typing when you come to the end of a screen line, and only press RETURN when you have completed the LOGO statement. The ! mark is placed at the end of the line by LOGO to tell you that you have gone to the end of the line. You can ignore the ! mark.)

Type

```
    WRITE :HAIKU1
```

The computer might respond

```
    LATE MISERABLE CHILD RUNS.
    TINY DOG SLEEPS AND
    RUNS A QUICK BLUE PLANE.
```

If this doesn't make much sense, then try again (and again and again).

Type

```
    WRITE :HAIKU1
```

You might get (we don't say you will)

```
    LATE FRIENDLY LIZARD FLIES.
    FRIENDLY HOUSE SMELLS AND
    SLEEPS THE MISERABLE BLUE SKY.
```

or

```
    LATE MISERABLE CHILD SINGS.
    QUICK DOG RUNS AND
    SMELLS THE FRIENDLY TINY CHILD.
```

Slightly better!

Another example of Haiku poetry, not produced by a computer, is

THE SUN SHINES BRIGHTLY
WITH ITS GLOWING FLAMES SHOOTING
IT GOES DOWN AT NIGHT.

Let us now set up a template from this

```
THE  SUN  SHINES  BRIGHTLY
     NOUN VERB    ADV

WITH  ITS  GLOWING  FLAMES  SHOOTING
           ADJ      NOUN

IT  GOES  DOWN  AT  NIGHT.
    VERB  ADV
```

Now type

```
MAKE "HAIKU2 ["THE NOUN VERB ADV  NL "WITH "ITS ADJ!
NOUN "SHOOTING NL "IT VERB ADV "AT "NIGHT FS]
```

Type

```
WRITE :HAIKU2
```

Whether we get uplifting poetry or mean nasty poetry will depend on the words we have in our dictionary.

EXTENDING THE DICTIONARY

Let us start our dictionary again with new words in our lists.

Now type

```
MAKE "NOUN []
MAKE "VERB []
MAKE "ADV []
MAKE "ADJ []
```

This has created a new dictionary with initially no words in the lists.

Now type

```
MAKE "NOUN [SUN TREE FLOWER BIRD CAT BUSH SNOWFLAKE!
GARDEN RIVER ROCK MOUNTAIN]

MAKE "VERB [SWAYS SHINES FALLS RISES FLIES SIGHS RUNS!
MOVES COVERS]

MAKE "ADJ [BLUE PURPLE LITTLE BIG GLOWING WHITE!
SWEET LOVELY BEAUTIFUL SOUR HUNGRY MISERABLE]
```

```
MAKE "ADV [GENTLY BRIGHTLY STEADILY SLIGHTLY!
FOREVER SELDOM]
```

Now if we type

```
WRITE :HAIKU2
```

we might get

```
THE TREE FALLS SLIGHTLY
WITH ITS PURPLE SNOWFLAKE SHOOTING
IT SWAYS GENTLY AT NIGHT.
```

or

```
THE BIRD SIGHS FOREVER
WITH ITS LITTLE FLOWER SHOOTING
IT SHINES SELDOM AT NIGHT.
```

We could use our HAIKU1 template with our new word list and we might get

```
LATE SWEET MOUNTAIN FLIES.
GLOWING ROCK COVERS AND
SHINES A BEAUTIFUL PURPLE GARDEN.
```

or

```
LATE WHITE FLOWER MOVES.
LOVELY SNOWFLAKE FLIES AND
FALLS A GLOWING SOUR SUN.
```

EXERCISES

8.5 Make a template (call it HAIKU3) for

THE TREES ABOVE ME
SWAYING ACROSS THE BLUE SKY
MAKE A LOVELY SOUND.

8.6 Add ten new words to each of the lists in the dictionary. Use them to produce new poems for the HAIKU1, HAIKU2 and HAIKU3 templates.

8.7 Extend the dictionary, with lists of pronouns, prepositions and so on. Use these word types to make more complicated templates.

8.9 Think about ways to introduce rhyming into your computer poetry. Hint: you would need lists of words that rhyme with each other.

IDEAS INTRODUCED IN THIS CHAPTER

Building dictionaries

Sentence generating procedures

Writing poetry using templates

APPENDIXES

APPENDIX A

SUMMARY OF COMMANDS

This Appendix lists the command sets of both MIT LOGO and APPLE LOGO in summary form. Where possible, commands are grouped according to function. Similar MIT and APPLE commands are shown on the same line.

A full description of LOGO commands can be found in the documentation supplied with the language.

ACTION	MIT LOGO	APPLE LOGO
TURTLE GRAPHICS		
Move turtle back BACK 50	BACK, BK	BACK, BK
Set background color (a number from 0-5, Black 0, White 1, Green 2, Violet 3, Orange 4, Blue 5)	BACKGROUND, BG	SETBG
Output background color number	----	BACKGROUND, BG

ACTION	MIT LOGO	APPLE LOGO
Clear the screen, (does not change turtle state)	CLEARSCREEN, CS	CLEAN
Clear the screen, place turtle home	DRAW	CLEARSCREEN, CS
Move the turtle forward FORWARD 50	FORWARD, FD	FORWARD, FD
Full graphic screen	FULLSCREEN	FULLSCREEN
Output turtle's heading	HEADING	HEADING
Make the turtle invisible	HIDETURTLE, HT	HIDETURTLE, HT
Move the turtle to centre of screen pointing up	HOME	HOME
Rotate the turtle left LEFT 50	LEFT, LT	LEFT, LT
Enter text mode with clear screen	NODRAW,ND	see CLEARTEXT
Prevent drawings from wrapping around screen	NOWRAP	FENCE
Make turtle drawings wrap around screen	WRAP	WRAP
Turtle continues to move unseen off screen	----	WINDOW
Set pencolor (See BACKGROUND for color numbers)	PENCOLOR, PC	SETPC
Output pen color	----	PENCOLOR, PC

ACTION	MIT LOGO	APPLE LOGO
Turtle leaves trail	PENDOWN, PD	PENDOWN, PD
Turtle ceases to leave trail	PENUP, PU	PENUP, PU
Turn turtle right RIGHT 90	RIGHT, RT	RIGHT, RT
Set turtle heading SETH 180	SETHEADING, SETH	SETHEADING, SETH
Move the turtle to the specified x coordinate SETX 50	SETX	SETX
Move the turtle to specified x,y coordinates MIT SETXY 50 50 APPLE SETPOS [50 50]	SETXY	SETPOS
Move the turtle to specified Y coordinate SETY 50	SETY	SETY
Show turtle shape (see HIDETURTLE)	SHOWTURTLE, ST	SHOWTURTLE, ST
Output TRUE if turtle is shown	----	SHOWNP
Mixed graphics and text screen	SPLITSCREEN	SPLITSCREEN
Output heading turtle needs to face an x,y coordinate MIT TOWARDS 0 0 APPLE TOWARDS [0 0]	TOWARDS	TOWARDS
Output a list of turtle pen and color state	TURTLESTATE,TS	PEN

ACTION	MIT LOGO	APPLE LOGO
Output current x coordinate of turtle	XCOR	XCOR
Output current y coordinate of turtle	YCOR	YCOR
Place a dot at the specified x y position DOT [50 10]	----	DOT
Output current x,y coordinates of turtle	see XCOR YCOR	POS
Set vertical screen scale factor, default is 0.8	.ASPECT	SETSCRUNCH
Output current vertical screen scale factor	----	SCRUNCH
Turtle erases lines it passes over	----	PENERASE, PE
Screen devoted to text only	----	TEXTSCREEN
ARITHMETIC OPERATIONS		
a plus b	a + b	a + b
difference of a and b	a - b	a - b
product of a and b	a * b	a * b
a divided by b	a / b	a / b
a smaller than b (outputs TRUE or FALSE)	a < b	a < b
a equal to b (outputs TRUE or FALSE)	a = b	a = b
a greater than b (outputs TRUE or FALSE)	a > b	a > b

ACTION	MIT LOGO	APPLE LOGO
Output the arctangent of sides a and b MIT ATAN a b APPLE ARCTAN a/b	ATAN	ARCTAN
Output cosine of angle expressed in degrees	COS	COS
Output integer part of input number	INTEGER	INT
Take two inputs divide first by the second and output the integer result. MIT LOGO first rounds the inputs	QUOTIENT	QUOTIENT
Take one input (N) and output random number between 0 and N-1 MIT random numbers are repeatable APPLE random numbers are not repeatable	RANDOM	RANDOM
Make MIT LOGO random numbers unrepeatable	RANDOMIZE	----
Make APPLE LOGO random numbers repeatable	----	RERANDOM
Output the integer remainder, first input modulo the second; MIT LOGO rounds the numbers first if they are not integers	REMAINDER	REMAINDER
Output the integer nearest the input	ROUND	ROUND

ACTION	MIT LOGO	APPLE LOGO
Output Sine of angle expressed in degrees	SIN	SIN
Output square root	SQRT	SQRT
Output sum of inputs	----	SUM
Output product of inputs	----	PRODUCT

LIST AND WORD OPERATIONS

Output the ASCII code of a character	ASCII	ASCII
Output all but the first element of a word or list	BUTFIRST, BF	BUTFIRST, BF
Output all but the last element of a word or list	BUTLAST, BL	BUTLAST, BL
Output the number of elements in a list	----	COUNT
Take an ASCII code as input and output the corresponding character	CHAR	CHAR
Output TRUE if input is empty word or list	----	EMPTYP
Output TRUE if inputs are equal	----	EQUALP
Output first element of a word or list	FIRST	FIRST
Output a list of first input followed by the second	FPUT	FPUT

ACTION	MIT LOGO	APPLE LOGO
Output the Nth element of the list or word	----	ITEM
Output the last element of a word or list	LAST	LAST
Take two or more lists and output a list of them (see SENTENCE)	LIST	LIST
Output TRUE if input is a list	LIST?	LISTP
Output a list of the second input followed by the first input (the second input must be a list)	LPUT	LPUT
Output TRUE if first input is an element of its second input (a list)	----	MEMBERP
Output TRUE if the input is a number	NUMBER?	NUMBERP
Take two or more inputs and output them in a single list	SENTENCE, SE	SENTENCE, SE
Output TRUE if the input has a value associated with it	THING?	NAMEP
Take two or more inputs and output them in a single word	WORD	WORD
Output TRUE if the input is a word	WORD?	WORDP

ACTION	MIT LOGO	APPLE LOGO
DEFINING AND EDITING PROCEDURES		
Output TRUE if the input is the name of a procedure	----	DEFINEDP
Enter the editor with the procedure title MIT EDIT CIRCLE APPLE EDIT "CIRCLE	EDIT, ED	EDIT, ED
Terminate a procedure definition	END	END
Output TRUE if the input is a LOGO primitive	----	PRIMITIVEP
Define a procedure MIT enters screen editor APPLE enters line editor	TO	TO
VARIABLES		
Make input name local to a procedure	----	LOCAL
Assign the second input to be the value of the first, which must be a word MAKE "A 45 PRINT :A 45	MAKE	MAKE
Output the value associated with a name PRINT THING "A is the same as PRINT :A	THING	THING

ACTION	MIT LOGO	APPLE LOGO
CONTROLLING PROCEDURE EXECUTION		
Resume execution after pause	CONTINUE, CO	CO
Unconditional transfer of control (see LABEL)	GO	GO
Label line for use by GO MIT AGAIN: APPLE LABEL "AGAIN	name	LABEL
Turn off trace	NOTRACE	----
Stop the current procedure and output the result to the calling procedure	OUTPUT, OP	OUTPUT, OP
Pause execution of procedure	PAUSE	PAUSE
Inputs a number and a list; the list is run the designated number of times.	REPEAT	REPEAT
Stop the current procedure and return to the calling procedure (see OUTPUT)	STOP	STOP
Return to immediate level	TOPLEVEL	THROW "TOPLEVEL
Execute one line at a time (see NOTRACE)	TRACE	----
CONDITIONALS AND FLOW OF CONTROL		
Output TRUE if all inputs are TRUE	ALLOF	AND

ACTION	MIT LOGO	APPLE LOGO
Output TRUE if any inputs are TRUE	ANYOF	OR
Output TRUE if input FALSE	NOT	NOT

Conditional transfer of control
MIT
IF condition THEN action
IF :A = "TODAY THEN STOP
IF :B = "MONDAY THEN STOP ELSE SOMETHING
APPLE
IF condition [list 1][list 2]
IF :A = "TODAY [STOP]
IF :B = "MONDAY [STOP][SOMETHING]
	IF	IF

Test condition to be used with IFTRUE or IFFALSE TEST :A=:B	TEST	TEST
Execute line only if test is TRUE MIT IFTRUE STOP APPLE IFTRUE [STOP]	IFTRUE, IFT	IFTRUE, IFT
Execute line only if test is FALSE	IFFALSE, IFF	IFFALSE, IFF
INPUT AND OUTPUT		
Clear text screen	CLEARTEXT	CLEARTEXT
Take two inputs, a column and line number, and position the cursor there	CURSOR	SETCURSOR
Output the current cursor position	----	CURSOR

ACTION	MIT LOGO	APPLE LOGO
Take two inputs; the first is an address, the second is contents to be written to that address	.DEPOSIT	.DEPOSIT
Take one input, a slot number, and direct screen output there	OUTDEV	.PRINTER
Output a number between 0-255 representing the setting of the games dial	PADDLE	PADDLE
Output TRUE if the input paddle button number that is its input is pressed	PADDLEBUTTON	BUTTONP
Print input on the screen	PRINT, PR	PRINT, PR
As in PRINT but the cursor remains on the same line	PRINT1	TYPE
Output TRUE if a character has been typed but not yet read	RC?	KEYP
Output the first character typed	READCHARACTER, RC	READCHAR, RC
Output a list that has been typed (terminated with RETURN key)	REQUEST, RQ	READLIST, RL
LOGO will wait for N 60ths of a second WAIT 40	----	WAIT

ACTION	MIT LOGO	APPLE LOGO
FILING AND MANAGING WORKSPACE		
Print names of diskette files	CATALOG	CATALOG
Erase the named titles from workspace	ERASE, ER	ERASE, ER
Erase all procedures from workspace	ERASE ALL	ERALL
Erase a file from diskette	ERASEFILE	ERASEFILE
Erase a picture from diskette	ERASEPICT	-----
Erase named variables	ERNAME	ERN
Restart LOGO	GOODBYE	----
Print out procedures in workspace	PRINTOUT, PO	PO
Print out all procedures in workspace	PRINTOUT ALL	POALL
Print out titles in workspace	POTS	POTS
Read contents of diskette file into workspace	READ	LOAD
Read a picture file from diskette	READPICT	----
Save workspace on file on diskette	SAVE	SAVE
Save picture on diskette	SAVEPICT	----

ACTION	MIT LOGO	APPLE LOGO
EDITING COMMANDS		
Rub out character to immediate left of cursor	ESC	<——(or CTRL-H)
Move cursor one character to right	——>	CTRL-F(or ——>)
Move cursor one character to left	<——	CTRL-B
Repeat character	REPT	REPT
Move cursor to beginning of line	CTRL-A	CTRL-A
Move cursor back one screenful	CTRL-B	ESC V
Leave editor and define procedure	CTRL-C	CTRL-C
Delete character at current cursor position	CTRL-D	CTRL-D
Move cursor to end of current line	CTRL-E	CTRL-E
Move cursor one screenful forward	CTRL-F	CTRL-V
Stop editor without defining procedure	CTRL-G	CTRL-G
Delete all characters to right of cursor	CTRL-K	CTRL-K
Move cursor to centre of screen	CTRL-L	CTRL-L
Move cursor down one line	CTRL-N	CTRL-N

ACTION	MIT LOGO	APPLE LOGO
Open line for data entry	CTRL-O	CTRL-O
Move cursor up one line	CTRL-P	CTRL-P
Quote next character	'	CTRL-Q

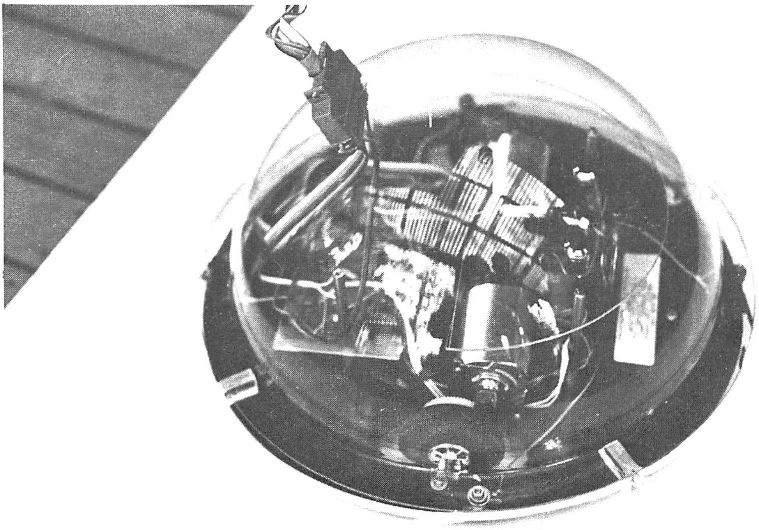
OUTSIDE THE EDIT MODE

Full screen graphics mode	CTRL-F	CTRL-L
Stop execution	CTRL-G	CTRL-G
Mixed graphics/text screen mode	CTRL-S	CTRL-S
Text screen mode	CTRL-T	CTRL-T
Stop program execution, any character will resume processing	CTRL-W	CTRL-W
Pause	CTRL-Z	CTRL-Z
Left square bracket [	SHIFT N	SHIFT N
Right square bracket]	SHIFT M	SHIFT M
Retrieve previous line typed	CTRL-P	CTRL-Y

APPENDIX B

ROBOT TURTLES

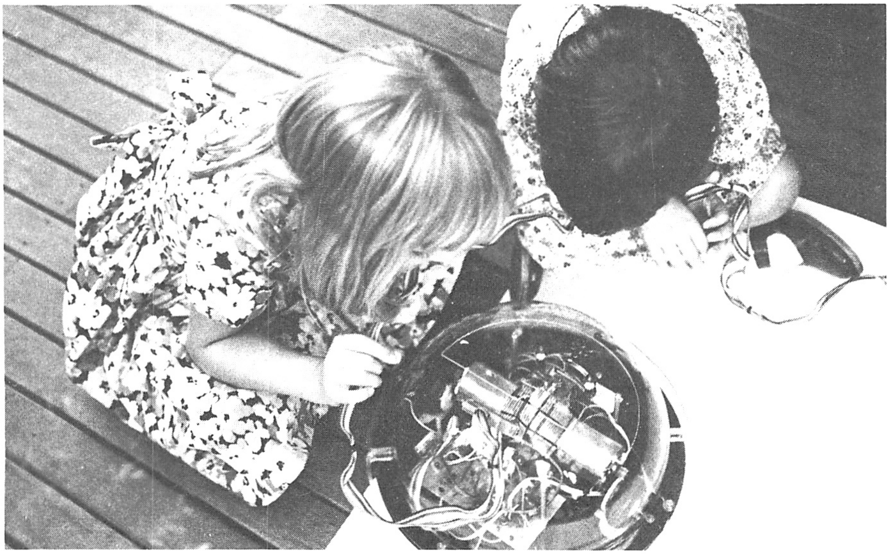
A robot turtle is a wheeled robot with a felt-tipped pen underneath. Programmed by LOGO commands, it moves about on a large piece of paper on the floor. It can draw its path as it goes. It usually has a pair of little lamps (near the front, positioned like eyes), a small speaker which can produce a tooting sound, and touch sensors around its rim.



A robot turtle is obviously bigger and slower than a screen turtle. Particularly for younger children, its movements can provide a very concrete outcome for their programming projects.

The robot turtle moves in a horizontal plane, so that FORWARD is just that; it does not become "upwards" as it does for the screen turtle in its vertical screen world. A robot turtle's LEFT and RIGHT can be checked by a person standing behind it - a feat not possible with the screen turtle.

Robot turtles and the pictures they draw are big enough for groups of students to see, so they are very suitable for classroom use. Groups of children working with turtles can share co-operative planning and problem solving tasks.



At the time of writing, robot turtles which can be interfaced to the Apple Computer for use with LOGO are available from the following two addresses:

Flexible Systems Pty. Ltd.
219 Liverpool St
Hobart
Tasmania 7000
Australia

Terrapin Inc.
380 Green St.
Cambridge
Massachusetts 02139
U.S.A.

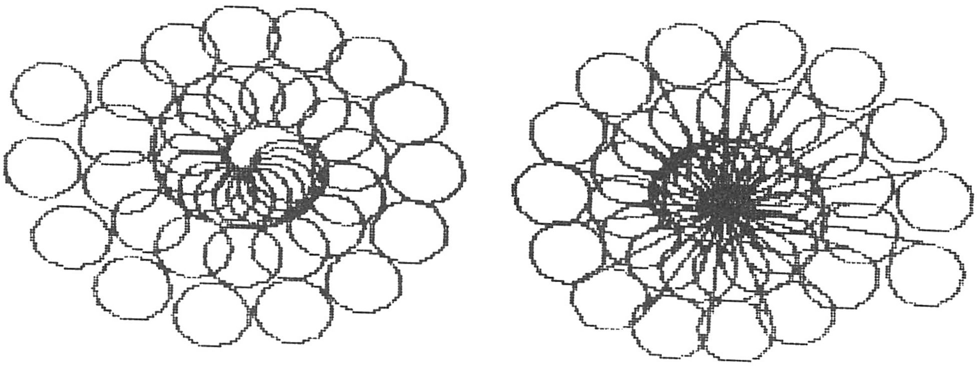
Detailed documentation about the robots is provided from each of the sources above. Further information can be obtained by contacting these sources.

APPENDIX C

MORE TURTLE GEOMETRY PROJECTS

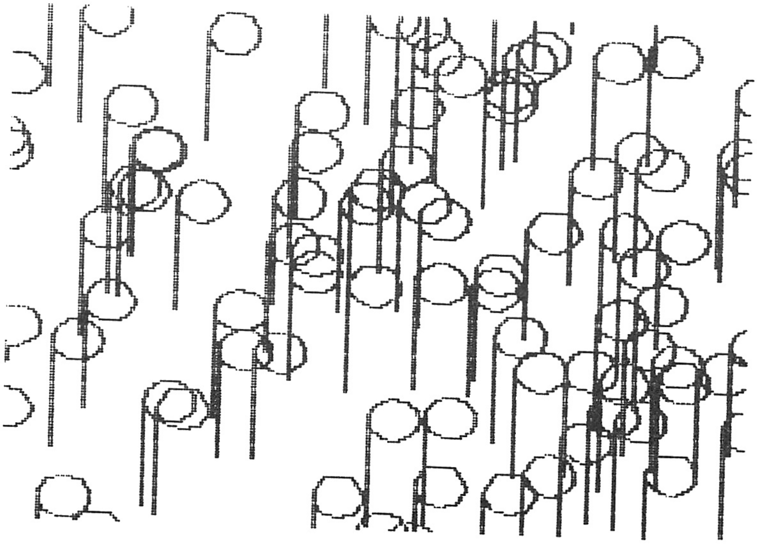
Some of the drawings in this Appendix are developments of ideas in earlier chapters, whilst others are completely new. Sometimes the procedures are given, sometimes some hints, and sometimes only the drawings themselves. The main purpose is to provide a source of ideas for further LOGO projects. Other sources of LOGO projects mentioned in the bibliography are Harold Abelson's book, APPLE LOGO, and its MIT LOGO companion; these books also have an introduction to the use of the dynaturtle, which will be of value to those interested in physics. MINDSTORMS, the beautiful exposition of the LOGO philosophy by Seymour Papert, has a number of project ideas that haven't been shown elsewhere, and is a rich source of inspiration in other ways. For those with mathematical skills, TURTLE GEOMETRY, by Abelson and DiSessa, has some exciting possibilities based on more sophisticated mathematical relationships. Many of the papers in the bibliography will provide ideas. Ultimately LOGO is about exploration, so your own imagination should prove the richest source of all.

SPIRALLING BALLS



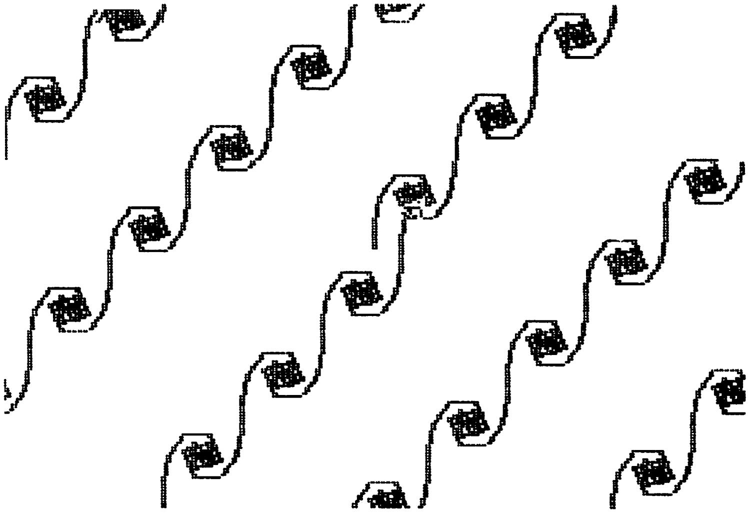
These figures are a modification of the WHEEL project in Chapter 4.

BALLOONS



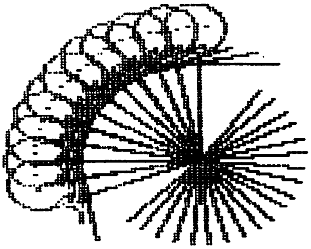
This is based on BUBBLES from Chapter 5, but incorporates some ideas from WHEEL.

A MYSTERY DRAWING



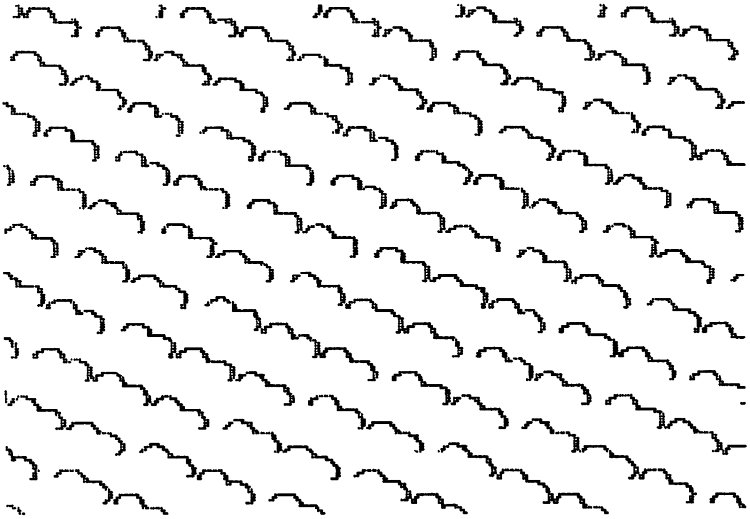
All the authors know about this one is that it happened when they were playing with the SIN procedure. It bears some resemblence to the INSPI procedures in Harold Abelson's books, but how we don't know.

MORE SPINNING STICK FIGURES



Here MAN (Chapter 3) is spun at 10 degree intervals, but then stopped before completion with CTRL-G.

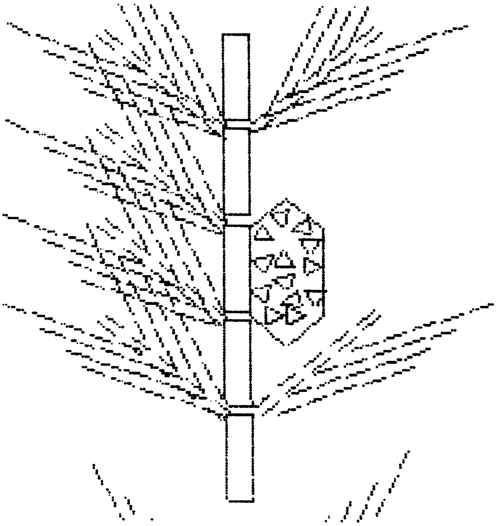
FLOCKS OF BIRDS

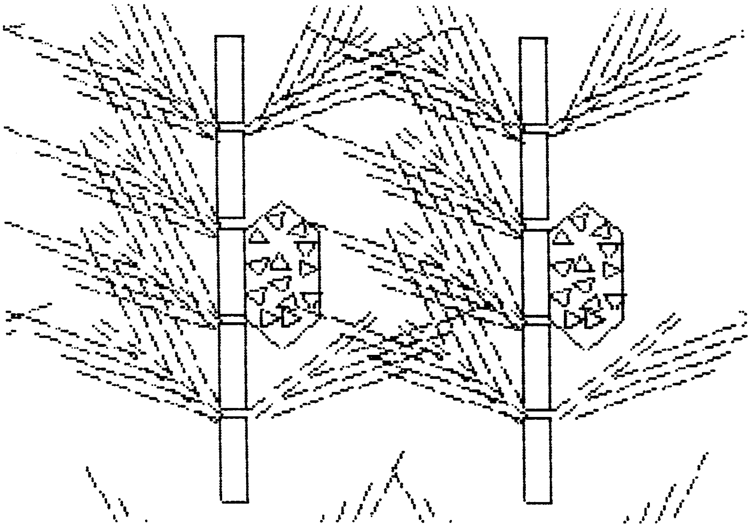


This was a 12 year old's inspiration from a MINDSTORMS project.

THE CASUARINA TREE AND FOREST

This tree was an all day project for a 12 year old and his mother.





```
TO CASUARINA
 STALK
 PU
 BK 184
 PD
 REPEAT 4 [LT 45 BRANCH PU BK 70 RT 45 FD 44 PD]
 PU
 RT 90
 FD 10
 LT 90
 BK 170
 PD
 RT 45
 BRANCH
 PU
 BK 70
 LT 45
 FD 44
 PD
 FRUITA
 PU
 SETH 0
 LT 90
 FD 16
 RT 90
 FD 48
 RT 45
 PD
 BRANCH
 PU
 BK 60
 LEAVES1
 RT 90
 FD 4
 LT 90
 LEAVES2
 LT 45
 BK 136
 RT 45
 FD 15
 PD
 LEAVES2
 LT 135
 FD 10
 RT 45
 FD 10
 LEAVES2
 LT 90
 FD 6
 RT 90
 LEAVES1
 RT 45
 FD 50
 LT 45
 FD 10
 LEAVES2
 LT 90
 FD 6
 RT 90
 LEAVES1
 RT 45
 FD 52
 LT 45
 FD 10
 LEAVES2
 LT 90
 FD 6
 RT 90
 LEAVES1
 RT 45
 FD 54
 LT 45
 FD 10
 LEAVES2
 LT 90
 FD 6
 RT 90
 LEAVES1
 HT
END
```

```
TO FRUIT
 FD 40
 RT 45
 FD 20
 RT 90
 FD 20
 RT 45
 FD 40
 RT 45
 FD 20
 RT 90
 FD 20
 BK 20
 SETHEADING 0
 PU
 FD 10
 PD
 TRI
 PU
 LT 90
 FD 8
 RT 90
 PD
 TRI
 PU
 FD 10
 LT 45
 PD
 TRI
 PU
 RT 135
 FD 6
 LT 45
 PD
 TRI
 PU
 LT 45
 FD 8
 LT 45
 FD 10
 PD
 TRI
 PU
 LT 180
 FD 10
 RT 90
 PD
 TRI
 PU
 LT 225
 FD 14
 PD
 TRI
 PU
 FD 10
 PD
 TRI
 PU
 RT 90
 FD 10
 PD
 TRI
 PU
 LT 65
 FD 10
 PD
 TRI
 PU
 LT 110
 FD 6
 PD
 TRI
END
```

```
TO FRUITA
 FD 40
 RT 45
 FD 20
 RT 90
 FD 20
 RT 45
 FD 40
 RT 45
 FD 20
 RT 90
 FD 20
 BK 20
 SETHEADING 0
 PU
 FD 10
 PD
 TRI
 PU
 LT 90
 FD 8
 RT 90
 PD
 TRI
 PU
 FD 10
 LT 45
 PD
 TRI
 PU
 RT 135
 FD 6
 LT 45
 PD
 TRI
 PU
 LT 45
 FD 8
 LT 45
 FD 8
 PD
 TRI
 PU
 BK 12
 PD
 TRI
 PU
 SETHEADING 0
 RT 90
 FD 6
 PD
 TRI
 SETHEADING 0
 PU
 FD 8
 PD
 TRI
 PU
 SETH 0
 FD 4
 LT 90
 FD 4
 PD
 TRI
 PU
 SETH 0
 FD 12
 LT 90
 FD 8
 PD
 TRI
 PU
 SETH 0
 RT 90
 FD 10
 PD
 TRI
 PU
 LT 135
 FD 8
 PD
 TRI
 PU
 SETH 0
 BK 4
 RT 45
 FD 4
 PD
 TRI
 PU
END
```

```
TO BRANCH
 REPEAT 5 [FD 10 RT 90 PU FD 4 RT 90
 PD FD 10 RT 90 PU FD 4 RT 90 FD 14 PD]
END
```

do not press RETURN; type straight on

```
TO STALK
 PU
 BK 100
 PD
 REPEAT 5 [FD 40 RT 90 FD 10 RT 90
 FD 40 RT 90 FD 10 RT 90 PU FD 44 PD]
END
```

```
TO LEAVES1
 LT 22.5
 PD
 LEAF
 PU
 BK 65
 RT 22.5
 FD 14
 LT 22.5
 PD
 LEAF
 PU
 BK 60
 RT 22.5
 FD 14
 LT 22.5
 PD
 LEAF
 PU
 BK 55
 RT 22.5
 FD 14
 LT 22.5
 PD
 LEAF
 PU
 BK 55
 RT 22.5
 BK 60
END
```

```
TO LEAF
 REPEAT 5 [FD 10 PU
 FD 2 PD]
END
```

do not press RETURN

```
TO LEAVES2
 RT 22.5
 PD
 LEAF
 PU
 BK 65
 LT 22.5
 FD 14
 RT 22.5
 PD
 LEAF
 PU
 BK 60
 LT 22.5
 FD 14
 RT 22.5
 PD
 LEAF
 PU
 BK 55
 LT 22.5
 FD 14
 RT 22.5
 PD
 LEAF
 PU
 BK 55
 LT 22.5
 BK 50
END
```

```
TO FRUIT2
 LT 180
 FD 10
 RT 90
 PD
 TRI
 PU
 LT 225
 FD 14
 PD
 TRI
 PU
 FD 10
 PD
 TRI
 PU
 RT 90
 FD 10
 PD
 TRI
 PU
 LT 65
 FD 10
 PD
 TRI
 PU
 LT 110
 FD 6
 PD
 TRI
END
```

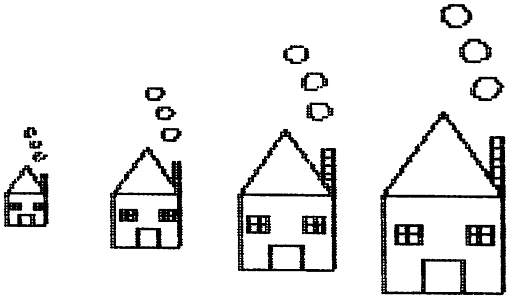
```
TO FRUIT1
 FD 40
 RT 45
 FD 20
 RT 90
 FD 20
 RT 45
 FD 40
 RT 45
 FD 20
 RT 90
 FD 20
 BK 20
 SETHEADING 0
 PU
 FD 10
 PD
 TRI
 PU
 LT 90
 FD 8
 RT 90
 PD
 TRI
 PU
 FD 10
 LT 45
 PD
 TRI
 PU
 RT 135
 FD 6
 LT 45
 PD
 TRI
 PU
 RT 45
 FD 8
 LT 45
 FD 16
 PD
 TRI
 PU
END
```

```
TO TRI
 FD 8
 RT 120
 FD 8
 RT 120
 FD 8
 RT 120
END
```

```
TO FRUITB
 LT 180
 FD 10
 RT 90
 PD
 TRI
 PU
 LT 225
 FD 14
 PD
 TRI
 PU
 FD 10
 PD
 TRI
 PU
 RT 90
 FD 10
 PD
 TRI
 PU
 LT 65
 FD 10
 PD
 TRI
 PU
 LT 110
 FD 6
 PD
 TRI
END
```

MORE ELABORATE HOUSES

Houses have formed a major part of this book. This street of houses explores LOGO inputs and proportions to make houses of varying sizes.



```
TO STREET1 :SIDE

 PU
 SETX ( - 100 )
 PD
 REPEAT 4 [HOUSE8 :SIDE SETH 180 SETY 0 LT 90 FD 30 LT 90 PD
  MAKE "SIDE :SIDE+10]
END
```

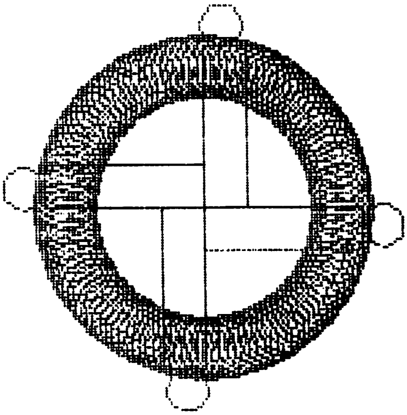
do not press RETURN

```
TO HOUSE8 :SIDE
 HOUSE6
 FD :SIDE
 LT 90
 FD :SIDE / 3
 DOOR1
 LT 90
 FD :SIDE / 2
 LT 90
 PU
 FD :SIDE * 9 / 10
 RT 90
 PD
 WINDOW1
 PU
 RT 90
 FD :SIDE * 4 / 10
 LT 90
 PD
 WINDOW1
 PU
 RT 90
 FD :SIDE / 10
 LT 90
 FD :SIDE / 2
 PD
 FD :SIDE / 10
 LT 90
 FD :SIDE / 10
 RT 90
 CHIMNEY1
 SMOKE1
 HT
END

TO PUFF1
 REPEAT 18 [LT 20 FD :SIDE*1/25]
END
```

```
TO SMOKE1
 PU
 REPEAT 3 [FD :SIDE*2/5 PD RT 80 PUFF1 LT 90 PU]
END

TO CHIMNEY1
 REPEAT 5 [PANE FD :SIDE/10]
END

TO WINDOW1
 REPEAT 2 [PANE FD :SIDE/10 PANE BK :SIDE/10 RT 90 FD :SIDE/10 LT 90]
END

TO PANE
 REPEAT 4 [FD :SIDE/10 RT 90]
END

TO DOOR1
 LT 90
 REPEAT 2 [FD :SIDE/3 RT 90]
 FD :SIDE / 3
 LT 90
 FD :SIDE / 3
END

TO SQUARE2
 REPEAT 4 [FD :SIDE LT 90]
END

TO TRIANGLE
 REPEAT 3 [FD :SIDE LT 120]
END

TO HOUSE6
 RT 90
 TRIANGLE
 RT 90
 SQUARE2
END
```

A SHIP'S WHEEL

This includes a rectangle spun through 90 degrees, with a small circle on top of it (RCIRCLE from Chapter 4) and WHEELS again.

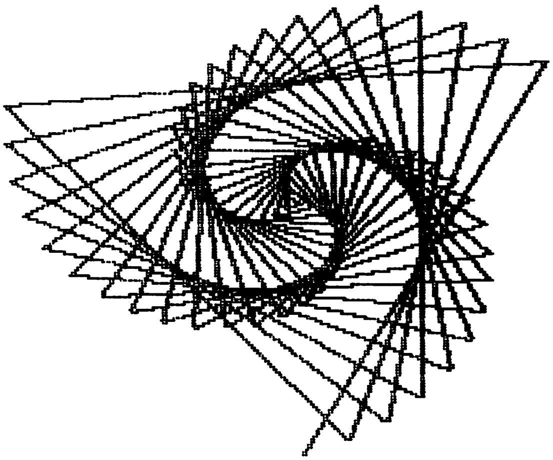


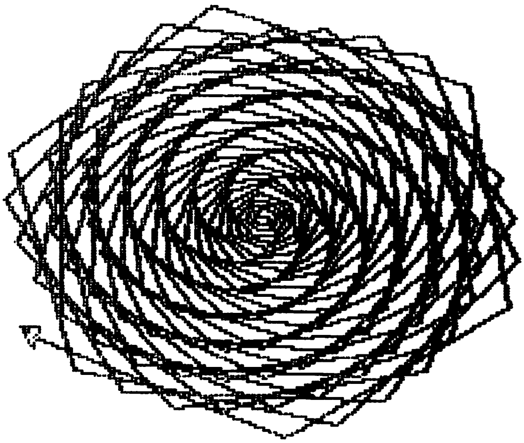
SWIMMING

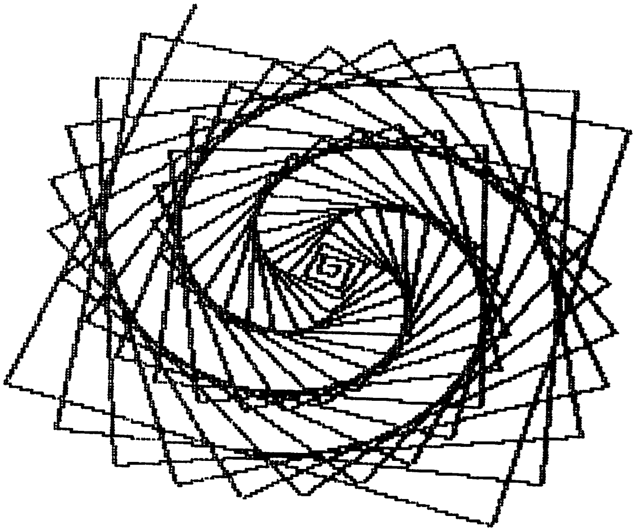
Playing with color commands (Appendix D) produces weird effects on a black and white screen.

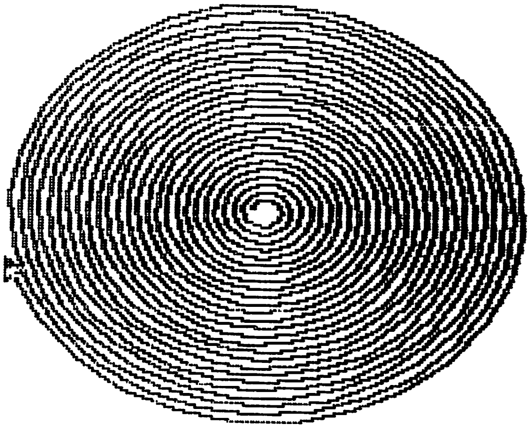
SPIRAL GALAXIES

These figures have been drawn with the spiralling procedure of Chapter 4, except that the angles used are 2 or 3 degrees more than you would use for a triangle, square, etc.









APPENDIX D

USING COLOR WITH LOGO

If a color video monitor is connected to your Apple you will be able to draw pictures with LOGO in color. Color is used in two ways. Firstly the background may take on a color such as orange or green. This has an effect similar to that of using a sheet of colored paper on which to do a drawing. As well as a background color LOGO allows the turtle to leave a trail in different colors, as if you were drawing on a colored sheet of paper with different colored crayons.

Colors are numbered 0 through 5:

black	0
white	1
green	2
violet	3
orange	4
blue	5

Background colors may be set by the following command.

```
BACKGROUND 2 (or BG 2)    SETBG 2
```

(Remember MIT LOGO is on the left and APPLE LOGO is on the right.)

The above example sets the background color to green.

The pen color may be set by the following command.

```
    PENCOLOR 1 (or PC 1)      SETPC 1
```

The pen color will be set to white. If we now move the turtle in immediate mode or run a turtle procedure, the computer will draw a white figure on a green background; try it and see.

Because of the way the Apple handles color, not all color combinations are possible. Some of the color combinations that do produce good results are:

BACKGROUND	PEN COLOR
green	white
violet	white, green
orange	white, blue
blue	white, orange
black	white, green, violet, blue orange

If no colors are specified the turtle will draw with a white trail on a black screen. The color codes can be used with a black and white monitor to provide unusual background and trail effects.

FILLING IN FIGURES

An interesting use of color is to fill in figures such as squares, houses, circles, etc. By starting with a rectangle 50 steps long and 1 step wide, and by increasing the width each time by one step, then after 50 changes in the width, a solid colored square will be produced. This example is a good way of testing different combinations of colors. Type

```
TO FILLINSQ  :GR :PN :COUNT
BG :GR PC :PN
IF :COUNT=50 STOP
REPEAT 2[FD 50 RT 90 FD :COUNT+1 RT 90]
FILLINSQ :GR :PN :COUNT+1
END
```

MIT LOGO

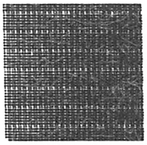
```
TO FILLINSQ :GR :PN :COUNT
SETBG :GR SETPC :PN
IF :COUNT=50 [STOP]
REPEAT 2[FD 50 RT 90 FD :COUNT+1 RT 90]
FILLINSQ :GR :PN :COUNT+1
END
```

This procedure can be run with different background and pen colors.

Type

```
FILLINSQ 4 3 0
```

The computer should respond in color (if you have it)



The first input is the background color, the second is the pen color and the third a counter to cause the procedure to stop when the square is filled in.

This principle can be applied to other shapes as well, although you may want to think about a more efficient approach where the turtle doesn't go over previously drawn lines.

APPENDIX E

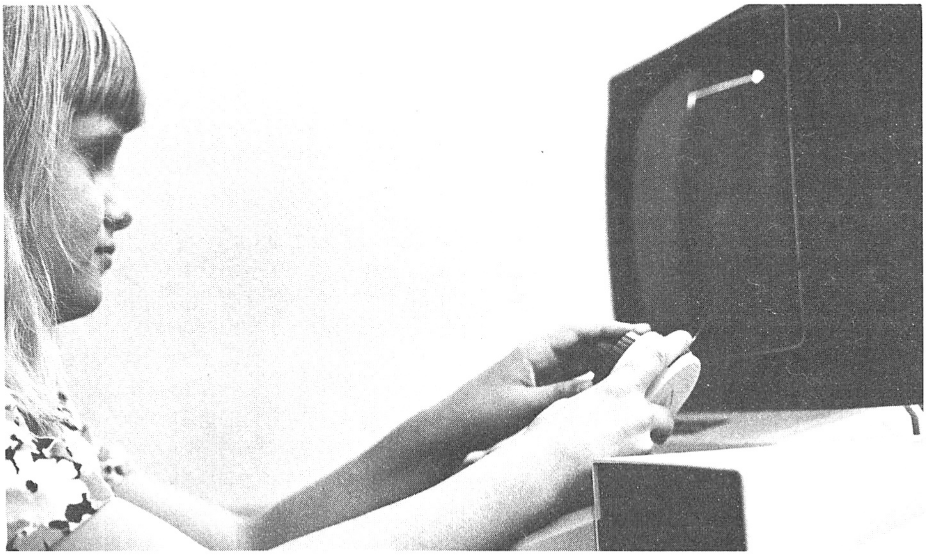
TODDLER: A LOGO MICROWORLD FOR YOUNG CHILDREN

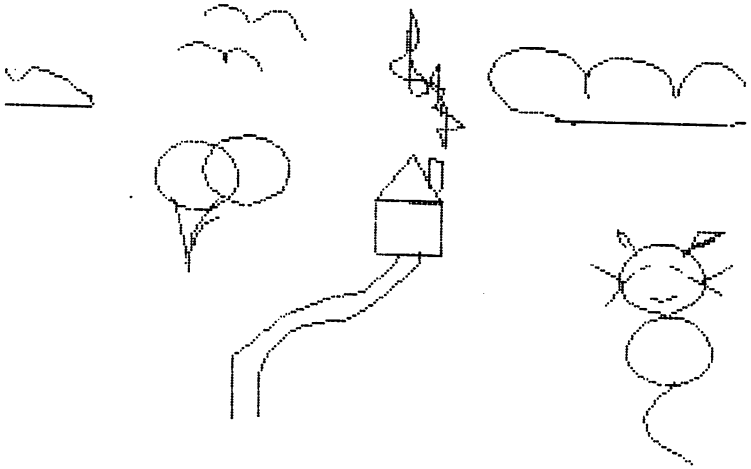
The toddler microworld is a simple set of single letter commands that can be used from the keyboard to move the turtle. Included are commands for moving the turtle forward, back, left and right, for starting and stopping the turtle's trail and for clearing the screen. In addition single character commands can be used to draw squares, triangles, and circles.

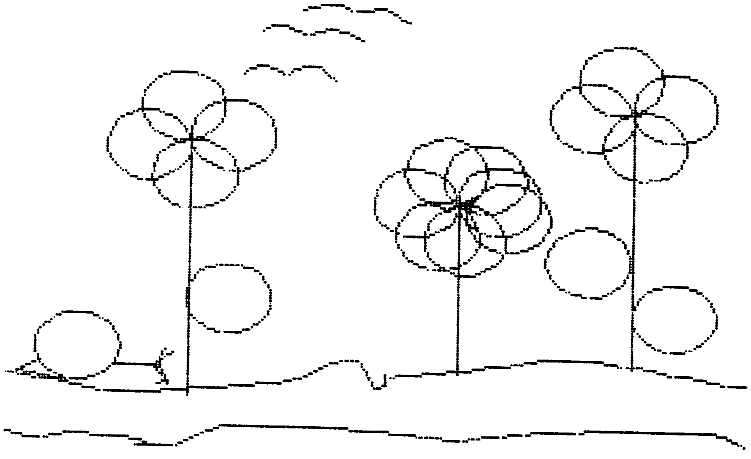
If games controllers are connected to the Apple (see the photographs), a games controller button can be used to move the turtle forward and the dial will set the turtle's direction.

These procedures provide an erasable sketchpad which is fun (even for adults), and introduces many LOGO concepts. The following drawings were made using these procedures, although a lot of fun can be had just by random doodling.

The Apple games controller







TO USE THE TODDLER PROCEDURES

The procedures are listed both for MIT and APPLE LOGO. They introduce a number of new commands to read data from the games controllers. These commands are simple to use and are explained fully in the LOGO manual. Just type in the procedures (a few minutes work) and use them. Don't forget to save them on a diskette for later use.

To use the procedures (once you have them in the workspace), type

```
STARTDOODLE 10
```

The screen will clear, and start with the turtle at HOME.

The following commands can be used from the keyboard.

F	forward
B	back
L	left 30 degrees
R	right 30 degrees
U	pen up
D	pen down
C	circle
S	square
T	triangle
W	clear screen

The RETURN key is NOT pressed after each command.

If you have games controllers, then the button has the same effect as F and the dial sets the turtle direction.

The input to STARTDOODLE is the number of turtle steps that the turtle moves forward or back for one command. For random doodling 10 is a good figure. With smaller numbers less happens each time, but you have better control over turtle movements; try some different inputs.

To stop the procedures, press CTRL-G.

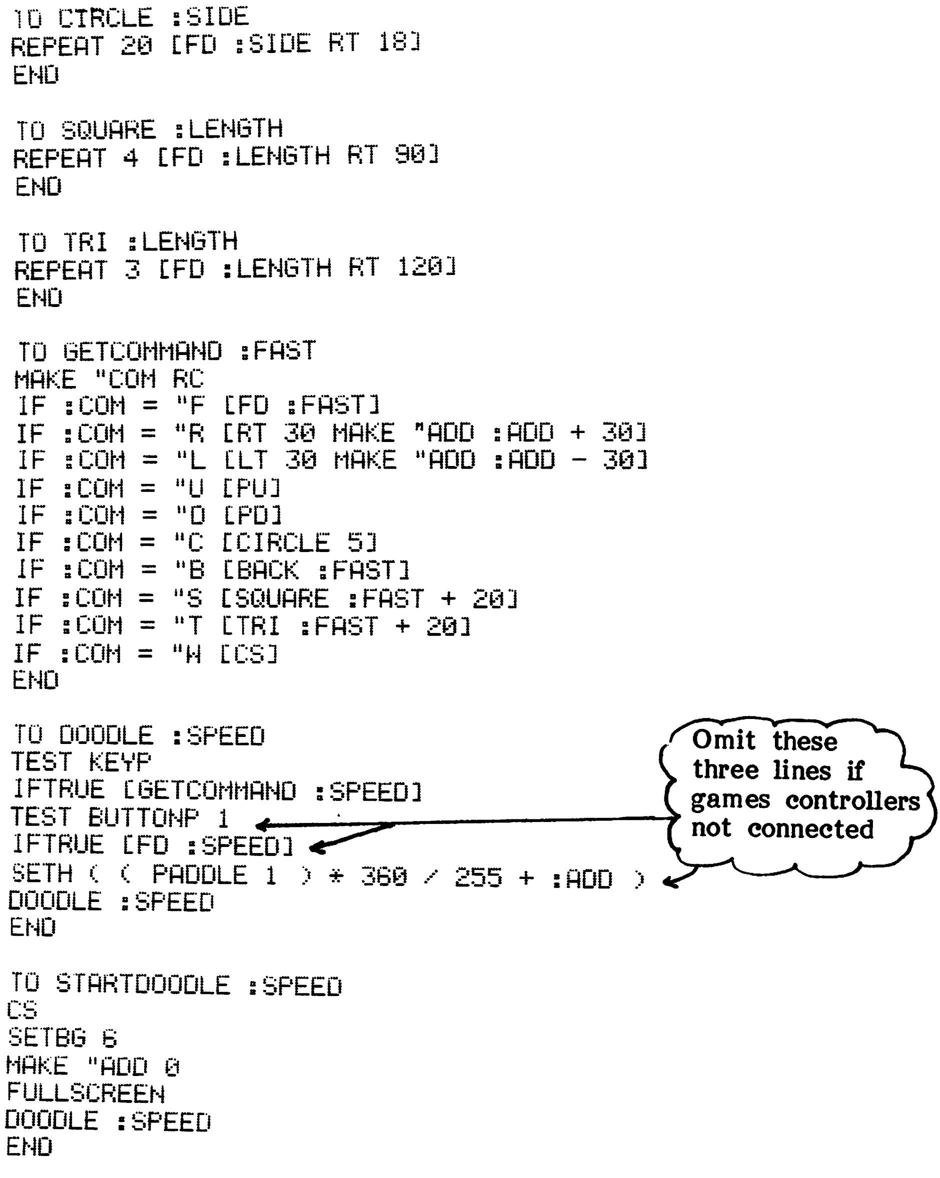
Like many other LOGO procedures, these provide a basis for your own extensions. Additional commands could be added to GETCOMMAND for such tasks as printer drawings, different circles, houses, spirals, flowers, stars, and so on. All it is necessary to do is to have a procedure (such as CIRCLE) in the workspace and to incorporate a letter to select the procedure in GETCOMMAND.

Have fun.

```
TO STARTDOODLE :SPEED
 DRAW
 BACKGROUND 6
 MAKE "ADD 0
 FULLSCREEN
 DOODLE :SPEED
END

TO GETCOMMAND :FAST
 MAKE "COM READCHARACTER
 IF :COM = "F FD :FAST
 IF :COM = "R RT 30 MAKE "ADD :ADD + 30
 IF :COM = "L LT 30 MAKE "ADD :ADD - 30
 IF :COM = "U PU
 IF :COM = "D PD
 IF :COM = "C CIRCLE 5
 IF :COM = "B BK :FAST
 IF :COM = "S SQUARE :FAST + 20
 IF :COM = "T TRI :FAST + 20
 IF :COM = "W DRAW
END

TO CIRCLE :SIDE
 REPEAT 20 [FD :SIDE RT 18]
END

TO SQUARE :LENGTH
 REPEAT 4 [FD :LENGTH RT 90]
END

TO TRI :LENGTH
 REPEAT 3 [FD :LENGTH RT 120]
END

TO DOODLE :SPEED
 TEST RC?
 IFTRUE GETCOMMAND :SPEED
 TEST PADDLEBUTTON 1
 IFTRUE THEN FD :SPEED
 SETHEADING ( ( PADDLE 1 ) * 360 / 255 + :ADD )
 DOODLE :SPEED
END
```

Omit these three lines if games controllers not connected

APPLE LOGO PROCEDURES



```
TO CIRCLE :SIDE
REPEAT 20 [FD :SIDE RT 18]
END

TO SQUARE :LENGTH
REPEAT 4 [FD :LENGTH RT 90]
END

TO TRI :LENGTH
REPEAT 3 [FD :LENGTH RT 120]
END

TO GETCOMMAND :FAST
MAKE "COM RC
IF :COM = "F [FD :FAST]
IF :COM = "R [RT 30 MAKE "ADD :ADD + 30]
IF :COM = "L [LT 30 MAKE "ADD :ADD - 30]
IF :COM = "U [PU]
IF :COM = "D [PD]
IF :COM = "C [CIRCLE 5]
IF :COM = "B [BACK :FAST]
IF :COM = "S [SQUARE :FAST + 20]
IF :COM = "T [TRI :FAST + 20]
IF :COM = "H [CS]
END

TO DOODLE :SPEED
TEST KEYP
IFTRUE [GETCOMMAND :SPEED]
TEST BUTTONP 1
IFTRUE [FD :SPEED]
SETH ( ( PADDLE 1 ) * 360 / 255 + :ADD )
DOODLE :SPEED
END

TO STARTDOODLE :SPEED
CS
SETBG 6
MAKE "ADD 0
FULLSCREEN
DOODLE :SPEED
END
```

Omit these three lines if games controllers not connected

APPENDIX F

MANAGING THE WORKSPACE

If you have worked systematically through the early chapters, and have not switched off the computer meanwhile, you should by now have quite a collection of procedures stored in the workspace in the computer's memory.

To check exactly which procedures are presently in your workspace, type

```
POTS  (standing for
       PrintOut TitleS)
```

The computer should print out the titles of all the procedures in the workspace; for example

```
SQ
SQ1
TR
PG
HOUSE
WALLS
ROOF    etc
```

The procedures listed will be those defined since you last turned the computer on, so they will not necessarily be the same as those listed above. If you haven't defined any procedures since turning the computer on, then define two or three now, perhaps from Chapter 1.

Now review the commands in one of these procedures by typing

MIT LOGO	APPLE LOGO
PO HOUSE	PO "HOUSE

(If HOUSE is not in the workspace use the title of a procedure that is.)

The computer should list the commands in the specified procedure, so for HOUSE it should print

```
TO HOUSE
WALLS
FD 50
LT 30
ROOF
END
```

A procedure you no longer want can be removed from the workspace using the ERASE command. Type

ERASE SQ1	ERASE "SQ1

Then type

```
POTS
```

to check that the SQ1 procedure is no longer in the workspace.

SAVING THE WORKSPACE

The workspace contains a collection of defined procedures, which remain there, to be used over and again as needed, unless you erase them or switch off the computer. If the computer is switched off, all the information in its internal memory is lost.

In order to use our work from one session to the next, we need a more permanent way to store LOGO procedures. This is provided by a magnetic diskette, similar in appearance to the LOGO language diskette itself, but different in function. Once a diskette has been initialized (see below) the workspace in the computer's internal memory can be copied onto the diskette, and this can be kept when the computer is switched off. At a later session, the diskette file can be copied back into the computer's memory, and the workspace used as before.

PREPARING LOGO DATA DISKETTES

The LOGO language diskette should be placed safely back into its cardboard folder as soon as LOGO is loaded (started up). To save your own procedure files you need to use a data diskette. This can be a diskette that has been used to save programs or data from the BASIC language, or previously from LOGO, or it can be a brand new unused diskette. If you are using data diskettes that have been used before with other LOGO procedures or BASIC programs, then skip the next section and go to LOGO DISKETTE FILES.

BRAND NEW DISKETTES (which have never been used before)

Before a new diskette can be used to store your procedures it must go through a process called initialization. A diskette should only be initialized when it is new. WHEN YOU INITIALIZE A DISKETTE ANY PREVIOUS INFORMATION ON IT IS DESTROYED. So when you are preparing to initialize diskettes it is a good idea for them to be clearly labelled and for all other diskettes to be safely stored away.

**** YOU HAVE BEEN WARNED ****

Now carry out the following steps carefully if your computer is an Apple II plus with Applesoft BASIC. (For Integer BASIC computers consult the Apple DOS 3.3 manual).

1. Obtain a DOS 3.3 Master Diskette. This diskette is provided with every Apple computer.

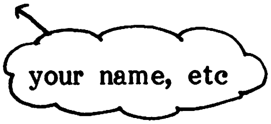
2. Turn the computer off. This will mean that you will lose any procedures in your workspace. Place the DOS 3.3 Master Diskette into the disk drive and turn the computer's power on. The DOS 3.3 programs will be loaded. After a few seconds in which the disk drive hums, APPLE II will be displayed at the top of the screen and the] prompt will appear on the bottom left of the screen.

3. Put the DOS 3.3 Master Diskette safely away.

4. Place the new diskette into the disk drive (remember YOU HAVE BEEN WARNED).

5. Now type

```
NEW
10 PRINT "LOGO DISKETTE"
20 PRINT "IRVING JONES 3RD DEC 1982"
30 END
```



Don't forget to press RETURN at the end of each line.
If the message

```
SYNTAX ERROR
```

appears or you make a typing mistake, just press RETURN and retype the line.

Now type

```
INIT HELLO
```

The disk drive will now hum for about 2 minutes, making clicking sounds as it initializes the diskette. You can now store your LOGO procedures on this diskette.

Remove the initialized diskette from the drive for a moment. Now load the LOGO language from its diskette as you were shown in Chapter 1, and then place your data diskette back in the drive. Define two or three procedures, perhaps from Chapter 1.

LOGO DISKETTE FILES

The workspace, and all the procedures it contains, can be saved as a file on the diskette. To do this you must first think of a name for the file. This diskette file will contain all of the current workspace, not just a single procedure, so the name might reflect this. Possible file names are POLYGONS, MONDAYWORK, MYSHAPES and so on. Once you have decided on a file name, the workspace can be saved, using the SAVE command with the file name. To save the present workspace with file name SHAPES1, make sure the initialized LOGO data diskette is in the drive, and type

```
SAVE "SHAPES1
```

The workspace and its contents are still in the computer's memory, but a copy now exists on the diskette, stored in a file called SHAPES1.

There is room on a diskette for many files, so a later SAVE command for a file of another name could save a later or different workspace, a later session's work perhaps. This would reside on the diskette as well as SHAPES1.

Save another copy of the present workspace, by typing

```
SAVE "SHAPES2
```

You can list the files stored on the disk with a CATALOG command. Type it now.

```
CATALOG
```

The computer should respond with the two file names

```
SHAPES1.LOGO
SHAPES2.LOGO
```

A file you no longer want can be removed from the disk using the ERASEFILE command. Type

```
ERASEFILE "SHAPES1
```

Note that you do not need to type the .LOGO after the name of the file. Now type

```
CATALOG
```

to check that the SHAPES1 file is no longer on the disk.

It is a good idea to save a new session's work on a project with a new but related file name, such as SHAPES2, SHAPES3 and so on, and to retain one or two earlier files of the project workspace in case you need to refer back to them.

A workspace in the computer's memory can be recreated by copying the diskette file back into the computer's memory.

Switch off the computer to clear the present workspace. Remove the LOGO data diskette, put the LOGO language disk into the drive, and switch the computer on. When the disk drive stops humming, type

```
POTS
```

to examine the new workspace. It should contain no procedures.

Now remove the LOGO diskette, and replace the data diskette in the drive. Type

```
CATALOG
```

to check the file names on the diskette. The computer should list one file

```
SHAPES2.LOGO
```

To copy this file into the computer's memory, type

```
READ "SHAPES2          LOAD "SHAPES2
   MIT LOGO            APPLE LOGO
```

Note that when the red light is on in the disk drive the computer is using it. When the light goes out the computer is finished and you can take the diskette out of the drive if you wish.

Now type

```
POTS
```

to examine the workspace. The computer should list the procedure titles that were in the workspace that was saved, for example

```
SQ2
SQ3
TRI
PENTAGON
HOUSE
```

Note that

```
ERASE ALL          ERALL
```

will delete all procedures in the workspace, so you don't have to turn the machine off each time.

PROTECTING DISKETTES

On the right hand edge of a diskette you will see a small notch. If this notch is uncovered then the diskette can have files written on it, be initialized (perhaps accidently) and otherwise changed. If this notch is covered then the computer will be prevented from changing the contents of the diskette in any way. Small sticky tabs are provided with new diskettes to enable you to protect your files.

If you want to change the contents of a diskette that has the notch covered, simply remove the tab.

Diskettes are reasonably reliable, but because of imperfections in the recording material, accidental damage (bending, folding or exposure to magnetic fields) or faults in the disk drive itself, they can become unusable. The safest approach is always to have two copies of your data stored. So every time you store your files, carry out the same SAVE command with two diskettes in turn. If you have a printer available then you should print the CATALOG of each diskette, and keep it with the diskette. It is also a good idea to print out all the procedures that you have saved.

You should ensure that your diskettes are well labelled.

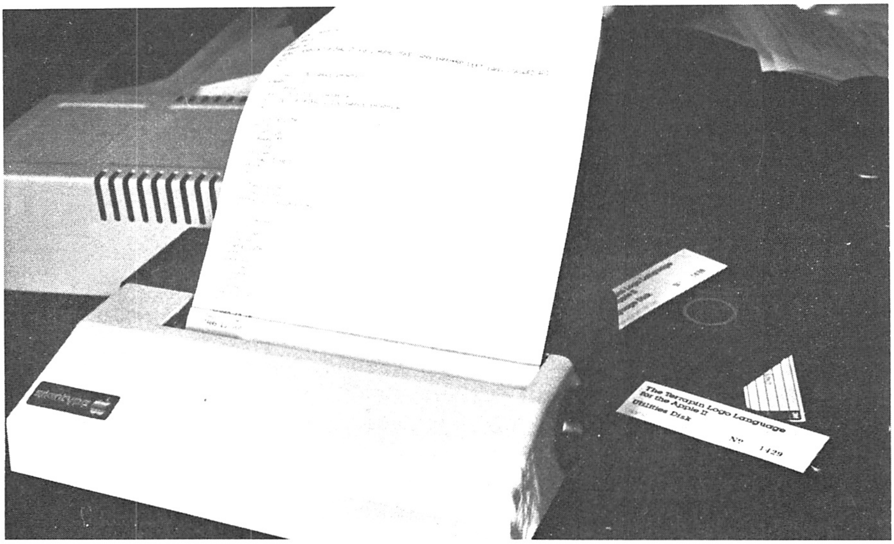
If you have your workspace saved with a file name of THURS and you save the workspace again with the same name (THURS), then in MIT LOGO the first file will be overwritten by the second. You will have lost the first file of information. In APPLE LOGO you must first erase the file (see ERASEFILE) before you can save a workspace with that file name again.

When you read a file back into the workspace, any procedure titles in the workspace that have the same title as a procedure in the file will be overwritten by the procedure from the file. You will lose the procedure that was already in the workspace.

SUMMARY OF WORKSPACE AND FILE CONCEPTS

The workspace refers to the internal memory of the computer. When you define procedures they exist in the workspace until you erase them or turn off the power to the computer. You can save the workspace at any time onto a data diskette that has been previously initialized. The workspace is saved onto a file on the diskette; the diskette can contain many files with different names. Any one of these files can be read back from the diskette at a later time, to create a new workspace in the computer.

PRINTING PROCEDURES



A printer is connected to the Apple computer by inserting a printer interface card into a numbered slot inside the computer. The usual position for a printer interface card is slot number 1. The command to send output to slot 1 is

MIT LOGO	APPLE LOGO
OUTDEV 1	.PRINTER 1

Type this, and subsequent computer output should be sent to the printer. (If not, check whether your printer is in fact connected to another slot in the Apple.).

For example, to list on the printer a procedure in the workspace, type the following command with the title of the procedure (in this example we will print out the contents of the HOUSE procedure).

```
PO HOUSE                PO "HOUSE
```

The commands

```
TO HOUSE
SQ2
FD 50
LT 30
TRI
END
```

should be printed. List the commands of some other procedures, using the PO command. When you have finished, type

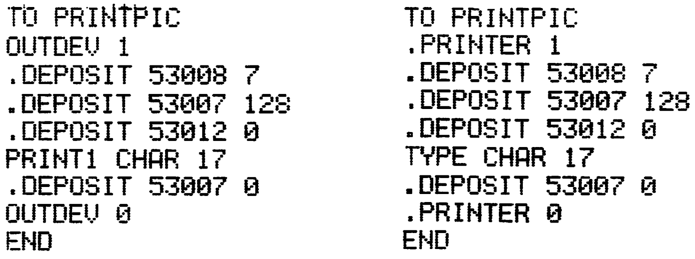
```
    OUTDEV 0                    .PRINTER 0
```

to return to using the screen again.

PRINTING PICTURES

The exact method to use for producing LOGO pictures on the printer depends on the type of printer you are using. Some printers are able to produce copies of pictures drawn on the screen; with others the picture must be saved on a diskette, then printed subsequently (MIT LOGO only). Still other printers cannot be used for printing pictures at all.

We will give here a procedure for printing a picture directly from the screen onto an Apple Silentype printer connected to slot 1.



```
TO PRINTPIC                TO PRINTPIC
OUTDEV 1                   .PRINTER 1
.DEPOSIT 53008 7           .DEPOSIT 53008 7
.DEPOSIT 53007 128         .DEPOSIT 53007 128
.DEPOSIT 53012 0           .DEPOSIT 53012 0
PRINT1 CHAR 17             TYPE CHAR 17
.DEPOSIT 53007 0           .DEPOSIT 53007 0
OUTDEV 0                   .PRINTER 0
END                        END
```

This procedure uses deposit commands to control the printing of the picture. The procedure is explained in the MIT LOGO Technical Manual, and the commands are explained in the MIT LOGO Technical Manual and in the APPLE LOGO Reference Manual.

Define and use a procedure. Then when you have a picture on the screen, type

```
PRINTPIC
```

Picture printing methods for some other printers are provided in the MIT LOGO Technical Manual.

APPENDIX G

PROCEDURES USED IN THE BOOK

Chapter 1

SQ	Square
TR	Triangle
SQ1	Larger square
TR1	Larger triangle
SQ2	Square, used as procedure demonstration

Chapter 2

PG	Pentagon
PENTAGON :SIDE	Pentagon, used as demonstration of inputs
POLY :SIDE :ANGLE	Polygon, using REPEAT
SQ	Square
EX	Editing exercise
MYNAME	Editing exercise
WALLS	Walls of HOUSE, a square without final turn
WALLS1	Walls of house
ROOF	Roof of house
HOUSE	House with walls and roof, debugging exercise

Chapter 3

EAT :S :A	Rotation
ROTATE :S :A	Rotation that includes EAT
ROTATE2 :S :A	Similar rotate procedure
LOLLIPOP	Candy on stick
POLY :S :A	Polygon
CRAZY	House without lifting pen
COTTAGE	House with eaves
SIDES	Walls of cottage
ENTRANCE	Entrance to cottage
EAVES	Eaves of cottage
SLIDEL :D	Slide left
SLIDER :D	Slide right
AVENUE	Avenue of cottages
TIME	Time on clock
CLOCK	Clock figure
HANDS	Hands of clock
NOSE	Nose of plane
FUSELAGE	Fuselage of plane
WINGS	Wings of plane
TAIL	Tail of plane
TL	Part of plane
PLANE	Delta wing plane
BODY :S	Body of stick figure
HEAD	Head of stick figure
POLY :S :A	Part of head
ARMS	Arms of stick figure
LEGS	Legs of stick figure
MAN	Stick figure
MARCH	Marching stick figures
MARCH1	Marching stick figures
MARCH2	More marching stick figures
A	Part of hangman
B	Part of hangman
C	Part of hangman
D	Part of hangman
E	Part of hangman
F	Part of hangman
G	Part of hangman
H	Part of hangman
I	Part of hangman
J	Part of hangman
K	Part of hangman
L	Part of hangman
POLY :S :A	Part of hangman
HANGMAN	Hangman

Chapter 4

BACKWARDS :NUMBER	Count backwards
BACK2 :NUMBER	Count backwards
SPI :LENGTH :ANGLE :INC	Spiral
POLY :SIDE :ANGLE	Polygon
POLY1 :SIDE :ANGLE	Recursive polygon
GROWSQUARE :SIDE	Growing squares
SQUARE :SIDE	Square
GROWTRIANGLE :SIDE	Growing triangle
TRIANGLE :SIDE	Triangle
SPINGS	Spinning, growing square
SPINGT	Spinning, growing triangle
SPINGT1	Spinning, growing triangle
BOW	Bow tie
SPINCIRCLE :ANGLE	Spinning circle
CIRCLE	Circle
SPITRI :SIDE :ANGLE :INC	Spinning, spiralling triangle
SPISQUARE :SIDE :ANGLE :INC	Spinning, spiralling square
TUNNEL :ANGLE	Repeated circles
RCIRCLE :ANGLE	Circle with variable input
EYES	Two tunnels
SELECT :NUMB	Using IF and TEST commands
SPIR :SIDE :ANGLE :INC	Inwards spiral
WALLPAPER :INC	Uses SPIR to wrap a pattern
WINDMILL :ANGLE	Spins a windmill blade
BLADE :ANGLE	The windmill blade
CRESCENT	Crescents for the blade

Chapter 5

POLYSPI :SIDE :ANGLE	Spiral
PRINTIT	Print demonstration
READNUM	Extract number from list
ADD :NUMBER	Continuous addition by 1
DOUBLE :NUMBER	Continuous double
DOUBLE1	Doubling procedure
STOPWATCH :COUNT	Computer stopwatch
TOSS	Simulation of coin tossing
COUNTOSS	Simulate 100 coin throws
ADDUP	Add coin throws
TAN :ANGLE	Tangent of angle
GETRADIUS :AREA	Get radius of circle
PYTHAG1 :A :B	Calculate side of triangle
PYTHAG2 :A :B	Calculate side of triangle

CALCSIDE	Uses PYTHAG2
CALCSIDE1	Calculate and draw side of triangle
TRIDRAW :A :B :H	Draw triangle

Chapter 6

PRINTDATE	Demonstrate sentence
EXAMINELIST :A	Demonstration of lists
EXAMINELIST1 :A	Demonstration of lists
PRETTYLIST :A	Demonstration of lists
FIND	Find element of list
SEARCH :A :B	Test element in list
FIND1 :GUESS	Element in list, true or false
BUILDLIST :NAME	Build list with SENTENCE
EXAMINEWORD :A	Demonstration of words
EXAMINE :A	Procedures for words/lists
EMPTYP :ELEMENT	Test empty list or word
CHECKLIST :ELEMENT	Check if list
CHECKWORD :ELEMENT	Check if word
ADD1 :B	Global and local variables

Chapter 7

VOWELQ :LETTER	Test if letter is a vowel
PRINTNOVOWEL :SENT	Prints list without vowels
REMOVEVOWEL :NEXTWORD	Removes vowels from word
REVERSELIST :SENT	Reverses list
SCRAMBLE :MESSAGE	Code a message
SCRAMBLEWORD :NEXT	Code a word
GETPOS :CHR :LNAME :COUNT	Position of character in list
GETCHAR :NUMB :LNAME	Gets character from known position
UNSCRAMBLE :MESSAGE	Decode message
UNSCRAMBLEWORD :NEXT	Decode word

Chapter 8

RAND :DICT	Print random word from list
GETRANDOM :DICT :NUMB	Get random word from list
LENGTH :DICT :COUNT	Determine length of list
WRITE :TEMPLATE	Write a random sentence, version 1
WRITE :TEMPLATE	Write random poetry or sentences, version 2

Appendix A

nil	----

Appendix B

nil	----

Appendix C

CASUARINA	A project to draw a tree
FRUIT	Fruit for tree
FRUITA	Fruit for tree
BRANCH	Branch of tree
STALK	Stalk of tree
LEAVES1	Leaves of tree
LEAF	Leaf of tree
LEAVES2	Leaves of tree
FRUIT2	Fruit of tree
FRUIT1	Fruit of tree
TRI	Part of fruit
FRUITB	Fruit of tree
STREET1 :SIDE	Street of variable sized houses
HOUSE8 :SIDE	Variable sized house
PUFF1	Puff of smoke
SMOKE1	Smoke from chimney
CHIMNEY1	Chimney for house
WINDOW1	Window for house
Pane	Pane for window
DOOR1	Door for house
SQUARE2	Square with variable sides, but no inputs
TRIANGLE	Triangle with variable sides, but no inputs
HOUSE6	Variable sized house drawing procedure

Appendix D

FILLINSQ :GR :PN :COUNT	Color demonstration and test

Appendix E

STARTDOODLE :SPEED	Toddler procedure
DOODLE :SPEED	Use games controllers to draw
GETCOMMAND :FAST	Single character LOGO commands
CIRCLE :SIDE	Circle for toddlers
SQUARE :LENGTH	Square for toddlers
TRI :LENGTH	Triangle for toddlers

Appendix F

PRINTPIC	Print LOGO drawing on Apple Silentype printer

Appendix G

nil	----

Appendix H

nil	----

APPENDIX H

BIBLIOGRAPHY

Abelson, H., APPLE LOGO, Byte/McGraw-Hill, 1982.

Abelson, H., LOGO FOR THE APPLE II, Byte/McGraw-Hill, 1982.

Abelson, H. and DiSessa, A., TURTLE GEOMETRY, The M.I.T. Press, 1981.

Abelson, H. and Klotz, L., LOGO FOR THE APPLE II: TECHNICAL MANUAL, Terrapin Inc., 1982.

Apple Computer Inc., THE APPLESOFT TUTORIAL, Apple Computer Inc., 1981.

Apple Computer Inc., THE DOS MANUAL, Apple Computer Inc., 1981.

BYTE, Vol.7, No.8, McGraw-Hill, August, 1982.

Davidson, L., APPLE LOGO REFERENCE MANUAL, Logo Computer Systems Inc., 1982.

Howe, J., "Teaching Mathematics Through Programming", Research Paper No. 129, Department of Artificial Intelligence, University of Edinburgh, 1980.

Howe, J., O'Shea, T. and Plane, F., "Teaching Mathematics Through LOGO

Programming: An Evaluation Study", in Lewis, R. and Tagg, W. (eds.). COMPUTER ASSISTED LEARNING, North-Holland, 1980.

McDougall, A. and Adams, T., "LOGO Environments: The Development of the Language and Its Use in Education and Research", PROCEEDINGS OF THE NINTH AUSTRALIAN COMPUTER CONFERENCE, Australian Computer Society, 1982.

Papert, S., MINDSTORMS: CHILDREN, COMPUTERS AND POWERFUL IDEAS, Harvester, 1980.

Sharples, M., "A Computer Written Language Lab", Research Paper No. 134, Department of Artificial Intelligence, University of Edinburgh, 1980.

Sharples, M., "A Computer Based Teaching Scheme for Creative Writing", in Lewis, R. and Tagg, W. (eds.), COMPUTERS IN EDUCATION, North-Holland, 1981.

Solomon, C., APPLE LOGO: INTRODUCTION TO PROGRAMMING THROUGH TURTLE GRAPHICS, Logo Computer Systems Inc., 1982.

Watt, D., "Final Report of the Brookline LOGO Project, Part III: Profiles of Individual Students' Work", LOGO Memo No. 54, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1979.

Wills, S., "Computers in Tasmanian Schools", THE AUSTRALIAN COMPUTER BULLETIN, August, 1980.

INDEX

LEARNING LOGO ON THE APPLE II

Here's an introduction to the exciting new language LOGO.

LOGO is

EASY

it's the quickest and easiest way into computing

SATISFYING

you'll be writing programs within minutes of
sitting down at your Apple II

MULTIFACETED

you can do anything with LOGO
that you could using very advanced BASIC

VERSATILE

you can produce fascinating graphics, write poetry,
solve mathematical problems and do a host of other exciting things
with just a few commands

This practical, hands-on introduction to LOGO uses both versions of LOGO available for the Apple: MIT LOGO and Apple LOGO.

The eight chapters are packed with interesting things to do:

Starting Up the Turtle
Editing and Debugging Procedures
Turtle Projects
Recursion and More Turtle Projects
Naming Things and Doing Arithmetic
Recursion and Lists
Secret Codes
Creating a Computer Poet

There are scores of suggestions for experimenting further and loads of ideas for projects to undertake on your Apple.
Whether you're a first time programmer or have already used another language, you'll find LEARNING LOGO fun!

® Apple is a trademark of Apple Computer Inc.

PHA
0 7248 0732 2