LOGOTRON
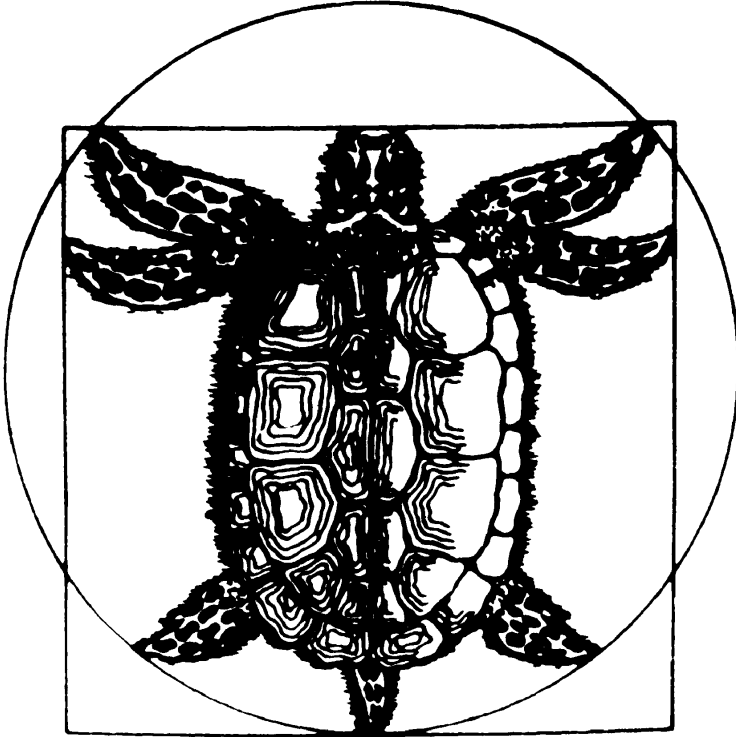

◄LOGO►

**The LCSI Standard Logo for the BBC Model "B"
by Systèmes d'Ordinateurs Logo International © 1984.**

                    LOGOTRON LIMITED
            5 GRANBY STREET, LOUGHBOROUGH
                LEICESTERSHIRE LE11 3DU
                   TEL. 0509 230248

        SAVING PROCEDURES ON TAPE AND ECONET

Some customers have reported problems with the primitive
SAVE. when trying to save files on cassette or on Econet
Fileserver.

The way round this problem is to create two small procedures.
The first saves your entire workspace and the second saves
named procedures or global variables. There is no problem
with the primitive LOAD on either cassette or Econet.

```
TO SSAVE  :  FILENAME
(* SPOOL  : FILENAME)  POALL  (*SPOOL)
END

TO SSSAVE  : FILENAME  :  PROCNAMES
(*SPOOL  : FILENAME)  PO  : PROCNAMES (*SPOOL)
END
```

Imagine you have created three procedures, TRIANGLE, SQUARE,
PENTAGON and one global variable, ANGLES.

SSAVE "SHAPES saves the entire workspace in a file called
SHAPES.

SSSAVE "SHAPES "SQUARE would save the single procedure
"SQUARE in the file.

SSSAVE "SHAPES [SQUARE TRIANGLE "ANGLES] would save two
procedures and the global variable ANGLES.

In short the procedures work exactly as the primitive
SAVE as documented in the manual.

CONTENTS

SECTION ONE – INSTALLATION

Congratulations, you have bought the Logotron Logo, produced by Systèmes d'Ordinateurs Logo International, or SOLI for short. This is undoubtedly the most advanced Logo available for the BBC Micro, and at the time of its implementation can claim to be the most advanced Logo on any 8-bit micro in the world.

This is the time to register as a Logotron Logo user. It provides you with a valid guarantee. It also entitles you to information about all the supplementary software available to Logotron users. This is offered to register users at a substantial discount on the full retail sale price. Your registration card is enclosed in the box with this manual.

You want to get started. If the ROM has already been installed, you can skip straight to the next section of this manual. Here is how you check. Turn on the computer. It may say:

```
(c) 1984 ACT/SOLI
WELCOME TO LOGO
?
```

In which case you are in business. Even if you do not receive this rousing welcome, it is worth typing

```
*LOGO
```

This may produce the desired effect

```
(c) 1984 ACT/SOLI
WELCOME TO LOGO
?
```

If not, your ROM needs to be inserted in any of the "sideways" or "paged" ROM sockets. You should find the ROM itself inserted into a piece of plastic foam inside the Logotron loose-leaf binder. Leave it there until you have removed the top of your computer.

BEFORE you begin work on the computer, switch it OFF and REMOVE the mains plug from the power socket. Then follow these instructions:

1. Remove the four screws holding the top on the computer (on early machines, they were marked FIX). There are two of these at the top of the back panel of the computer (you need a posidrive or Philips screwdriver) and two underneath the computer towards the front.

2. When the top is off, release the nuts holding the keyboard in place. This is a good moment to look at the diagram on this page, below. There is no need to disconnect the keyboard completely, simply move it to one side, to expose the sideways ROM sockets.

3. Locate the row of five large sockets at the front right hand corner of the main printed circuit board (see diagram). Two or more of these sockets will already be filled with ROMS. The rightmost four of these sockets, identified as IC52, IC88, IC100 and 1C101 are sideways ROM sockets.

LOGOTRON
LOGO

Half-moon notch

Pin 1

IC51   IC52   IC88   IC100   IC101

BBC Model B
board layout

OS

4 Paged ROM sockets
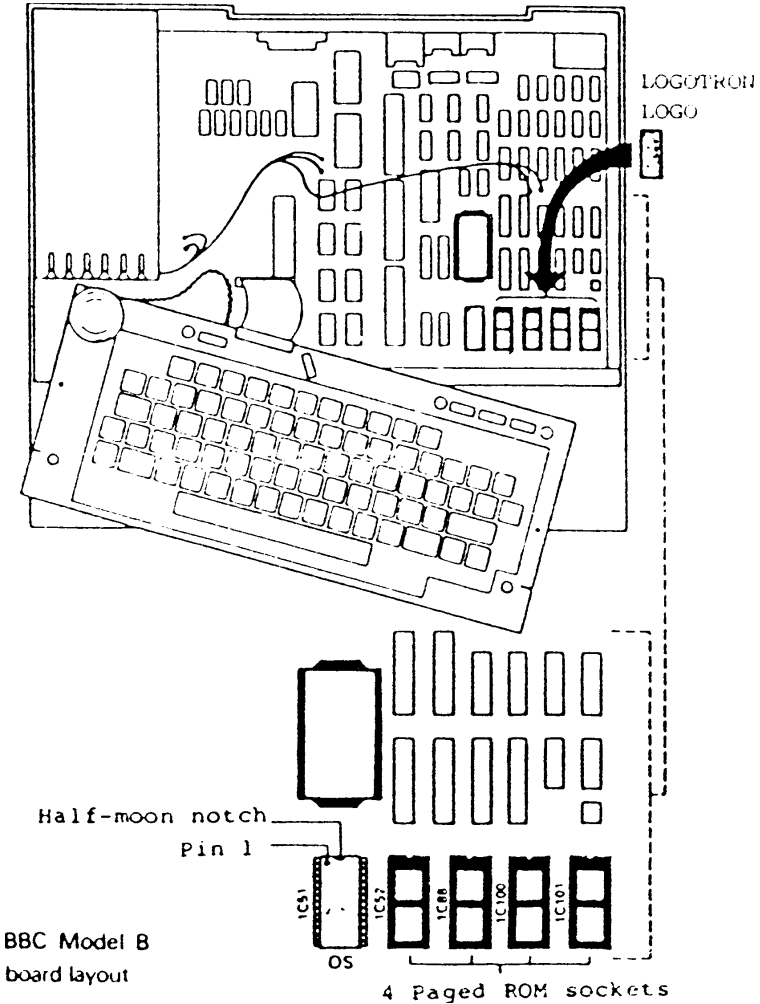
4. You can choose where to put your Logo ROM. If you want Logo to be available as soon as you switch on your computer, then put it on the extreme righthand side. But this is not necessary, you can put it into any empty slot. If you don't have an empty slot, you have three possibilities. You can learn to live without BASIC, or without some other program which is occupying a slot. The second possibility is to buy one of those expansion boards, which allow you to plug in additional ROM. The third possibility is to use one of your ROM slots to create a ROM cartridge system. Whatever you decide, the one bad choice is to be constantly taking ROMS out of the computer and putting them back. One day, one will get damaged.

5. Having decided on the slot your ROM will occupy, it is time to take your Logo ROM out of its plastic foam seating, first locating a semicircular notch at one end (see diagram). This notch tells you which way the ROM goes into the computer. The notch points towards the back of the computer. You will see that all the other ROMS are aligned in the same way. **MAKE SURE YOU UNDERSTAND THIS.** Before touching the ROM, it is good practice to earth yourself, by touching a metal desk or radiator. Static electricity can damage electronic components. Try to handle the ROM as little as possible, and avoid touching its metal legs.

6. These legs have to fit into slots along either side of the socket. Make sure they are correctly aligned before pressing the ROM home. It is essential that all the legs are inserted and that none bends outwards or underneath the ROM. If you have never done this before, nor seen anyone else do it, seek help. If necessary, get your dealer to help you. It's not worth making a mistake at this stage.

7. Replace the keyboard and lid, reversing steps 1 and 2, and switch on the computer as normal. Plug in the computer, switch the power to ON, and you should be in business. If not make sure you followed all the steps correctly, checking in particular that all the legs of the ROM are properly seated. If it still does not work, consult the dealer from whom you bought Logo. If possible, take in the machine with the faulty chip installed. This should not happen, as all chips are tested before leaving the factory.

SECTION TWO – TUTORIAL INTRODUCTION

The fact that Logo attracts such a wide variety of people, of all ages and all levels of computer experience, makes it very difficult to write an introduction to the language which is right for everyone.

What we have tried to do in this tutorial part of the manual (**Sections 2 – 19**) is to provide something for almost everyone. I think that anyone who can read, from the age of ten, say, should be able to manage the first three sections, without trouble and without help.

Older children should be able to cope with most of the first 12 sections on their own. Teenagers should find no difficulty with any of the material, and will explore some of the more advanced ideas in the Reference sections of the manual.

If you are already familiar with Logo, or an experienced computer programmer, you can probably skip the tutorial sections of the manual, and go straight on to the reference sections, beginning with **Section 20**.

Sections 17 and 18 are specifically aimed at teachers, and parents who want to help their children with Logo. The first of these sections **(No.17)** explains how you can provide a simplified Logo for children who are too young to read, or who face severe learning difficulties.

The second **(No.18)** is designed to help you to guide children from the realtively easy world of Turtle graphics into the rather more puzzling world of language processing.

The LCSI Standard Logo provided by Logotron for the BBC Micro is a very complete programming system, which will carry users far beyond the realms of Turtle graphics. When used in conjunction with a second processor, Logotron Logo can cope with virtually any programming problem likely to be encountered in school.

You have full access to the operating system of the BBC Micro, through the VDU and ∗FX commands. Furthermore, the system is highly extensible. Additional software is available to drive a Sprite Board and robots. Other extensions are planned to provide advanced programming functions, for use by 'O' and 'A' level students.

If you are already an advanced programmer, then you can use the USE primitive to link up with extensions written in machine code. This will be particularly relevant from early 1985, when we plan to release Advanced Logo, on a disc, to complement the intitial release.

This explains the design of this manual; it is an open-ended. You can bind in your own notes, and details of procedures. There is room for additional documentation, which will come from Logotron in connection with future products. We expect teachers may want to make photocopies of some pages, especially where they are dealing with small children, and do not want to confuse them with piles of printed matter.

Logo is widely regarded as a "programming language for children". It also happens to be a "programming language for computer scientists". Much early work in artificial intelligence used Logo, and it is closely related to the leading language for designing expert systems, LISP.

Logo is now in the vanguard of the microcomputer revolution. As home computers grow in memory power and speed, Logo will grow with them, infinitely extensible. While BASIC will become a forgotten curiosity, a fossil of the early days of microcomputers.

The most important feature of Logo is that you can make it reflect your needs, interests and personality. Most early educational software offered an implicit model, in which the computer was the teacher, explaining, questioning and encouraging. The child's role was reactive, learning from the computer, by responding appropriately to its prompting.

Computer Assisted Learning and Computer Based Training all accepted this model. Logo offers a completely different model, diametrically opposed. In our model, **the child (user) is the teacher**, while the computer learns. **The child is active, and the computer reactive.**

But don't take our word for it. Get cracking. This manual is designed to be used, sitting at the computer, teaching it what to do next.

Each section provides enough work for a single session if you are a complete beginner. Do try out all the examples. The text does not make much sense on its own without

hands on practice. Do not hesitate to turn to the reference sections for further details of how to use the system.

Where we expect you to type on the key board, the words you have to type are written in red. Occasionally, we refer to a key, which has to be pressed, such as the RETURN key or the ESCAPE key. These words, too, appear in red. If you have to press two keys at once, for example the CTRL key and C, we would write CTRL C. When we talk about the red function keys at the top of your keyboard, we write about F0, F1, F2 ......etc.

SECTION THREE – MEET THE TURTLE

You can start using Logo with a very few words:
FORWARD, BACK, LEFT and RIGHT. We write Logo words
in CAPITAL LETTERS. So it may help to press the CAPS
LOCK key at the bottom left of your keyboard.

Let's try them. Type FORWARD 300 and press the RETURN
key. A little triangle appears in the middle of the screen, and
darts forward, drawing a line.

If you make a mistake typing FORWARD 300 and type
FROWARD, TORWARD, FORWORD or something, the
computer will say

     I DON'T KNOW HOW TO FROWARD

We call this a **Logo message**. Do not worry about it. Just
type the line again, checking you have it right before
pressing the RETURN key.

We call the little triangle a Turtle. We call its drawings Turtle
graphics, pictures drawn by a Turtle.

Try some more Turtle graphics. Type

     FORWARD 200
     RIGHT 90
     FORWARD 150
     LEFT 90
     FORWARD 100

pressing the RETURN key after each line. Make sure to
leave a space between the words. If you type
FORWARD100, for example, you will read another puzzled
message from your computer

     I DON'T KNOW HOW TO FORWARD100

Don't worry about such messages. Just start again and leave
a gap. One of the biggest differences between a language
like English and a computer language like Logo is that
computers want you to spell words just so, using the same
letters every time. This can be a bore.

Now type CS to Clear the Screen and try again. Change the
numbers to see what happens. Type BACK instead of

FORWARD. Write all the words on one line, like this

    FORWARD 500 RT 90 BACK 200 RIGHT 90 BACK 150 LEFT 90 BACK
    100

and only press the RETURN key at the end of the line. If you spot a mistake before pressing RETURN, you can rub out the words you have written by using the DELETE key. You can change anything until you have pressed the RETURN key.

Once you have pressed RETURN, the computer tries its best to carry out your instructions. If there is a something it doesn't understand, it complains.

But you don't have to type out everything again. Use the UP-ARROW key (top right hand side of the keyboard) to move the cursor up to the first character of the last line you typed. Now press the COPY key, and you should see a copy of the line you typed previously, appearing on the screen.

Using the arrow keys to move the cursor around the screen, you can use the COPY key to copy any writing from the screen into a new line. You can learn more about this from your BBC USER Guide, (Pages 29 and 30). This describes the use of the COPY key in a BASIC program. It works just as well with Logo.

Try typing CLEAN instead of CS. Discover the difference between the two commands.

There are just two more useful words to know when playing with Turtle graphics for the first time. They are PU, standing for Pen Up, and PD, standing for Pen Down. If you type:

    FORWARD 100
    PU
    FORWARD 100
    PD
    FORWARD 100

you will quickly understand what these words mean. They are very useful if you want to move the Turtle from one picture to another without leaving a trail. From now on, I won't remind you every time to press RETURN at the end of each line, or whenever you want the Turtle to follow your instructions. (You still have to do it!).

If you want to draw many pictures with the Turtle, you will soon get bored of typing FORWARD, RIGHT, LEFT, BACK, PU, PD. The answer is to use shorter forms like FD for FORWARD. In this book, when we introduce a new word we will write its short form in brackets after the long form, as follows: FORWARD (FD), BACK (BK), LEFT (LT), RIGHT (RT), PU, PD.

So try out some pictures using FD, BK, RT, LT, PU and PD. The words are easier to type, so there is less chance of making mistakes.

Try to discover how many steps the Turtle has to take from the very bottom of the screen to the very top. And how many steps from one side to the other. Three more words to explore at this stage are WINDOW, WRAP and FENCE. Type the following.

```
FD 600
```

The Turtle disappears at the top of the screen and reappears at the bottom. Its path wraps around the screen as if the top edge were attached to the bottom edge. Type

```
WINDOW FD 1000
```

The Turtle disappears off the top of the screen, and is lost to view. It is as if the screen were merely a small window on the Turtle's world.

```
CS FENCE FD 600
```

You get a Logo message

```
FD DOESN'T LIKE 600 AS INPUT
```

The Turtle refuses to accept any command which sends it off the screen. It is as if a fence had been built around it. Type

```
WRAP FD 600
```

and you are back in WRAP. Some people like that best. They can keep an eye on the turtle. I like WINDOW best. I imagine the Turtle drawing amazing coloured pictures out into space, where no one can bother it.

The Turtle is always in WRAP, FENCE or WINDOW. When you first turn it on, it is in WRAP. We call this the default value. Once you change it, it stays in that mode until you change it again.

Finally, discover what happens if you use minus (negative) numbers instead of plus (positive) numbers. Try

>   FD –100
>   RT –45

I like that. You could do without LEFT and BACK and just use negative numbers. At the same time. Here's one last word for this section, PE. Try this

>   FORWARD 300 WAIT 120 PE BACK 300

To cancel PE type either PU or PD.


```
FD 200
RT 90
FD 200
```


```
FD 200 RT 135 FD 100 LT 135
FD 200 RT 140 FD 100
```


```
?CLEARSCREEN■
?
```


```
?CLEARSCREEN
?■
```


```
?BK100■
```


```
?BK100
?■
```
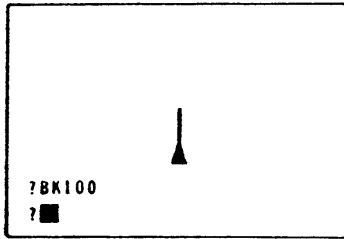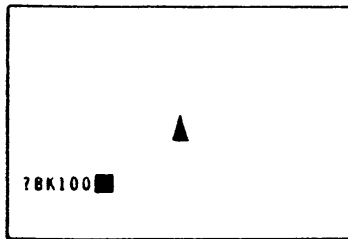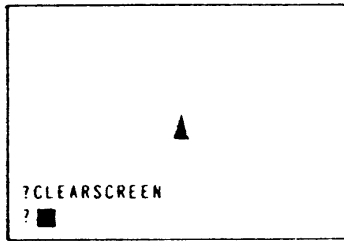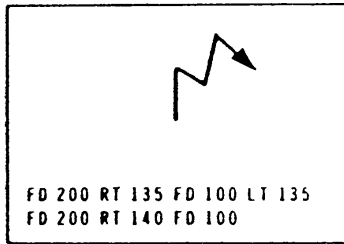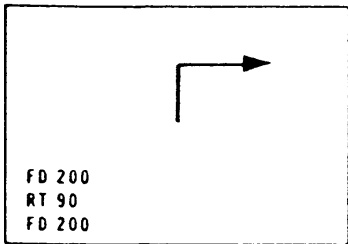
SECTION FOUR – TEACHING LOGO NEW TRICKS

Once you feel comfortable with the commands you learnt in Section 3, FORWARD, BACK, LEFT, RIGHT, PU,
PD, PE, CLEAN, CS, WINDOW, FENCE and
WRAP, you are ready to teach your turtle some very clever tricks.

Try this one for a start. Type

        REPEAT 4 [FD 200 RT 90]
and press the RETURN key. Notice the short forms of the commands FORWARD (FD) and RIGHT (RT). Would you get the same effect if you typed

        REPEAT 4 [BK 200 LT 90]
Either way, the Turtle draws a square. This is a lot easier than typing:

        FORWARD 200 RIGHT 90
        FORWARD 200 RIGHT 90
        FORWARD 200 RIGHT 90
        FORWARD 200 RIGHT 90
You may be wondering what those square brackets [ ] mean. You will see a good deal of them in Logo, so let's explain them once and for all.

The brackets [ ] enclose lists. They can be lists of words, lists of numbers, or even lists of lists. In this case

        REPEAT 4 [FD 200 RT 90]
it is a list of instructions to the Turtle. You can put any instructions you like inside the brackets [ ]. For example:

        REPEAT 5 [FD 100]
It comes to the same thing as

        FD 500
or try this

        REPEAT 2 [FD 250 LT 120]
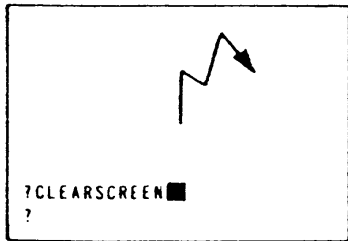Could you change that last command so that the turtle draws a triangle? REPEAT means just what you expect it to mean. It REPEATS a list of instructions just as many times as you want. But lists are used in many other ways. If you want the computer to print out a sentence or a list of

words, you do it like this. Type

    PRINT [APPLES PEARS ORANGES]

    PRINT [GOOD MORNING]

Now I would like you to leave your computer for a minute, and find an open space on the floor. Walk in a circle. Try to think of your actions in turtle steps. Start by walking a square, repeating the commands to yourself. Unless you are in the playground, keep the sides fairly short (3 or 4 steps) as person steps are much bigger than turtle steps.

How could you tell the turtle to draw a circle, using the REPEAT command? Try to walk in a circle using the commands FORWARD and RIGHT.

Walk one step FORWARD and turn a little to the RIGHT. Walk another step FORWARD and turn a little to the RIGHT. Walk another step . . . and so on.

Now go back to the computer and type

    REPEAT ? [FD ? RT ?]

But with numbers instead of question marks. The Turtle will walk round the screen just as you did on the floor. You may find that it has only drawn part of a circle. See if you can complete the circle. Try repeating the instructions more than 20, 30 . . . 100 . . . 150 times. Try telling the turtle to take bigger steps. Say, FORWARD 5, or turning a little more, RIGHT 5.

When you have drawn a circle, try drawing bigger circles and smaller circles.

Use PU and PD to draw circles inside one another, or draw a face. Remember, you can use CLEAN or CS to wipe the screen and start a new drawing.

Perhaps you think the Turtle spoils your pictures, and should disappear when it has finished drawing. Well you can easily fix that with two more words:

HT for HideTurtle
ST for ShowTurtle

Try them. Type HT, and your Turtle has disappeared. Type ST and it's back again. If you give it some instructions while

it is hidden, it draws just the same. Try typing

HT REPEAT 36 [FD 10 RT 10] ST

Just play around with the REPEAT command until you feel really good about it. How long that will take really depends on how old you are, how much you already know about computers, and so on.

REPEAT 60 [FD 5 LT 1]

REPEAT 150 [FD 5 LT 1]

REPEAT 3 [FD 300 RT 120]

REPEAT 2 [LT 30 REPEAT 3
[FD 200 RT 120] RT 90] RT 90
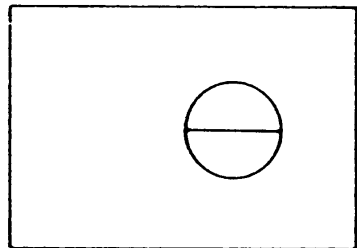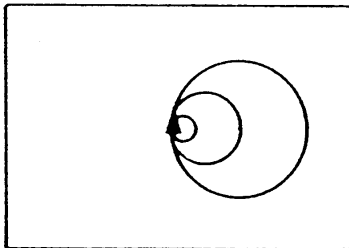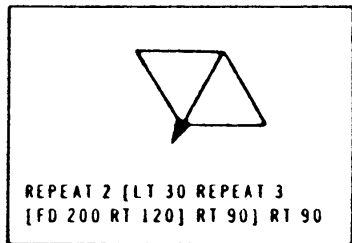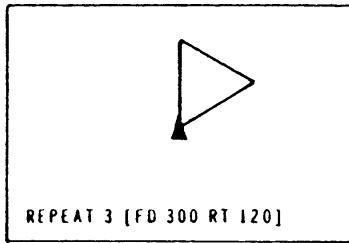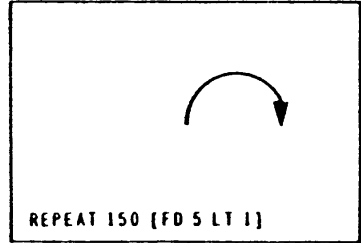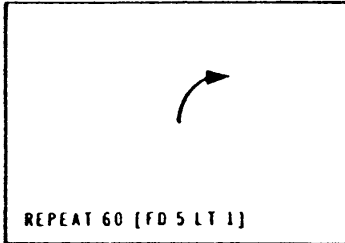
SECTION FIVE – TURTLES CAN REMEMBER

Everything you have learnt so far has been designed to reduce the number of words you have to type. First you learned about shorter words. Then you learned to use REPEAT.

Our motto is: **Make the turtle do the work.**

Now, imagine teaching the Turtle to remember lists of instructions, just like the ones we used with REPEAT.

Leave your computer again. Find a piece of paper and a pencil.

Write at the top of the paper TO SQUARE then underneath, write

[Take five steps forward and turn right. Repeat this action three more times. End]

then write TO OPENTHEDOOR

[Walk across the room. Take hold of the door handle; turn it and pull it towards you. If it opens, stop. If it does not open, then try pushing it. End]

Give the paper to a friend, and ask her to listen. When you call out "SQUARE" or "OPENTHEDOOR", she should follow the right set of instructions.

I expect you can think of other sets of instructions, which could be called in this way. ITS-TIME-TO-GET-UP or RUN!-THE-BUS-IS-COMING or COULD-YOU-GO-DOWN-TO-THE-SHOP-FOR-ME or CAN'T-YOU-BLOW-YOUR-NOSE.

In LOGO, you give lists of instructions to the computer, and each list of instructions has its own name. Let's see how it works. Go back to the keyboard and type TO SQUARE, followed by the RETURN key. The computer will then give you a new prompt. Instead of the ? at the beginning of the line, you will see a > . This tells you that the computer is waiting for its first instruction. You type

```
> REPEAT 4 [FD 200 RT 90]
```
just as you did before. Press RETURN and type

```
> END
```

Press RETURN and the computer will respond

```
SQUARE DEFINED
```

Type SQUARE again, and the computer will draw a square. Move the Turtle to another part of the screen, with

```
PU RT 45 FD 200 PD
```

and type SQUARE again. The Turtle has remembered exactly what to do. It is just like your friend with the sheet of paper and your instructions. In fact it's better. Unlike your friend, the computer never gets tired of daft games. You call SQUARE and it knows which instructions it has to follow. They are contained in a list, very similar to the one you wrote out for your friend.

We call this a **procedure**. Teaching new procedures to your computer is what Logo is all about. Some people look forward to a world in which computers teach children.

At Logotron, we are more interested in children teaching computers. In fact, the idea of computers teaching children is quite horrible. Here are some more procedures to teach your computer:

```
TO ARC
REPEAT 10 [FD 10 RT 60]
END

TO BEND
REPEAT 2 [FD 50 RT 30]
END

TO ZIGZAG
REPEAT 20 [FD 10 RT 90 FD 10 LT 90]
END

TO SUN
REPEAT ?? [FD 100 BK 100 RT 5]
END
```

When you copy in the last procedure, you need to put a number instead of the question marks. Some people say at this point: "But when do we start programming?" The answer is that these are programs.

You will see that every procedure begins with TO, and ends with END. When you want your computer to draw a sun, you just type SUN.

Make sure that each line is just right before you press the RETURN key. If you do make a mistake, and the procedure doesn't work, just type ER and then the name of the procedure. For example, ER "SUN. ER is short for erase, and makes the computer forget the list of instructions associated with the name of the procedure. Remember the quotes (") in front of the name. You will learn in **Section 6** why they are needed.

Once the procedure is forgotten, you can type it in again, without any mistakes.

Perhaps you can think up some procedures of your own. Anyway, type in the four definitions given above. You will need them in **Section 7.** Once they are working, type:

```
SAVE "XAMPLES [ARC BEND ZIGZAG SUN]
```
As you will realise, this **saves** a **list** of procedures onto your disk or cassette in a file called XAMPLES. The BBC Micro's disk filing system forces us to choose filenames with fewer than seven letters, which is why I had to chop the first "e" off examples. When you return to the computer and want to use these procedures again, put the right disk in the disk drive (or cassette in the recorder) and type:

```
LOAD "XAMPLES
```
and they will be right there, ready for use.

There is one small but important point about procedures. Their names must consist of a single word. You could not have TO ZIG ZAG, for example. You must run the words together to make ZIGZAG. Sometimes people put a full stop between the words; in this case, it would be ZIG.ZAG. Logo reads it as one word.

SECTION SIX – CHANGING LOGO'S MIND

Those procedures the Turtle has learned are fine. But have you wondered what you would do if you wanted a bigger square, or a smaller circle? In the last section you learned that you sometimes have to put quotes (") in front of a word. In this section, you will meet another way of writing words, with dots (:) in front of them. Before very long we will explain what they mean, but for the moment, just type them in without worrying about their meaning. Just type in:

```
TO NEWSQUARE :SIDE
REPEAT 4 [FORWARD :SIDE RIGHT 90]
END
```

Then try

```
NEWSQUARE 100
NEWSQUARE 200
```

This is a very neat idea. The word :SIDE stands for the number of steps you want the Turtle to take for each side of the square. Instead of going FORWARD 100 it goes FORWARD :SIDE and :SIDE can stand for any number you please. Try it and see for yourself. Then try to write a set of instructions for:

```
TO NEWTRIANGLE :SIDE
```

To understand just what is happening, you need to learn a new Logo word, MAKE. Logo uses MAKE to give **names** to **things**.

Type in:

```
MAKE "SIDE 150
```

In this case, the NAME is "SIDE and the THING is 150. Now type

```
PRINT :SIDE
```

the computer will respond, 150. Now try

```
MAKE "SIDE –250
```

and then again

```
PRINT :SIDE
```

The answer, as you might expect, is –250. You don't have to call it "SIDE. "LENGTH or "FRED would do just as well. NAMES can have all kinds of THINGS attached to them, not

just numbers. Try this:

    MAKE "FAMILY [GRANDFATHER GRANDMOTHER MOTHER FATHER AUNT
    UNCLE SON DAUGHTER]

And then

    PRINT :FAMILY

I am sure you will have noticed that sometimes we write "SIDE and sometimes :SIDE, sometimes "FAMILY and sometimes :FAMILY. This can be quite confusing. But when you are referring to the NAME you use quotes ("), and when you are referring to the THING, which is attached to the name, you use dots (:). You may remember in the previous section, when you wanted the computer to forget a bad procedure, you typed in ER "SUN. You had to use the quotes (") because you were referring to the name of the procedure, not to the procedure itself. This can be quite a difficult idea. Think about the difference between you and your name.

Seymour Papert, the inventor of Logo, uses an old riddle to explain the difference:

"Mississippi is the longest river in America, how do you spell it?"

"M-I-S-S-I-S-S-I-P-P-I"
"No I-T"

The riddle works in English because the word "it" could be standing for itself, a two-letter word, or it could be standing for the word Mississippi. In Logo, there is no ambiguity.

If Logo saw "IT, it would know you were referring to the word "it". If it saw :IT, it would ask itself what other word it could be referring to and the answer would be Mississippi.

This will become quite easy with practice. When we say :SIDE we are referring to the number of steps we want the Turtle to take on each side of the square. When we write "SIDE, we mean the word SIDE, not its value. So if you type PRINT "SIDE, the computer will answer SIDE.

Practise this as much as you like until you really understand how it works. There are really three possibilities that Logo has to consider when it meets a word which is not enclosed in square brackets (ie not part of a list).

a) It has neither quotes (") nor dots (:) in front of it, and it's not a number or a logical value (TRUE or FALSE). In this case, Logo will treat the word as a procedure or a primitive procedure. If it cannot find the word in its list of procedures, Logo complains:
I DON'T KNOW HOW TO . . . .

b) It has dots (:) in front of it. Logo tries to evaluate it, to discover what it stands for. In the case of :SIDE, this was the length of a line. We'll meet lots more examples of different kinds.

c) It has quotes (") in front of it. Logo will use it as it is, not as a procedure, and not to be evaluated.

Here are some more examples. Type them in to the computer and see what happens.

```
MAKE "SQUARE [REPEAT 4 [FD 100 RT 90]

RUN :SQUARE

MAKE "HEIGHT 30

MAKE "LENGTH 200

MAKE "AREA :HEIGHT ∗ :LENGTH

PRINT :AREA
```

We will be seeing a good deal more of dots (:) and quotes ("), so be sure you get the hang of them. If you want more informnation on this subject, look up **Section 23** dealing with **variables**. We call these words, which name things, variables, because you can make them hold all kinds of different (variable) things.

CAPITAL LETTERS are sometimes called upper case letters, and small letters are called lower case letters. Logo expects **upper case letters** for **primitives, procedure names, variable names, and boolean values**, but you can use upper or lower case in other cases. For example:

```
MAKE "XNAME "John

PRINT :XNAME
John

MAKE "VERBS [enter eat gallop cry]

PRINT :VERBS
enter eat gallop cry
```

SECTION SEVEN – MAKING MORE CHANGES

In the last section, it would have been nice to have been able to change the list of instructions associated with the word SQUARE. We wanted to improve it by adding in the variable word :SIDE. Since we didn't know how to change it, we invented a new procedure TO NEWSQUARE.

We are now going to learn how to change procedures after they have been defined. If there are no procedures in your computer's memory at present, type in the procedures SQUARE and TRIANGLE.

```
TO SQUARE
REPEAT 4 [FD 200 RT 90]
END

TO TRIANGLE
REPEAT 3 [FD 200 RT 120]
END
```

Now type EDIT "SQUARE At the bottom of the screen, you will see:

LOGO EDITOR

^C <exit>   ESC <abort>

and the procedure SQUARE, all ready to be changed, at the top of the screen.

Before we look at the EDITOR and all the things it can do, I would like to tell you a little about what is happening inside your computer.

When you switch it on, Logo takes charge. You read the message:

```
WELCOME TO LOGO
?
```

The question mark (?) asks you to type in your next instruction. The Logo program looks after the computer's memory, and stores the words you teach it. When you type in words it cannot understand, it complains.

Part of the computer's memory is kept free for new procedures you write. This is called your **workspace**. But once a procedure is written into that **workspace,** you cannot easily change it. It would confuse the computer if you could.

In order to change a procedure, you have to move it into a special part of the memory, called the EDITOR. It's rather like taking the car to the garage to have it mended.

In the LOGO EDITOR, we provide you with your own set of tools for fixing or changing your procedures. When you are in the EDITOR, ordinary LOGO commands do not work. All you can do is change the words you have written. But as we will see, that is very, very useful.

Using the right arrow key (→), you can move the cursor to the end of the first line and type :SIDE. Then drop down to the next line, with the down arrow key, and use the DELETE key to rub out the 200. Type in :SIDE instead.

You now have:

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

When you are satisfied you have it right, press CTRL C. The screen will go blank, and the computer will respond.

```
SQUARE DEFINED
```

You are now out of the editor, with a new list of instructions for SQUARE fixed in the computer's memory. Try SQUARE. Logo responds:

```
NOT ENOUGH INPUTS TO SQUARE
```

SQUARE is now a command which needs an input, the length of each side. Try SQUARE 100. Let's look at all the things we have learnt so far:

1. Some Logo words: FORWARD (FD), BACK (BK), RIGHT (RT), LEFT (LT), PU (pen up), PD (pen down), PE (pen erase) HT (hide turtle), ST (showturtle), REPEAT, MAKE, EDIT (ED), ER (erase), CS (clear screen), CLEAN, WINDOW, WRAP, FENCE, PRINT (PR), TO, END.

2. How to reach the Turtle new procedures like SQUARE, TRIANGLE and ZIGZAG.

3. How to make pictures of different :SIZE or :LENGTH, by attaching a **name** to a **thing**.

4. How to change **procedures** in the LOGO EDITOR. You will find a special list of all the EDITOR tools in Section

25. Don't try to learn them all at once. Just use the ones you need, the arrow keys and DELETE.

5. The way Logo uses square brackets [ ] to enclose **lists**, quotes (") to indicate **names**, and dots (:) to indicate **named things**.

You really know a good deal about LOGO now. The important thing is to feel comfortable with the ideas we have met so far. This may take a different amount of time for different people. But take your time. If you understand these ideas, you will get on well.

If you are still puzzled about any of it, put the manual away, and look at it again tomorrow, or even next week. Then work through the first sections again. They may well seem easier. Or find someone else who wants to talk about Logo. Two heads are usually better than one when it comes to talking turtle and teaching computers.

SECTION EIGHT – MAKING PICTURES

In the first part of this manual, we have been learning how to use Logo, with very simple shapes, like SQUARE, CIRCLE and TRIANGLE.

The procedures which tell the Turtle to draw these shapes are quite short and simple.

We are now going to look at some different shapes, and you may want to save these on a disk or cassette, so that you don't have to type them in each time.

This means two more Logo words SAVE and LOAD. We will assume SQUARE :SIDE is already in the computer's memory. If it isn't, go back to the last section and type it in. Make sure it works. If it doesn't, EDIT "SQUARE and get it just right. Now type:

    SAVE "SHAPE "SQUARE

There will then be a gentle whirring sound as the computer saves the procedure SQUARE on a cassette or a disk, under the **filename** SHAPE. The filename comes first. Remember, the BBC disk filing system does not allow you to use filenames with more than 7 letters. If you called your file SAMANTHA for example, you would receive the Logo message, BAD FILE NAME.

When you have saved SQUARE. Type ER "SQUARE. You know that this makes the computer forget the procedure SQUARE. Check that it really has forgotten by typing SQUARE 100. The computer answers

    I DON'T KNOW HOW TO SQUARE

Now type LOAD "SHAPE. This time, you use the filename alone. There is more gentle whirring, and when the ? reappears, type SQUARE 100. The Turtle does its stuff and draws a square. That's all there is to SAVE and LOAD.

Now let's go straight into the EDITOR by typing EDIT [ ]. You are in the LOGO EDITOR. it says so at the bottom of the screen. But there is no procedure for you to work on. You can just type one in. If you had typed EDIT alone, without the square brackets, you would have found the last procedure edited still on the screen.

Many people prefer to build their procedures inside the

LOGO EDITOR. Then they can change them around, and correct typing mistakes, without worrying about the RETURN key fixing a line in the computer's memory. When you are typing inside LOGO EDITOR, nothing is fixed until you press CTRL C. If you don't like the changes you have made, just press the ESCAPE key, and you are out of the EDITOR without making any changes to the procedures in your workspace.

So let's try building a procedure inside the LOGO EDITOR. It is a very famous one, familiar to readers of Seymour Papert's book Mindstorms. Here it is:

```
TO HOUSE :SIDE
SQUARE :SIDE
TRIANGLE :SIDE
END
```

Then press CTRL C and wait for the computer to say HOUSE DEFINED. Now this is quite different from the other procedures we have seen. If you now type HOUSE 100, the Turtle first draws a SQUARE, with 100 steps to each :SIDE, and then a TRIANGLE, also with 100 steps to each :SIDE.

Of course, if the procedures SQUARE and TRIANGLE are not in the computer's memory, it will complain:

```
I DON'T KNOW HOW TO SQUARE
```

or

```
I DONT KNOW HOW TO TRIANGLE
```

Silly beast. Can't it remember anything? Well, it can remember, so long as it hasn't been told to forget (ER) or it hasn't been switched off. If any of these things have happened, LOAD the file containing "SQUARE and/or "TRIANGLE from your disk or cassette. And try HOUSE 200 again.

My idea was that the square would be the bottom part of the HOUSE and the triangle would be the roof. It hasn't quite worked out. Never mind, we can fix it in the LOGO EDITOR, the garage for broken down procedures. Type EDIT or even ED. You should now be in the LOGO EDITOR, with the definition of HOUSE just as you left it.

Before we change it, let's think what happened. Get a pencil and paper, and imagine the pencil is the Turtle.

You draw a line running up the page, 200 turtle steps. Turn right, and another line, also 200 steps. Another line down the page, and another line back to where you started. That's SQUARE taken care of. Now for TRIANGLE.

You go up the page again (200 steps), retracing your first line, then turn 120 degrees to the right (that's more than a right angle), and draw another line; RIGHT 120 again, and back to where you started.

Now, if you could start the triangle in the top left-hand corner of the square, instead of the bottom left-hand corner, it might be more like a roof. So let's use the arrow key to go to the end of the line SQUARE :SIDE. Press the RETURN key and you will be ready to type FD :SIDE. The procedure should now look like this:

```
TO HOUSE :SIDE
SQUARE :SIDE
FD :SIDE
TRIANGLE :SIDE
END
```

Press CTRL C and try it again. HOUSE 200. It's still not very good. The triangle isn't sitting on the house as a good roof should. Let's EDIT "HOUSE again. This time, we will tell the Turtle to change direction before drawing the triangle.

I am not going to tell you how much. You see if you can discover for yourself what number to type in where I have left two question marks ??.

```
TO HOUSE :SIDE
SQUARE :SIDE
FD :SIDE
RT ??
TRIANGLE :SIDE
END
```

NB. HOUSE won't work unless you do type in a number instead of the ?? question marks.

Now press CTRL C to leave the EDITOR and try HOUSE 200. Well, I am sure you could draw a better house with a pencil and paper, and very soon you will teach the Turtle to make a better effort, with procedures called DOOR, WINDOW, CHIMNEY, and SMOKE.

If you think you could do that already, then by all means try. That might give you ideas for bigger houses, factories or churches. But why buildings? Try drawing faces, (HEAD, NOSE, MOUTH, RT.EYE LT.EYE) or aeroplanes and rockets (WING FIN NOSE TAIL BODY).

Play with gluing shapes together in a single procedure, which combines other procedures. You will see that you often have to put in extra commands, as we did with HOUSE, to get the effect you want.

SQUARE, CIRCLE, TRIANGLE, ZIGZAG and SUN should give you plenty of ideas, but we will give you some more powerful tools in the next sections, to make more exciting shapes. This might be a good time to look at the other ways you can use the EDITOR, in **Section 25.**

The best way to make pictures is to keep changing your procedures, see how they work, and then look for little ways in which you can make them better. Its sometimes worth keeping the old one, and giving the new version a slightly different name. For example, you might decide that you had a better way of drawing a sun:

```
TO SUN1 :RAY
REPEAT 72 [FORWARD :RAY LEFT 5 BACK :RAY RIGHT 10]
END
```

Can you now think of a way to turn TO SUN1 into TO SUNFLOWER with TO STALK, TO LEAF, TO SEEDS and so on?

SECTION NINE – TURTLE ARITHMETIC

Most people know that computers are very handy for doing arithmetic. If you have ever seen a pocket calculator, you will understand this very easily. A pocket calculator is a small computer, specially designed to do arithmetic very quickly.

Try the Turtle out on some easy sums. Type PRINT 3 + 4. Quick as a flash, you get the answer 7. Let's see what would have happened if you had simply typed 3 + 4. Try it. The computer will respond:

YOU DON'T SAY WHAT TO DO WITH 7

This is very important. In LOGO you are the boss. The Turtle doesn't do anything or know anything, unless you tell it or teach it. If you type PRINT in front of a calculation, the computer prints it onto the screen. If you had a printer it would print the result onto the paper. But you don't always want the result printed on the screen. Often you want the calculations to be used inside a procedure.

If you have never used a computer before, you may be surprised to learn that they don't use quite the same symbols for multiplication and division, as those you learned at school. Plus (+) and minus (–) are just the same, but instead of × for multiply, computers generally use ∗, and for divide, they use /. So:

3 ∗ 4 = 12

2.5 ∗ 2 = 5

6 / 3 = 2

12.5 / 2.5 = 5

It takes a bit of time to get accustomed to the new symbols when you first meet them. But, just make sure they work for you by typing PRINT 48 / 8. Or PRINT 9 ∗ 7. Or any other sum you can think of. You can, of course, use decimals, and type

PRINT 345.9876 ∗ 200.0001234

Let's look at how we might use arithmetic to buid more interesting procedures. Type

MAKE "SIDE 100

then, when you see the ? again, type

PRINT :SIDE / 2

The computer sees the dots : and knows you are talking

about the value associated with the name "SIDE, and so divides 100 by 2, and gives you the answer 50. Let's use this idea to write a new procedure called BOX. You can build it inside the EDITOR, typing EDIT first, or just type it straight into the memory.

```
TO BOX :SIDE
FD :SIDE / 2
RT 90
FD :SIDE
RT 90
FD :SIDE / 2
RT 90
FD :SIDE
END
```

Try it out, with BOX 200. Can you think of a way of shortening that procedure using the REPEAT command?

You could then improve BOX with another procedure called LID.

```
TO LID :SIDE
REPEAT 2 [FD :SIDE / 6 RT 90 FD :SIDE RT 90]
END
```

You then modify BOX in the EDITOR to read:

```
TO BOX :SIDE
FD :SIDE / 2
LID :SIDE
RT 90
FD :SIDE
RT 90
FD :SIDE / 2
RT 90
FD :SIDE
END
```

Then type BOX 300. The next project might be to draw a group of BOXES. The procedure might look like this:

```
TO BOXES :SIDE
BOX :SIDE
MOVE :SIDE
BOX :SIDE * 2
MOVE :SIDE
BOX SIDE / 2
END
```

You have to work out how to write the procedure MOVE. The clue is to use PU when you are moving the Turtle without leaving a trail. Remember to SAVE any procedures you would like to keep on to a disk or cassette before switching off the computer.

If you have several procedures to save you can put them in a list, which saves time. For example you could type:

SAVE "SHAPES [BOX BOXES SQUARE HOUSE TRIANGLE]

This would save the listed procedures under the **filename** SHAPES. LOAD "SHAPES and they would all be loaded from disk, back into the workspace. if you had typed just SAVE "SHAPES, without any names of procedures, Logo would have saved everything into the workspace in the file named SHAPES.

Logo's ability to do arithmetic can be useful in many ways. Perhaps you have a lot of sums. This procedure might help:

```
TO CALCULATE :SUM
PRINT :SUM
END

CALCULATE [12 * 4]

CALCULATE [3 + 3 + 17 + 19]

CALCULATE [12 / 16 * 100]
CALCULATE [(3 + 4) * (9 − 2)]
```

There are better ways of using Logo as a calculator, but that will do for the present (see **RUN** in **Section 26**).

SECTION TEN – TURTLES EAT TURTLES

If you have any procedures in the memory of your computer, SAVE them and then empty its memory. You do this by typing ERALL. It means erase all, or "forget everything I ever told you." Then try this little procedure:

```
TO CIRCLE
FD 10 RT 5
CIRCLE
END
```

Now try it, by typing CIRCLE. You will find the Turtle goes round and round for ever. Forward a little bit, right a little bit, and then CIRCLE again. Forward a little bit, right a little bit. You can stop it by pressing the ESCAPE key.

```
STOPPED!!! IN CIRCLE
```

This may not look very useful at first sight, a procedure which never stops. It is also quite confusing. Like standing between two mirrors and looking into one of them and seeing a reflection of a reflection of a reflection . . . and so on (for ever?).

Procedures which call themselves in this way are **recursive procedures**. Here are some more to play with. Remember, the only way to stop them, is to press the ESCAPE key. Otherwise, they go on for ever.

```
TO SQUARE :SIDE
FD :SIDE RT 90
SQUARE :SIDE
END
```

```
TO TALLY :N
PRINT :N
TALLY :N + 1
END
```

NB to use TALLY, you must type TALLY with a number for :N. TALLY 1, for example.

```
TO SQUIGGLE
FD RANDOM 50 RT RANDOM 360
SQUIGGLE
END
```

RANDOM is another Logo word. It chooses any number between 0 and the number following RANDOM. So RANDOM 5, chooses 0, 1, 2, 3, or 4.

It's easy to see that we need to find some way of controlling these **recursive procedures**. Turtles should never be running out of control.

Let's look again at our recursive procedure for drawing a square. Type EDIT "SQUARE

```
TO SQUARE :SIDE
FD :SIDE RT 90
SQUARE :SIDE
END
```

Change this procedure as follows:

```
TO SQUARE :SIDE :BRAKE
PR SE [BRAKE = ] :BRAKE
IF :BRAKE = 0 [STOP]
FD :SIDE RT 90
SQUARE :SIDE :BRAKE − 1
END
```

Press CTRL C and try out SQUARE 300 4.

NOTE: SQUARE 200 5 might have come in rather handy when we were drawing our HOUSE.

Now let's look at another procedure, very similar to SQUARE:

```
TO SPIRAL :SIDE
FD :SIDE RT 90
SPIRAL :SIDE − 5
END
```

Before you try it out on the computer, why not try walking it on the floor or the lawn. Every :SIDE is a little bit shorter than the one before. So you never quite complete the square. That's why it's called SPIRAL.

Try SPIRAL 300. The procedure is still never-ending, but perhaps you can see how recursion might come in handy. Now edit SPIRAL to read as follows:

```
TO SPIRAL :SIDE
IF :SIDE < 10 [STOP]
FD :SIDE RT 90
SPIRAL :SIDE − 5
HT
END
```

This time, if you try SPIRAL 300 again, the procedure comes to an end. The key lies in the second line. Every time the

procedure calls SPIRAL, the value of :SIDE is reduced by 5. The second line tells the computer that IF that value is less than 10, it should look for its next instruction in the square brackets [ ], and there it is told to STOP.

There are several new Logo words and ideas here, especially if this is your first time on a computer. Let's start with the easiest. STOP means just what you would expect. It tells the computer to STOP whatever it is doing and get on with the next procedure, if there is one.

IF is a useful word in most computer languages.

IF suchandsuch **THEN** do thisandthat.

In Logo, you don't have to write THEN, you just put the instructions (do thisandthat) in a list, enclosed in square brackets [ ], just as you would for REPEAT.

You know what an equals = sign looks like, but you may not have met the signs meaning greater than > or less than <.

Let's practise using them for a minute. Type:

    If 3 < 4 [PRINT [THREE IS LESS THAN FOUR]]

Note the second set of square brackets, one nested inside the other [[]]. The first, or outer, set are controlled by the IF, while the second, or inner, set are controlled by PRINT. You can have any number of brackets nested inside one another [[[[ ]]]]. What you must avoid is [[]]][, for example:

    IF 4 > 3 [PRINT] FOUR IS GREATER THAN THREE]]

Type MAKE "SIDE 100. Type:

    IF :SIDE = 100 [PRINT "OKAY]

Now make up some more of your own. If you want further details, look in the reference sections of this manual. One thing to notice. If you want the computer to print out a single word, like okay, you use the quotes " symbol, as in:

    PRINT "OKAY

If you want it to print out more than one word, put them into a list, inside square brackets:

    PRINT [HOW ARE YOU TODAY]

There are other ways of stopping a recursive procedure, without using the word STOP. Look at SPIRAL again, and

change it once more to:

```
TO SPIRAL :SIDE
FD :SIDE RT 90
IF :SIDE > 10 [SPIRAL :SIDE – 5]
HT
END
```

Here the brakes are in the third line, but we have turned the idea round. If the value of :SIDE is greater than 10, then we go ahead and call SPIRAL again. IF NOT we move on to Hide the Turtle and END the procedure.

Another method, where you know how many times you want to call the procedure is to introduce a second input, which we could call :COUNTER, or any other name that takes your fancy, like :METER or :NUMBER or just :N. The important thing is to give it a name which means something to you. SPIRAL would then appear as follows:

```
TO SPIRAL :SIDE :COUNTER
FD :SIDE RT 90
IF :COUNTER > 0 [SPIRAL :SIDE – 5 :COUNTER – 1]
HT
END
```

When the :COUNTER reaches 0, the procedure ENDS. Try it by typing SPIRAL 300 12. You can, of course, change SPIRAL by making :SIDE increase on each call. You would then start with a small value in :SIDE and change your stopping mechanism to put the brakes on before the value of :SIDE outgrows the screen.

A good example of recursion is the old story of the fairy who offered a child two wishes, anything she liked to choose. I don't know what her first wish was, but her second was to be granted just two wishes, anything she liked to choose . . . .

SECTION ELEVEN – TURTLE COLOURS

That last section was fairly tough. We looked at some recursive procedures, and ways of stopping them from going on and on and on and on . . . .

We haven't said anything about Colour. Well, some people don't have colour. They are stuck with a single colour. But if you are lucky enough to have a colour monitor or a colour TV for your computer, you should certainly experiment with two more Logo words.

By the way, the words Logo already knows when you turn on your machine are known as **primitives**. This is short for primitive procedures. If this seems a funny word, the Shorter Oxford English Dictionary tells us that a "primitive word" is a root word, from which another or others are derived. The Latin word primitivus meant the first or earliest of its kind. If you type in PRIMITIVES, Logo responds by printing out a list of all its primitive procedures.

Before you start experimenting with colour, it is worth switching your computer into the MODE which has the most colour possibilities. You do this in the following way. First SAVE any procedures you might want again.

The simplest way to do this is to save the whole workspace, by typing: SAVE "WORK8. Logo will create a file (on disk or cassette) called WORK8. The file can have any name you like. If you are sharing the disk with other people, your own name might be a good filename. SAVE "ANNA9. The numbers are not necessary, but they can be helpful if you want to keep a series of files with similar names. Remember, the BBC puts a limit of 7 letters on filenames. "WEDNESDAY'S.WORK would be much too long, and you would receive a Logo message:

    BAD FILE NAME
Having saved your work, type ERALL. This makes Logo forget all the procedures in its workspace. Then type:

    SETMODE 2
This takes you into the BBC Micro's MODE 2. The primitives which handle colour are SETPC :N and SETBG :N. In each case :N is a number.

If you are like me, and find it difficult to remember numbers,

you can make your own procedures as follows:

```
TO REDPEN
SETPC 1
END

TO BLUEPEN
SETPC 4
END

TO REDPAPER
SETBG 1
END

TO YELLOWPAPER
SETBG 3
END
```

Try them out, to see how they work. You can also write procedures, which help you remember the colours and the numbers. Try this:

```
TO COLOURS
CS
SETBG 0
REPEAT 15 [SETBG BG + 1 SETCURSOR [10 12] PR SE [BG =] BG
WAIT 60]
END
```

Don't worry about how it works, just type it in and then type: COLOURS and see what happens. While we are playing with colours, let's look at another primitive, .SETNIB, which allows you to do a number of interesting things. Try .SETNIB 85 SQUARE or .SETNIB 21 FD 300. For more about .SETNIB, check the reference in Section 21. **Note** the dot in front of .SETNIB.

If you are familiar with the BBC machine, but not with Logo, you may have learnt about the VDU commands. If you have, you will be pleased to find that you can use VDU as a Logo primitive. As you might expect, VDU looks for its numbers in the form of a list enclosed in square brackets [ ].

If you haven't heard of VDU commands, don't worry about them. When you feel ready to learn about them, either ask someone, or read the BBC User Guide. The important thing to remember is that they are available from Logo, just like all the other powerful features of your BBC Computer.

As you probably know, the BBC Computer can be in any of 8 MODES .When you enter Logo, you are in MODE 4. Type PRINT MODE and the computer will reply 4.

The command SETMODE :N, where :N is a number from 0-7 will put you in a new mode. In MODE 2, you can have up to 16 different colour combinations on the screen at the same time.

0 = BLACK
1 = RED
2 = GREEN
3 = YELLOW
4 = BLUE
5 = MAGENTA
6 = CYAN
7 = WHITE
8 = FLASHING WHITE-BLACK
9 = FLASHING RED-CYAN
10 = FLASHING GREEN-MAGENTA
11 = FLASHING YELLOW-BLUE
12 = FLASHING BLUE-YELLOW
13 = FLASHING MAGENTA-GREEN
14 = FLASHING CYAN-RED
15 = FLASHING BLACK-WHITE

In MODE 7, too, you can use all the colour combinations, but these are obtained in a different way. See the section dealing with VDU commands for a full explanation, or plunge into the BBC User Guide, but you need a fairly strong stomach to sort out the information you want.

In MODES 1 or 5, you are restricted to 4 colours, but you can choose which four you want. The default colours, set when you turn on your computer, are

0 = BLACK
1 = RED
2 = YELLOW
3 = WHITE

Your choice is made through a VDU command. If you want to know all the details, look at page 382 in your BBC User

Guide. But here is a useful procedure:

```
TO SETPAL :A :B
MAKE "A (SE 19 :A :B 0 0 0)
VDU :A
END
```

The first input (:A) is the number of the colour you want to replace, and the second (:B) is the number of the colour you want to introduce. For example SETPAL 1 4, in MODE 5, would replace red with blue. The best way to experiment with colour is to go back over the earlier material, look at your procedures again, and see how they might work, in colour.

In MODES 0, 3 , 4 and 6, you have only two colours. These are set as

0 = BLACK
1 = WHITE

when you switch on your computer, but they, too, can be changed through application of the right combinations of VDU commands (Use SETPAL if you like). The VDU driver system is quite foreign to the Logo philosophy. We have to put up with it because it is part of the BBC computer hardware.

WARNING: The different MODES use up different amounts of your computer memory. You'll find more about this in the Reference Manual. But here are some helpful hints.

You can always discover how much memory you have for storing procedures by typing PRINT NODES. Logo organises its memory in NODES, each of which is equivalent to 5 bytes. But you needn't worry about that.

The most nodes are available in MODE 7, the least number in MODES 0, 1 and 2. There are no Turtle graphics in MODE 7, but it can be very useful for some other applications.

You can discover for yourself, by changing modes with SETMODE :N and then typing PR NODES. If you change modes, with a procedure in the computer's memory, you will get a different result from changing modes without a procedure in the memory. Check this out for yourself.

So if you want to get extra memory, changing, for example,

from MODE 2 to MODE 5, or from MODE 3 to MODE 7, be
sure to SAVE your procedures, and type ERALL before
changing modes with SETMODE :N. It sounds complicated,
but if you play around with it for a little, it will seem much
clearer. You will find more information on all this in the
reference part of this manual, in **Section 29.**
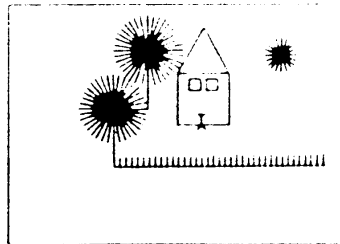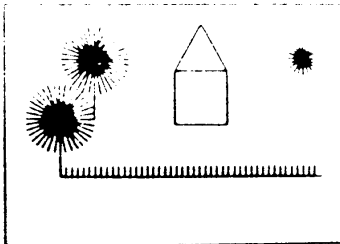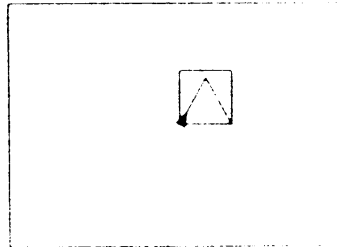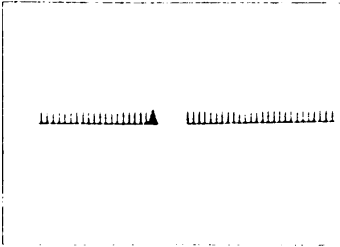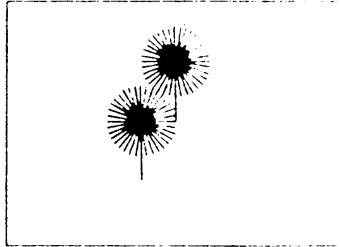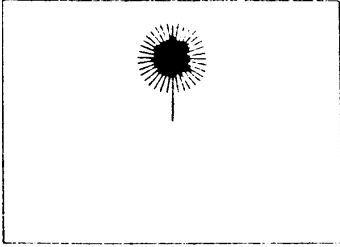
SECTION TWELVE – MORE ABOUT PICTURES

In the first sections of this manual, we have concentrated on very simple shapes. I wanted the younger children to be able to follow this part of the book , and assumed the older ones would gallop through it in a few hours. The next few sections are aimed at older readers, or anyone who feels completely comfortable with the ideas we have introduced so far.

This might be a good moment to read the first sections of the reference manual, especially the introductory section on Logo Grammar, and the sections on Turtle Graphics and the Editor.

I would then like you to look at one of the most famous turtle graphics procedures:

```
TO POLYTRIP :SIDE :ANGLE :ANGLES
FD :SIDE RT :ANGLE
MAKE "ANGLES :ANGLES + :ANGLE
PR SE [:ANGLES =] :ANGLES
PR [DO YOU WANT TO ADD ANOTHER SIDE? Y/N]
MAKE "ANSWER RC
IF :ANSWER = "Y [POLYTRIP :SIDE :ANGLE :ANGLES]
HT
END
```

This procedure introduces several new LOGO words and ideas. You can either look up SE and RC in the reference manual, or just accept them on trust for the moment. The idea of this procedure is to learn something important about polygons and turtle graphics.

Now type

```
POLYTRIP 200 90 0
```

if you type in Y on the first three occasions you are asked whether you want to add another side, you will find you have completed a SQUARE, and :ANGLES, which is the sum of all the Turtle's turns, will equal 360.

CLEAN the screen and try again with:

```
POLYTRIP 100 72 0
```

This time, you will need to add four sides, before completing a polygon, but the sum of the Turtle's turns will still be 360.

Now look for other regular Polygons, changing the first two inputs. Keep the third one always a zero. Some of the angles you choose won't produce a regular polygon. Instead of returning to your starting point, the lines will cross. But if you do return to your starting point without crossing a line, you will find that :ANGLES always equals 360. Some POLYTRIPS,

```
POLYTRIP 100 144 0
```

for example, will take you back to your starting point, but cross several lines on the way. :ANGLES equals 720, which is exactly twice 360.

Playing with POLYTRIP will help you understand the next procedure:

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

This procedure allows you to draw a triangle, a square, a pentagon, a hexagon, an octagon, or any other regular convex polygon. You just tell it the :NUMber of SIDES you want, and the computer caluclates the angle the turtle has to turn at each corner. It just divides 360 by :NUMSIDES.

Another way of writing the same procedure would be:

```
TO POLY1 :SIDE :ANGLE
REPEAT 360 / :ANGLE [FD:SIDE RT :ANGLE]
END
```

One method isn't "better" than another, just different. I prefer the first method, when I know that I want to draw a hexagon or an octagon, because I can never remember what the turning angle should be. On the other hand, if I wanted to explore the effect of different turning angles, POLY1 would be better.

Do you remember how we developed SPIRAL out of SQUARE? Well, you can do just the same with POLY, and it's often called POLYSPI:

```
TO POLYSPI :SIDE :ANGLE
FD :SIDE RT :ANGLE
POLYSPI :SIDE + 1 :ANGLE
END
```

We will leave you to play with POLYSPI. Remember, you need to put in a brake line, to halt the procedure

(see **Section 10** if you have forgotten how that is done). You can make :SIDE grow or shrink. You can make :ANGLE grow or shrink if you like. There's plenty of room for experiments. We will leave you to explore POLYSPI. Other weird and wonderful shapes can be generated by taking a simple shape, and then rotating it and repeating it. Look at this for example.

```
TO SQUARES1 :SIDE
REPEAT 36 [SQUARE :SIDE RT 10]
END
```

You want to make it more complicated?

```
TO SQUARES2 :SIDE
REPEAT 36 [SQUARE :SIDE RT 5 SQUARE :SIDE / 2 RT 5]
END
```

Or change the definition of SQUARE to:

```
TO SQUARE1 :SIDE
FD SIDE
CIRCLE
RT 90
REPEAT 3 [FD :SIDE RT 90]
END
```

Then try SQUARES1 again.

Add some colour with

```
SQUARES3 :SIDE
REPEAT 36 [SETPC PC + 1 :SQUARE :SIDE RT 10]
END
```

I hope you have been collecting useful procedures on a disk or cassette as you have worked through this introduction. A Logo programmer soon acquires a whole library of useful procedures, which can be used over and over again.

SECTION THIRTEEN – MOVING TURTLES

Before leaving turtlegraphics, I would like to look at the ways we have of moving the Turtle around the screen.

Using just 2 numbers, you can always describe the position of the Turtle on the screen. These numbers are called its X and Y coordinates, or XCOR and YCOR. Imagine the X line going from side to side and the Y line going up and down.

This is a very old idea, and you may have met it in maths at school, or reading map references. We are told the idea originated with a Frenchman, Rene Descartes, more than 300 years ago. He had the idea, while lying ill in bed, looking at flies walking around on the ceiling of his room. He saw that with two numbers he could always describe the position of a fly.

The mid-point of the screen is 0 on both lines. Points to the left, on the X line, are negative, with a minus sign, and points to the right are positive, with a plus sign. Points above the mid point are positive on the Y line, and points below are negative.

Type the following

```
PU SETX -200

PD SETY -250

CLEAN

PU SETX 280 SETY 200

PRINT YCOR

PRINT XCOR
```

This should give you a good feeling for moving the Turtle around the screen. You will see that if the PEN is DOWN, the Turtle will draw lines. This might give you an idea for another method of drawing a square.

```
TO SQUARE :SIDE
SETX XCOR + :SIDE
SETY YCOR − :SIDE
SETX XCOR − :SIDE
SETY YCOR + :SIDE
END
```

Try it. I don't much like it myself, but it's very fast, much

faster than the normal Logo way of drawing a square, and gives you a good sense of the meanings of four new Logo words SETX, SETY, XCOR and YCOR.

You can see for yourself that if you know the value of XCOR and YCOR, you know the position of the TURTLE.

The Logo word for this is POS. Type

```
CS FD 100 PRINT POS
```
The Turtle will draw a line 100 steps forward from the centre of the screen, and print below: 0 100.

The XCOR is unchanged at 0, while YCOR is now 100. In printing POS, the XCOR is always given first. Equally, if you want to set the Turtle down in a new position, you give the XCOR first.

The Logo word for this last operation is, as you might guess, SETPOS. SETPOS takes a list of two numbers, as follows: SETPOS [100 -35]. If you don't want the Turtle to draw a line on its way to its new POS, be sure to type PU first. Try:

```
CLEAN PU SETPOS [-250 35]
```
I often use a procedure called MOVETO, which moves the Turtle to a determined point on the screen without leaving a trail.

```
TO MOVETO :X :Y
PU
SETX :X
SETY :Y
PD
END
```
Another way of writing MOVETO:

```
TO MOVETO :X :Y
PU SETPOS SE :X :Y PD
END
```
SETPOS needs the values of :X and :Y in a list, and that's what SE does. If you wrote SETPOS [:X :Y], Logo would complain:

```
SETPOS DOESN'T LIKE [:X :Y] AS INPUT
```
Once something is in a list, LOGO treats it quite literally and doesn't look for its value. This makes SE a valuable word.

MOVETO can be used, for example, to sprinkle stars across the screen.

```
TO STAR
POLY 30 144 0
END

TO STARS
REPEAT 50 [STAR MOVETO (RANDOM 300 – 300) (RANDOM 300 – 300)]
END
```

While considering the problem of moving the Turtle about the screen, we should look at another pair of Logo words, HEADING and SETH (which is short for SETHEADING).

The Turtle's HEADING is the direction in which it is pointed. If you type FD 100, it will set off in the direction of its HEADING.

Try typing SETH 45 SETH 100 SETH 180. Then type

```
SETH RANDOM 360 PRINT HEADING
```

if you experiment with these last two instructions, you will soon discover that the Turtle measures its HEADING in degrees. 0 is straight up the screen, and 180 is straight down.

I hope you understood what RANDOM is doing. I used it previously in SQUIGGLE. If you have any doubt, look it up in the reference manual. RANDOM is a very useful word as is allows you to break away from straight lines and stiff geometrical shapes. Perhaps you will find uses for this procedure, which can be used instead of FORWARD

```
TO WIGGLE :STEPS
MAKE "H :HEADING
REPEAT :STEPS [SETH (:H + 5 – RANDOM 10)]
SETH :H
END
```

It can make drawings look a lot more natural.

SECTION FOURTEEN – AFTER TURTLE GRAPHICS

Quite a lot of Logo manuals get to this point and then leave the readers to find their own way. They are taught Turtle graphics, which are the easiest part of Logo to understand, and have to pick up the rest from reference manuals.

We are taking a rather different approach. There is still plenty for you to discover on your own about Turtle graphics. You will find a good bit more in the reference manual. Four very good books are:

Apple Logo, By Harold Abelson, BYTE/McGraw Hill. Although this is about Apple Logo, not BBC Logo, the commands are almost identical, and his programs work even better on the BBC Micro than they do on an Apple II. There are two versions of this book, one with a red cover, the other blue. Be sure to get the BLUE one.

LOGO Programming by Peter Ross, Addison Wesley, Small Computer Series. His programs need more changes than Abelson's. But his book is full of good ideas.

Turtle Geometry, by Andy DiSessa and Harold Abelson, MIT Press, is the most complete book of Turtle Graphics, but don't buy it unless you are really keen, and willing to work rather hard. It is most suitable for 'A' level students, university undergraduates and teachers. But it is also a fascinating source of ideas for anyone seriously interested in Logo programming.

Learning with Logo, by Daniel Watt, McGraw-Hill Book Company, is very popular with many teachers.

So we will leave Turtle graphics for the time being, though we include some sample programs later in this manual.

In fact, you can make Turtles or Turtle graphics in any computer language. Before Logo was available on the BBC, there were several turtle graphics programs, which you may have seen.

Lists make Logo special, and different from other languages. We already met some lists in the first part of this manual. In particular, we met lists of instructions, enclosed in square brackets [ ] after words like REPEAT. But there were other

lists. In case you have forgotten, type:

```
MAKE "ALPHABET
    [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]
```

I am choosing this one because it is a list you should know well. Type:

```
PRINT COUNT :ALPHABET
```

Did you get the answer you expected? Try:

```
PRINT ITEM 4 :ALPHABET

PRINT FIRST :ALPHABET

PRINT LAST :ALPHABET

PRINT BF :ALPHABET

PRINT BL :ALPHABET
```

BF and BL stand for But First and But Last, respectively. In each case, make sure you understand what is happening to the list we named "ALPHABET. We could have given it any other name. Type:

```
MAKE "ABC :ALPHABET
```

Now type: PRINT :ABC. So there is nothing to stop you giving a list a new name.

The first thing to do is to make some more lists, and to teach the computer their names. Try the following:

```
MAKE "FAMILY [GRANDPARENT MOTHER FATHER AUNT UNCLE BROTHER
    SISTER CHILD]

PRINT ITEM 3 :FAMILY
```

Did it behave as you expected?

```
MAKE "TREES [ELM OAK ASH HOLLY THORN MAPLE]

MAKE "MUSIC [ROCK DANCE REGGAE POP RADIO2 LATINAMERICAN
    CLASSICAL GUITAR]
```

Try out the words FIRST, ITEM, LAST, COUNT, BF, and BL on these lists, using PRINT statements. When you feel comfortable, try combining members of two lists into a new list, using SE. SE is short for SENTENCE and it creates a list out of two or more inputs. If there are more than two, you have to put SE, with its inputs inside parentheses ( ). For example

```
MAKE "FRUITS (SE "APPPLES "PEARS "ORANGES "LEMONS)
```

Then type

```
SHOW :FRUITS
```

The Logo word SHOW is different from PRINT in that it shows you a list with the square brackets [ ] in place. PRINT strips them off. In order to combine elements from two lists into a third list, try the following:

```
MAKE "NEWLIST SE FIRST :FAMILY LAST :FRUITS
```

```
SHOW :NEWLIST
```

This should show you:

```
[GRANDPARENTS LEMONS]
```

Let's now look at the possibility of adding elements to lists we have already named. Type:

```
MAKE "FRUITS FPUT "PINEAPPLES :FRUITS

PRINT :FRUITS

MAKE "FRUITS LPUT "MANGOS :FRUITS

PRINT :FRUITS

MAKE "FRUITS BUTFIRST :FRUITS

PRINT :FRUITS

MAKE "ALPHABET BUTFIRST BUTFIRST :ALPHABET

PRINT :ALPHABET
```

Now put "A and "B back at the beginning of :ALPHABET. This looks quite simple, but out of these very simple ideas, computer scientists have built incredibly complex programs.

We are going to look first at some very simple things you can do with lists. In another section, we have provided some more advanced procedures, which you can use. But first, let's look at a very simple example given by Harold Abelson in the book mentioned above.

```
TO CHATTER
MAKE "NOUNS [DOGS CATS CHILDREN TIGERS]
MAKE "VERBS [RUN BITE TALK LAUGH]
BABBLE
END

TO BABBLE
PRINT SE PICKRANDOM :NOUNS PICKRANDOM :VERBS
BABBLE
END
```

47

In order to make these procedures work, you need another one called:

```
TO PICKRANDOM :X
OP ITEM 1 + (RANDOM COUNT :X) :X
END
```

Now try CHATTER. It goes on for ever, so you will have to stop it with the ESCAPE key. Harold Abelson provides many ideas for extending the chatter program, so that it stops itself, and learns new words.

The PICKRANDOM procedure may puzzle you, so look at it carefully. The :X is always a list. You can use it on any of the lists we have made in this section. Try:

```
PRINT PICKRANDOM :FRUITS
```

or

```
PRINT PICKRANDOM :ABC
```

So you can see what PICKRANDOM does. The second line may give you more trouble:

```
OP ITEM 1 + (RANDOM COUNT :X) :X
```

This word OP is a new Logo word, standing for OutPut. It simply passes the result of the procedure in which it appears back to the procedure which called it. Demonstrations are always better than explanations.

EDIT "PICKRANDOM so that it reads:

```
TO PICKRANDOM :X
PRINT ITEM 1 + (RANDOM COUNT :X) :X
END
```

Press CTRL C to leave the EDITOR. Now try

```
PRINT PICKRANDOM :FRUITS
```

You will get a Logo message saying:

```
PICKRANDOM DIDN'T OUTPUT TO PRINT
```

Then try

```
PICKRANDOM :FRUITS
```

which should work perfectly.

The problem the first time was that the PRINT command was expecting PICKRANDOM to tell it what to PRINT. Your new PICKRANDOM is fine if all you ever want to do is use it directly to print a RANDOM ITEM from a list, but no good at all if you want to use PICKRANDOM in a program.

This idea of OUTPUT is very important. We met it before, in a different guise, when you typed 3 + 4, and Logo sent a message saying:

YOU DON'T SAY WHAT TO DO WITH 7

So you typed

PRINT 3 + 4

The same would happen if you typed COUNT :ABC. You would receive the Logo message:

YOU DON'T SAY WHAT TO DO WITH 26

COUNT outputs a value. The Logo word OP allows you to create new Logo words, like PICKRANDOM which OUTPUT results, and expect to be told what to do with them.

This is another example of Logo putting you in charge. Let's go back to that troublesome, but interesting second line of PICKRANDOM:

OP ITEM 1 + (RANDOM COUNT :X) :X

Let's work backwards from the right. That's what Logo is doing. First it finds :X, it makes sure that :X is in its memory as a list. Let's imagine that you have typed in PRINT PICKRANDOM :ABC. So in this case, :X stands for :ABC and we know that "ABC contains the 26 letters of the alphabet.

It then goes back to deal with COUNT :X. That's easy. It OUTPUTS 26 back to RANDOM. RANDOM 26 produces a random number between 0 and 25. Since we wouldn't want ITEM 0, we add 1 to what ever number it chooses.

This number in turn is OUTPUT to ITEM. Let us imagine the random number was 6, add 1, makes 7. So we have:

OP ITEM 7 :ABC

Since the 7th letter of the alphabet is G, PICKRANDOM :ABC OUTPUTS G to PRINT, and G is what appears on your screen. The same kind of logic is at work in:

TO DICE
OP 1 + RANDOM 6
END

Try PRINT DICE a few times, and you will soon see that you have the equivalent of throwing a six-sided dice.

Every Logo procedure, including the primitive procedures, is either a command or an operation. This is one of the most important single ideas you need to understand when learning the Logo language. There's more about it in the first reference section of the manual.

An operation outputs a Logo object (a word, a number, or a list).

A command does not output a Logo object. in order to see the difference, it might help you to think of some pairs of Logo primitives:

Command	Operation
SETH	HEADING
SETMODE	MODE
SETPOS	POS
SETPC	PC
SETBG	BG
SETCURSOR	CURSOR

In the left-hand column you have primitives which require inputs; they then use these inputs to carry out your instructions. On the left are words you can use to discover the state of Logo or the state of the turtle. They output **values** for your information. You will find more about this in **Section 21.**

SECTION FIFTEEN – BACK TO FRONT

A rather more complicated example of the kind of Logo word you can create yourself, using lists, is provided by REVERSE X. Perhaps you can imagine what REVERSE does. PRINT REVERSE :ABC churns out the alphabet backwards.

REVERSE, like PICKRANDOM, is not a word you find in Logo to begin with. It's not a primitive procedure. You have to make it. See whether you think it is a command or an operation.

```
TO REVERSE :LIST
IF :LIST = [ ] [OP [ ]]
OP SE REVERSE BF :LIST FIRST :LIST
END
```

Try it out on

```
PRINT REVERSE :TREES
```

Let's see if we can understand why it works. We are back with the recursive ideas we explored in Turtle graphics. All the work is done in the third line. First look at :TREES [ELM OAK ASH HOLLY THORN MAPLE]. One way of folowing this procedure through its steps would be as follows:

PRINT REVERSE :TREES
LIST = [ELM OAK ASH HOLLY THORN MAPLE]
LIST = [OAK ASH HOLLY THORN MAPLE]
LIST = [ASH HOLLY THORN MAPLE]
LIST = [HOLLY THORN MAPLE]
LIST = [THORN MAPLE]
LIST = [MAPLE]
LIST = [ ]
REVERSE OUTPUTS [MAPLE] TO REVERSE
REVERSE OUTPUTS [MAPLE THORN] TO REVERSE
REVERSE OUTPUTS [MAPLE THORN HOLLY] TO
    REVERSE
REVERSE OUTPUTS [MAPLE THORN HOLLY ASH] TO
    REVERSE
[REVERSE OUTPUTS [MAPLE THORN HOLLY ASH OAK] TO
    REVERSE
REVERSE OUTPUTS [MAPLE THORN HOLLY ASH OAK
    ELM] TO REVERSE
REVERSE OUTPUTS [MAPLE THORN HOLLY ASH OAK
    ELM] TO PRINT

MAPLE THORN HOLLY ASH OAK ELM

Until successive calls to REVERSE have emptied the LIST (in this case TREES), Logo can't get on with the business of building up the new list, in REVERSE order.

Look at that third line again:

```
OP SE REVERSE BF :LIST FIRST :LIST
```

SE has two inputs (REVERSE BF :LIST) and (FIRST :LIST), which it has to combine into a single list.

The second of these is easy to understand. FIRST :LIST is ELM. But what do we do about REVERSE BF :LIST. Well the program is telling the computer to do the same thing all over again to BF :LIST. In other words, the :LIST without its FIRST ITEM.

Once it gets down to an EMPTY :LIST, we put on the brakes by making it output an EMPTY :LIST, which we show as [], square brackets with nothing in them. At that time, the procedure OUTPUTS all the words it has been stacking up in REVERSE order.

You will almost certainly have to play with this for some time before you understand how it works.

```
PRINT REVERSE :TREES
```

produced

```
MAPLE THORN HOLLY ASH OAK ELM
```

But wouldn't it be nice to have another possibility: PRINT REVERSEALL :TREES and produce ELPAM NROHT YLLOH HSA KOA MLE. Well, it isn't difficult. We need another procedure to reverse the letters in a word:

```
TO REV :WD
IF :WD = " [OP "]
OP WORD REV BF :WD FIRST :WD
END
```

This is virtually the same as REVERSE :LIST. The only difference is that we are dealing with words instead of lists. The empty word is written " , while the empty list, as we have seen, is [ ]. WORD works rather like SE, but instead of gluing its inputs together to make a LIST, it makes a WORD. Try

```
PRINT (WORD "O "K "A "Y)
```

The inputs to WORD must be words, not lists. Single letters

and numbers are counted as words by Logo. So:

```
TO REVERSEALL :X
IF :X = [ ] [OP [ ] ]
OP SE REVERSEALL BF :X REV FIRST :X
END
```

By the time you have worked out how to reverse lists, and put them back together again, you will really know a good deal about Logo. You should also have worked out that REVERSE is an operation, which receives a list as input and OUTPUTS another list (a Logo object).

Here's a procedure which will reverse words and lists, words or lists.

```
TO REV :OBJECT
IF EMPTY? :OBJECT [OP " ]
IF LIST? :OBJECT [OP SE REV LAST :OBJECT REV BL :OBJECT]
IF WORD? :OBJECT [OP WORD LAST :OBJECT REV BL :OBJECT
END
```

SECTION SIXTEEN – MORE ABOUT NUMBERS

As we said in our Introduction,
we don't expect everyone to read the whole of this manual.
It has quite deliberately introduced and used more difficult
ideas and examples as we have progressed.

We are now going to talk about the way Logo does
arithmetic. We should say again that Logo regards numbers
as special kinds of words. If a word is a number, it doesn't
have to be preceded by quotes, but if you want to write
"24, Logo won't protest.

As you know, if we want to add two numbers together, we
usually write: :FIRSTNUM + :SECONDNUM. For example 3
+ 4, but we might say ADD 3 and 4. In the first example,
the plus sign comes between the words we want to add
together. In the second example, the word ADD comes first,
followed by the numbers we want to ADD. Logo, too, has
two different ways of doing arithmetic. Try the following to
discover how it works:

        PRINT SUM 4 3

        PRINT SUM 4 -3

        PRINT (SUM 4 5 6)

        PRINT 4 + 3

        PRINT 4 – 3

        PRINT 4 + 5 + 6

        PRINT 4 ∗ 5

        PRINT 3 ∗ 2

        PRINT PROD 3 2

        PRINT 5 / 2

        IF EQUAL? 3 3 [PRINT "OKAY]

        IF 3 = 3 [PRINT "OKAY]

The technical terms for these different ways of writing
arithmetic are "**infix**" and "**prefix**". Words like SUM,
PROD, QUOT are known as "**prefix operations**"
because they precede the numbers they operate on, whereas
symbols like ∗, +, /, >, = and so on are known as "**infix
operations**" because they are placed in between the
numbers the operate on. Like all other Logo operations, the

arithmetical operations output Logo objects, in this case numbers.

Some people use postfix operations, (eg 3 4 +) but Logo does not, and we won't worry about them here. Computers like to work along a line, from right to left, or left to right, without having to go back on their tracks, so they prefer prefix or postfix operations.

Because most people grow up at present learning maths the infix way (3 + 4), Logo offers both prefix and infix operations. But this can lead to trouble. In order to sort out the infix and prefix operations, Logo deals with all the infix operations first, in a strict order of priority. Division, multiplication, subtraction, addition, equality/inequality.

Logo does protest if you try the following:

IF COUNT :ABC = 26 [PRINT "OKAY]

Provided you still have the list "ABC in your computer memory, you receive a mysterious Logo message, saying

IF DOESN'T LIKE 5 AS INPUT

This is a good one to try on people, who think they know something about Logo.

The Logo interpreter works from right to left. It starts with the list of instructions [PRINT "OKAY], then finds 26 and the = sign. Ah, = is an infix operation which takes two inputs. So it first looks to see if there are any more arithmetical operations to be evaluated; finding that there aren't, it compares :ABC and 26. Clearly they are not equal, so it OUTPUTS the word FALSE back to COUNT. COUNT counts the letters in FALSE, one two . . . five, and OUTPUTS 5 to IF. Logo is now faced with:

IF 5 [PRINT "OKAY]

Since the word IF always looks for a condition which is either TRUE or FALSE, it is totally baffled and says:

IF DOESN'T LIKE 5 AS INPUT

There are various ways of writing the line so as not to confuse Logo. Here are three:

IF (COUNT :ABC) = 26 [PRINT "OKAY]

IF 26 = COUNT :ABC [PRINT "OKAY]

IF EQUAL? COUNT :ABC 26 [PRINT "OKAY]

Looking at them one by one. The first puts parentheses ( ) round COUNT :ABC. You have probably come across this use of ( ) at school. You have to evaluate whatever is inside the parentheses first. You then use the result for the rest of the calculation. Putting ( ) round COUNT :ABC means that the compter calculates this to be 26 before comparing the two inputs to =.

The second solution simply switches the COUNT :ABC and the 26, so that the Logo interpreter meets COUNT :ABC before it meets the arithmetical operator =.

The third solution uses the prefix operator EQUAL? so that by the time the interpreter reaches that point, it has two inputs ready, 26 and 26, so IF receives EQUAL? 26 26, to which it answers TRUE and therefore goes ahead and prints OKAY.

I think that exploring this example will save you a deal of time and trouble later on. You will find that the three infix operations dealing with equality (=) and inequality (< >) can give you similar problems. The simple rule for dealing with these problems is:
If there is an expression to be evaluated (eg COUNT :X or ITEM 3 :X) to the left of the infix operations (= < or >) that expression should be enclosed in parentheses. This is not wholly consistent, but it stems from the simultaneous management of infix and prefix operations. Try changing the following examples to make them work properly (see also Sections 20 of this manual).

        IF COUNT :ABC < 20 [PRINT "OKAY][PRINT [NO WAY]]

        IF ITEM 5 :ABC = "E [PRINT "OKAY][PRINT [NO WAY]]

        IF ITEM 5 :ABC = "Z [PRINT "OKAY][PRINT [NO WAY]]

We haven't met the second set of square brackets before. One way of picturing their meaning is as follows:
IF condition is TRUE, **THEN** [do this] **ELSE** [do that].
What it means is that if the condition tested by IF outputs false, the second list of instructions is followed instead of the first list.

SECTION SEVENTEEN – FOR TEACHERS

Many people will have read about the use of Logo with very young children, or with children who have severe difficulties in reading.

It is quite possible to make turtle graphics accessible to children who cannot read or write, and have never used a keyboard, using only four keys of the computer. Look at these procedures:

```
TO ERNS
ER OPNS
END

TO SETUP
ERNS
MAKE "F [FD 30]
MAKE "B [BK 30]
MAKE "R [RT 15]
MAKE "L [LT 15]
MAKE "H [HT]
MAKE "S [ST]
MAKE "U [PU]
MAKE "D [PD]
MAKE "C [CS]
MAKE "Q [TOPLEVEL]
END

TO SIMPLIFY
MAKE "KEY RC
IF NOT NAME? :KEY [SIMPLIFY]
RUN THING :KEY
SIMPLIFY
END
```

The word which does all the work is RC. This stands for READCHARACTER. Whatever key is pressed next becomes :KEY. Then if :KEY corresponds to any of the 10 **global variables** created by setup, Logo runs the THING corresponding to that variable. Spot the difference between THING "KEY and THING :KEY. If the KEY chosen is not one of those listed, SIMPLIFY waits for the next key press.

This can be made easier for children by marking the active keys with coloured labels, or arrows. It doesn't matter which keys you choose to operate the system. You can choose

fewer or more keys, depending on the ability of the child. At its most simple, you could have just two keys, one to move the turtle FORWARD, and another to turn it RIGHT. SETUP can be modified in any way you choose. The command ERNS clears the system of any oother global variables which may be lurking about. If you are working with a fresh Logo, it is not needed.

The SIMPLIFY procedure is recursive and keeps the whole system going, indefinitely.

The SIMPLIFY system can be used in two ways by teachers. The first is to give very young children access to the computer. The second use, with older children, is to invite them to find ways of improving the system. Obvious extensions are RUBOUT, which allows a child to cancel its last command. Modify the SETUP procedure as follows:

```
TO SETUP
ERNS
MAKE "F [REMEMER [FD 30] FD 30]
MAKE "B [REMEMBER [BK 30] BK 30]
MAKE "R [REMEMBER [RT 15] RT 15]
MAKE "L [REMEMBER [LT 15] LT 15]
MAKE "P [RUBOUT]
MAKE "Q [TOPLEVEL]
END
```

And add two new procedures:

```
TO REMEMBER :ACTION
MAKE "HISTORY :ACTION
END

TO RUBOUT
IF EQUAL? FIRST :HISTORY "FD [PE BK 20 PD]
IF EQUAL? FIRST :HISTORY "BK [PE FD 20 PD]
IF EQUAL? FIRST :HISTORY "RT [LT 30]
IF EQUAL? FIRST :HISTORY "LT [RT 30]
END
```

Another use of SIMPLIFY will be found in the discussion of VDU commands in the section 28).

Another aid developed by teachers at MIT and Edinburgh, when working with children with special needs, was to switch on a DRIBBLE file, which recorded every keystroke made by the child. This often provided the teachers with clues as to the problems they were encountering with the

computer, and provided the basis for developing a remedial strategy.

Our LOGO does not include DRIBBLE as a primitive because this is provided by the BBC Computer's operating system as ∗SPOOL.

Check the way this is used in your BBC User Guide. It has been hard to decide how much reference should be made in this manual to the BBC Operating System. We have decided to concentrate on Logo, and the special features of the language. But do remember that all features of the BBC Operating System are open to users of Logotron's Logo.

For further details concerning the use of operating system commands, see **Sections 20 (p.96)** and **28 (p. 147).**

These can be built into procedures and provided for children directly, to be used as if they were primitives, without worrying about the complexities of the operating system. Some rules have to be followed as such programs are creating an interface between two very different environments: Logo and the programs which make up the BBC micro's powerful operating system.

There will be teachers, particularly in secondary schools, who will be looking for ways to put extra intellectual challenge into Turtle graphics. We would suggest they begin to experiment with movement.

The first experiment involves a procedure callled:

```
TO MOVE :STEP
FD :STEP
IF KEY? [MAKE "STEP RC]
MOVE :STEP
END
```

Start with MOVE 0, then see what happens when you press the number keys. Watch the turtle accelerate, and slow down to a stop when you press 0. Build on this with:

```
TO MOVE1 :STEP :INC
STROBE
FD :STEP
MOVE1 :STEP + :INC :INC
END

TO STROBE
PD FD 0 PU
END
```

A French logophile, Alain Texier, has built on these simple beginnings to simulate bouncing balls, falling stones, billiards, and other moving objects. He calls it Logomotion, and there is plenty of room for imaginitive exploration of this particular idea.

Many teachers will be concerned that the BREAK key can easily be depressed accidentally, especially by very young or disabled children. The BREAK key has the effect of wiping all procedures and variable names from the workspace and from the Editor. This can be very discouraging.

There is no programmable solution. The circuit design of the BBC Micro does not allow us to disable the BREAK key. The best we can suggest is to cut out a length of cardboard, 0.25 inches wide and two inches long. Fold it three times into the shape of a W. Take the top off the computer and wedge the card behind and under the BREAK key, so that it cannot be pressed.

There is no disadvantage to this, so long as you are using Logo. If you still need the BREAK key for any reason, the best way of resetting the machine is to turn off the power and then turn it on again. It may seem a crude solution, but we believe it will be necessary in some environments, as the BREAK key is set so close to F9.

It has not proved possible to include a number of long forms of commands. For example, Logotron Logo only offers HT, instead of HIDETURTLE, or BG instead of BACKGROUND. If for any reason teachers feel children need the long form, it is the easiest thing in the world to construct the long from the short form. For example:

```
TO HIDETURTLE
HT
END

TO BACKGROUND
OP BG
END
```

SECTION EIGHTEEN – LIST PROCESSING

Mike Sharples, until recently a member of the Department of Artificial Intelligence at Edinburgh University and now at the University of Sussex, has spent a good deal of time studying the difficulties children encounter when they try to move from turtle graphics to other kinds of programming. In particular he has studied problems involving the use of language.

He writes in a recent paper: "Attempts by members of this department to teach list processing to children and adults have not been successful. Learners who enjoyed and profited from turtle geometry were bored and confused by lists."

Sharples suggests that if newcomers to programming are to discover the utility of list processing, they need to be provided with a toolkit, which goes beyond the list handling primitives found in Logo: – LIST SE WORD LPUT FPUT BF BL ITEM COUNT FIRST and LAST.

We have already met these primitives in this manual, but we would not expect that a reader would yet be able to do anything very exciting with them.

Sharples has given permission for us to describe two elements of a possible list processing toolkit for beginners. These are not programs to be studied by the beginner, so much as used.

We would expect teachers or parents to provide them ready made, on disk to be loaded by children and used to create projects.

The first of Mike Sharples' tools for list processing is a Phrasebook. His Phrasebook can be used to contain any information a child might want to look up: questions and answers; words and synonyms; English phrases and their foreign equivalents; Logo words and their definitions. The same set of programs can be used instead of the SIMPLIFY program described above in **Section 17.**

Only three commands are needed to operate the phrasebook: TEACH, FIND and REMOVE. TEACH adds an entry to the book. It accepts either words or lists as inputs.

```
TEACH "CAT "CHAT
CAT CHAT

TEACH [THE DOG] [LE CHIEN]
CAT CHAT
THE DOG LE CHIEN
```

Sharples has endeavoured to remain as close as possible to the spirit of turtle graphics in creating his Phrasebook. The child "teaches" the computer. With every addition to the Phrasebook, its full contents are shown to the child. If this becomes wearisome, it can easily be changed.

FIND, as you would expect, allows the user to write FIND [THE DOG], and be answered LE CHIEN.

The third command, FORGET, deletes an entry. FORGET [CAT]. The user is then shown the remaining entries, DOG LE CHIEN.

Even in this elementary form, the Phrasebook provides an introduction to reference aids and to the techniques of table loook-up and pattern matching. For example, the child might be given a core dictionary or thesaurus which she could extend:

```
FIND "SAD
UNHAPPY, MOROSE, MELANCHOLY, DEPRESSING, UNFORTUNATE

FIND "WILD
WILD IS NOT IN THE PHRASEBOOK

TEACH "WILD [UNTAMED, SAVAGE, UNRULY, BOISTEROUS]
```

With the additional command FOREVER, a child can easily produce quizzes or "conversations":

```
TEACH [WHAT IS THE CAPITAL OF FRANCE?] "PARIS

TEACH "HELLO [HI THERE]

FOREVER [FIND RL]

HELLO
HI THERE

WHAT IS THE CAPITAL OF FRANCE
PARIS
```

"Wild cards" or "jokers" for pattern matching a simple, but powerful extensions to the Phrasebook. A single question mark – ? – matches any single word; a double question mark – ?? – matches a series of words; a question mark,

followed by one or more letters, ?X for example, matches a single word and assigns it to a variable (in this case to X); two question marks followed by one or more letters, ??PHRASE for example, matches and assigns a series of words. The following examples show how this facility could be used.

    TEACH [?? MY ?X HURTS ??] [YOUR X? LOOKS VERY PAINFUL]

    TEACH [MY ?X LIKES ??Y] [TELL YOUR ?X TO STOP ??Y AND TAKE UP
    DANCING INSTEAD]

    FOREVER [FIND RL]

    DOCTOR, MY KNEE HURTS A LOT
    YOUR KNEE LOOKS VERY PAINFUL

    MY CAT LIKES PROGRAMMING COMPUTERS
    TELL YOUR CAT TO STOP PROGRAMMING COMPUTERS AND TAKE UP
    DANCING INSTEAD

Children will soon find new ways of using Phrasebook.

    TEACH "SQUARE [REPEAT 4 [FD 200 RT 90]

FIND will execute the commands as well as printing them. This allows you to use Phrasebook instead of SIMPLIFY, described in the last section.

    TEACH "Y [FD 20]
    TEACH "B [BK 20]
    TEACH "F [LT 30]
    TEACH "K [RT 30]
    FOREVER [FIND RC]

NB. Neither Mike Sharples nor the authors of this manual believe that computerised quizzes of the type described are a useful way of teaching children geography, or any other subject. The learning will happen because the child is teaching the computer. The child asks the computer to give it the name of the Capital of France. The computer "knows" the answer if it has been properly "taught".

Here are the procedures you need to create the Phrasebook. Just enter them throught the EDITOR, SAVE them and make sure they work as we describe.

Phrasebook

```
TO FOREVER :PROCLIST
RUN :PROCLIST
FOREVER :PROCLIST
END

TO TEACH :ENTRY :DEFINITION
MAKE "PHRASEBOOK INSERT :ENTRY
    :DEFINITION "PHRASEBOOK
PHRASEBOOK
END

TO INSERT :ENTRY :DEFINITION :BOOKNAME
IF WORD? :ENTRY [MAKE "ENTRY FPUT
    :ENTRY []]
IF WORD? :DEFINITION [MAKE "DEFINITION
    FPUT :DEFINITION []]
OP LPUT LIST :ENTRY :DEFINITION
    THING :BOOKNAME
END

TO MANY :PHRASE :NEXTMATCH :VARIABLE :MATCHBIT
IF EMPTY? :NEXTMATCH [MAKE :VARIABLE
    :PHRASE OP []]
IF EMPTY? :PHRASE [OP []]
IF EQUAL? FIRST :PHRASE FIRST
    :NEXTMATCH [MAKE :VARIABLE :MATCHBIT
    OP :PHRASE]
OP MANY BF :PHRASE :NEXTMATCH
    :VARIABLE LPUT FIRST :PHRASE
    :MATCHBIT
END

TO DISPLAY :BOOK
IF EMPTY? :BOOK [STOP]
IF (COUNT FIRST FIRST :BOOK) > 5 [PRINT
    FIRST FIRST :BOOK] [TYPE FIRST
    FIRST :BOOK]
REPEAT 6 [TYPE " ]
PRINT FIRST BF FIRST :BOOK
DISPLAY BF :BOOK
END
```

TO FIND :ENTRY
PRINT "
PRINT LOOKUP :ENTRY :PHRASEBOOK
PRINT "
END

TO FILL :RESPONSE
IF EMPTY? :RESPONSE [OP []]
IF OR EMPTY? BF FIRST :RESPONSE NOT
    EQUAL? FIRST FIRST :RESPONSE "?
    [OP FPUT FIRST :RESPONSE FILL
    BF :RESPONSE]
MAKE "WHAT BF FIRST :RESPONSE
IF AND EQUAL? FIRST :WHAT "? NOT EMPTY?
    BF :WHAT [MAKE "WHAT BF :WHAT]
OP SENTENCE THING :WHAT FILL
    BF :RESPONSE
END

TO CHECK :PHRASE :ENTRY
MAKE "WHAT FIRST :ENTRY
IF EMPTY? BF :WHAT [OP BF :PHRASE]
MAKE "WHAT BF :WHAT
IF NOT EQUAL? FIRST :WHAT "? [MAKE
    :WHAT FIRST :PHRASE OP BF
    :PHRASE]
OP MANY :PHRASE BF :ENTRY BF
    :WHAT []
END

TO MATCHES? :PHRASE :ENTRY
IF AND EMPTY? :PHRASE EMPTY? :ENTRY
    [OP "TRUE]
IF EMPTY? :ENTRY [OP "FALSE]
IF AND EMPTY? :PHRASE ?P FIRST :ENTRY
    [OP "FALSE]
IF EQUAL? "? FIRST FIRST :ENTRY [OP
    MATCHES? CHECK :PHRASE :ENTRY BF
    :ENTRY]
IF EMPTY? :PHRASE [OP "FALSE]
IF EQUAL? FIRST :PHRASE FIRST :ENTRY
    [OP MATCHES? BF :PHRASE BF
    :ENTRY]
OP "FALSE
END

```
TO ?P :WORD
IF NOT EQUAL? FIRST :WORD "? [OP
     "FALSE]
MAKE "WORD BF :WORD
IF EMPTY? :WORD [OP "TRUE]
IF EQUAL? FIRST :WORD "? [OP
     "FALSE]
OP "TRUE
END

TO DO :ALIST
IF AND DEFINED? FIRST :ALIST NOT
     MEMBER? FIRST :ALIST [IF NOT AND OR
     WORD SE] [RUN :ALIST]
OP :ALIST
END

TO LOOKUP :ENTRY :BOOK
IF WORD? :ENTRY [MAKE "ENTRY FPUT
     :ENTRY []]
IF EMPTY? :BOOK [PRINT SE :ENTRY
     [IS NOT IN THE PHRASEBOOK]
     OP []]
IF MATCHES? :ENTRY FIRST FIRST :BOOK
     [OP DO FILL FIRST BF FIRST
     :BOOK] [OP LOOKUP :ENTRY BF
     :BOOK]
END

TO FETCH :ENTRY
OUTPUT LOOKUP :ENTRY :PHRASEBOOK
END

TO DEL :ENTRY :BOOK
IF EMPTY? :BOOK [PRINT SE :ENTRY
     [IS NOT IN THE PHRASEBOOK] OP
     :BOOK]
IF EQUAL? :ENTRY FIRST FIRST :BOOK
     [OP BF :BOOK] [OP SE
     FPUT FIRST :BOOK [] DEL :ENTRY BF
     :BOOK]
END
```

```
TO DELETE :ENTRY :DIC
IF WORD? :ENTRY [MAKE "ENTRY FPUT
     :ENTRY []]
OP DEL :ENTRY :DIC
END
```

```
TO FORGET :ENTRY
MAKE "PHRASEBOOK DELETE :ENTRY
     :PHRASEBOOK
PRINT "
PHRASEBOOK
END
```

```
TO PHRASEBOOK
DISPLAY :PHRASEBOOK
END
```

Note Before using PHRASEBOOK for the first time, you need one further procedure:

```
TO SETUP
MAKE "PHRASEBOOK [ ]
END
```

Once you have a working Phrasebook, with contents, you will not need SETUP, unless you want to wipe out its entire repertoire.

Boxes

The second model offered by Sharples is the box. It is simply a computer model of a physical box, labelled with a single word name and holding an assortment of paper slips, each bearing a string of one or more words. Any of these words may be the names of other boxes and, together, a group of boxes can be used to build a more complicated structure.

The command PUT adds a new word or list of words to a particular box. Suppose we had four boxes: Nounphrase; Article; Noun; Adjective.

```
PUT "CAT "NOUN
NOUN
CAT
```

PUT "DOG "NOUN
NOUN
CAT
DOG

PUT [ARTICLE NOUN] "NOUNPHRASE
NOUNPHRASE
ARTICLE NOUN

The command CREATE scans the word pattern given as its input (enclosed in square brackets) and replaces every box name with a word or word list taken, at random, out of the box in question. This may be another box name. The scanning is repeated until no box name remains. For example. Suppose our boxes are filled as follows:

NOUNPHRASE
ARTICLE NOUN
ARTICLE ADJECTIVE NOUN

ARTICLE
A
THE

NOUN
CAT
DOG
HEN
MOUSE
FOX

VERB
CHASES
EATS
ESCAPES FROM

ADJECTIVE
FURRY
GINGER
ANGRY
FRIGHTENED
TERRIFIED

Now we might enter (if you want to try this out, you will have to type in Sharples's procedures):

CREATE [NOUNPHRASE VERB NOUNPHRASE]
the first time through, CREATE might substitute

ARTICLE NOUN EATS ARTICLE ADJECTIVE NOUN

Looking again, there is a new round of substitution to produce:

A FOX EATS THE TERRIFIED HEN

PUT can also be used to create a new box. For example, working still with our existing set of boxes, and

PUT [NOUNPHRASE VERB NOUNPHRASE] "SENTENCE

CREATE [SENTENCE]

would now be enough to achieve the same effect as the previous example.

FOREVER [CREATE RL]

can be used to eliminate the need to type CREATE [ ] every time.

Mike Sharples has used boxes to generate poetry:

PUT [LINE1 & LINE2 & LINE3] "HAIKU

The ampersnd (&) is interpreted as a Carriage Return, or a call for a new line.

PUT [ADJECTIVE ADJECTIVE NOUN VERB] "LINE1

and so on. If you want limericks, or poems with rhyming patterns, you have to create suitable boxes.

The command REMOVE deletes a box and its contents.

Boxes

Here are the procedures needed to create BOXES:

```
TO ADDTOVOCAB :APART :AWORD
IF NOT MEMBER? :APART :BOXES [MAKE
    "BOXES LPUT :APART :BOXES MAKE
    :APART LPUT :AWORD [] STOP]
IF NOT MEMBER? :AWORD THING :APART
    [MAKE :APART LPUT :AWORD THING
    :APART]
END

TO CHOOSE :PART
IF NOT MEMBER? :PART :BOXES [OP
    LIST :PART " ]
MAKE "PARTVAL THING :PART
OP ITEM (RANDOM COUNT :PARTVAL) + 1
    :PARTVAL
END
```

```
TO INPUT :PATBIT
MAKE "PATBIT LIST :PATBIT
MAKE "INWORD ASK :PATBIT
IF EMPTY? :INWORD [OP :PATBIT]
IF NOT EQUAL? :PATBIT :INWORD
     [ADDTOVOCAB FIRST :PATBIT :INWORD]
OP :INWORD
END

TO LOOKAT :PATBIT
IF EQUAL? FIRST :PATBIT "£ [MAKE
     "CHOICE INPUT BF :PATBIT] [MAKE
     "CHOICE CHOOSE :PATBIT]
IF EQUAL? FIRST :CHOICE :PATBIT [OP
     FIRST :CHOICE] [OP SCAN :CHOICE]
END

TO ASK :THEPROMPT
TYPE :THEPROMPT
TYPE ":
TYPE "
OP RL
END

TO ADDWORDS
MAKE "INWORDS ASK [WORDS]
OP IF EMPTY? :INWORDS [[]] [FPUT
     :INWORDS ADDWORDS]
END

TO PUT :LINWORDS :WPART
IF OR EMPTY? :LINWORDS EMPTY? :WPART
     [STOP]
IF WORD? :LINWORDS [MAKE "LINWORDS FPUT
     :LINWORDS []]
IF LIST? :WPART [PRINT [YOU MUST GIVE A
     WORD AS THE BOX NAME] STOP]
IF NOT MEMBER? :WPART :BOXES [MAKE
     "BOXES LPUT :WPART :BOXES MAKE :WPART []]
MAKE :WPART LPUT :LINWORDS THING :WPART
CONTENTS THING :WPART
END
```

TO CONTENTS :LINBOX
IF EMPTY? :LINBOX [STOP]
PRINT FIRST :LINBOX
CONTENTS BF :LINBOX
END

TO CREATE :PATTERN
MAKE "LASTONE :PATTERN
PPRINT SCAN :PATTERN
END

TO AGAIN
PRINT :LASTONE
PRINT "
PRINT CREATE :LASTONE
END

TO SCAN :PATTERN
IF EMPTY? :PATTERN [OP []]
OP SE LOOKAT FIRST :PATTERN
     SCAN BF :PATTERN
END

TO GET :PATTERN
MAKE "LASTONE :PATTERN
OP SCAN :PATTERN
END

TO REMOVE :WPART
IF NOT EQUAL? :WPART
     [] [MAKE "BOXES DELETE :WPART
     :BOXES]
END

TO PPRINT :LIST
IF EMPTY? :LIST [PRINT [] STOP]
IF EQUAL? FIRST :LIST "& [PRINT []]
     [TYPE FIRST :LIST TYPE "]
PPRINT BF :LIST
END

TO BOXES
PRINT :BOXES
END

TO FOREVER :LIST
RUN :LIST
FOREVER :LIST
END

```
TO DELETE :ELEMENT :LIST
IF EMPTY? :LIST [PRINT [THAT IS NOT THE
     NAME OF A BOX] OP :LIST]
IF EQUAL? FIRST :LIST :ELEMENT [OP
     BF :LIST]
OP FPUT FIRST :LIST DELETE :ELEMENT
     BF :LIST
END
```

"Phrasebook" and "Boxes" are quite new additions to the world of Logo. There is plenty of scope to experiment with them.

Once you thoroughly understand how to use them, then you may feel ready to pull them to bits, or build extensions.

But do remember, that Phrasebook and Boxes are included in this manual, not as sample programs, but rather as parts of a toolkit for working with Logo.

SECTION NINETEEN – TOOLKIT

This section is just what it says, a toolkit. Procedures are provided for a variety of uses. They are set out without comment. They can be used in two ways. If you are already familiar with programming in other languages, or already have a good deal of experience with Logo, they are just a sampler of ways Logo can be used on the BBC micro.

Alternatively they can be provided as tools for less experienced programmers, who want to extend their range. They could have particular application for teachers using Logo in a Secondary school, where many students may be converting from BASIC.

For example, even though Logo procedures are generally much easier to read than BASIC programs, they may feel lost without REM statements. Here is the procedure they need:

```
TO REM :REMARK
END
```

As you can see, it does absolutely nothing, but it does allow them to put in REM statements without confusing the computer.

```
REM [THIS PROCEDURE DRAWS A SQUARE!]
```

Sets

The next set of procedures deal with sets, and are rather more useful, allowing you to return the intersection or union of two sets, or to discover if one set is a subset of another.

```
TO EQUAL :A :B
IF WORD? :A [OP :A = :B]
IF WORD? :B [OP "FALSE]
IF SUBSET :A :B [OP SUBSET :B :A]
OP "FALSE
END

TO SUBSET :A :B
IF EMPTY? :A [OP "TRUE]
IF MEMBER FIRST :A :B [OP SUBSET
      BUTFIRST :A :B]
OP "FALSE
END
```

```
TO INTERSECT :A :B
IF EMPTY? :A [OP []]
OP IF MEMBER FIRST :A :B [FPUT
     FIRST :A INTERSECT BF :A :B]
     [INTERSECT BF :A :B]
END

TO UNION :A :B
IF EMPTY? :A [OP :B]
OP IF MEMBER FIRST :A :B [UNION BF
     :A :B] [FPUT FIRST :A UNION BF :A :B]
END

TO MEMBER :A :B
IF EMPTY? :B [OP "FALSE]
IF EQUAL :A FIRST :B [OP "TRUE]
OP MEMBER :A BF :B
END

TO MINUS :A :B
IF EMPTY? :A [OP []]
OP IF MEMBER FIRST :A :B [MINUS
     BF :A :B] [FPUT FIRST :A MINUS
     BF :A :B]
END
```

Back to Basics

As we have seen, Logo uses REPEAT and RECURSION, where other languages use FOR, WHILE etc. For those who still pine for BASIC, here are some useful procedures

```
TO FOREVER :INSTRUCTIONLIST
RUN :INSTRUCTIONLIST
FOREVER :INSTRUCTIONLIST
END

TO UNTIL :CONDITION :INSTRUCTIONS
IF RUN :CONDITION [STOP] [RUN
     :INSTRUCTIONS]
UNTIL :CONDITION :INSTRUCTIONS
END

TO WHILE :CONDITION :INSTRUCTIONS
IF RUN :CONDITION [RUN :INSTRUCTIONS]
     [STOP]
WHILE :CONDITION :INSTRUCTIONS
END
```

Graphics

Here are some graphics tools. You have already met MOVETO. I expect you will think of some more if you get at all deeply involved in Turtle graphics.

```
TO MOVETO :X :Y
PU SETPOS SE :X :Y
END

TO LINE :X1 :Y1 :X2 :Y2
MOVETO :X1 :Y1
PD SETPOS SE :X2 :Y2
END

TO DIST1 :DX :DY
OP SQRT :DX * :DX + :DY * :DY
END

TO DIST :X1 :Y1 :X2 :Y2
OP DIST1 :X1 − :X2 :Y1 − :Y2
END

TO TDIST :PT
OP DIST XCOR YCOR FIRST :PT FIRST BF :PT
END
```

Pretty Printing

Teachers may often want to print out procedures. This not easy if they include long lines. In order to introduce a break in lines for printing purposes, we have included this suite of PRETTYPRINT programs. The way to use them is as follows: LOAD "PRETTYPRINT making sure that the PRETTYPRINT file includes all the procedures set out below.

Set the maximum line length you can accommodate on your printer by MAKE "PW 55, for example; the default value of "PW is 39. Then LOAD the procedure you want to PRETTYPRINT, and type PP "PROC where PROC is the name of the procedure. If you want to print out a number of procedures use MAP:

```
MAP "PP [PROC1 PROC2 PROC3 …]
```
MAP has many uses when you want to run a procedure a number of times with different inputs.

```
TO DEF :L
IF EMPTY? :L [ STOP ]
PR FIRST :L
DEF BF :L
END

TO MAP :F :L
IF EMPTY? :L [ STOP ]
RUN SE :F [ FIRST :L ]
MAP :F BF :L
END

TO PP :PROC
MAKE "PL TEXT :PROC
MAKE "PL LPUT [ END ] SE BL LIST (SE
      "TO :PROC FIRST :PL " BF :PL
MAKE "LPOS 0
PPLINES :PL CR
END

TO CR
PR "
MAKE "LPOS 0
END

TO PPLINES :L
IF EMPTY? :L [ STOP ]
PPLIST FIRST :L CR
PPLINES BF :L
END

TO PPLIST :L
IF EMPTY? :L [ STOP ]
IF WORD? FIRST :L [ PPWORD FIRST :L ]
      [ TYPE "[PPLIST FIRST :L TYPE
      "] SPACE ]
PPLIST BF :L
END

TO SPACE
MAKE "LPOS :LPOS + 1
TYPE CHAR 32
END
```

TO PPWORD :W
IF (LEN :L) > :PW [SPACE SPACE]
TYPE :W
MAKE "LPOS LEN :W SPACE
END

TO LEN :W
OP SUM :LPOS COUNT :W
END

TO TEXT :NAME
SAVE "PROG :NAME
SETREAD "PROG
OP FPUT BF BF RL READLINE [ ]
END

TO READLINE :TEXT
MAKE "LINE RL
IF EMPTY? :LINE [OP "]
IF [END] = :LINE [SETREAD [ ] *DELETE "PROG OP :TEXT]
OP READLINE LPUT :LINE :TEXT
END

TO DEFINE :NAME :LIST
SETWRITE "PROG
PR (SE "TO :NAME FIRST :LIST)
DEF BF :LIST
PR "END
SETWRITE [ ]
LOAD "PROG
*DELETE "PROG
END

SECTION TWENTY – LOGO GRAMMAR

Introduction

The sections which follow make up a reference manual, rather than a guide for newcomers. However, if you are a newcomer, you will find it useful to extend your understanding of procedures mentioned in the introductory sections of the manual, and they will certainly be an essential guide once you have gained confidence in programming.

If you have used Logo before, these sections should provide all you need to use Logotron's Logo on the BBC Model "B" Micro. In most respects, Logotron's Logo conforms to the conventions established by Logo Computer Systems Inc, of Montreal, and Systèmes d'Ordinateurs Logo International, of Paris in implementing Logo for a number of popular microcomputers, including Apple II, the IBM PC, the Sinclair Spectrum, the Atari range and the Coleco Adam. There are, however, some special features, made possible by the BBC's operating system. In some cases we have departed from the standard implementations, and we do draw attention to these (see, particularly, EDIT, ERASE, PO and SAVE. In every case, there are important innovations).

The Installation Guide (**Section 1**) gives full instructions concerning the installation of the 16K ROM containing the Logo system. This second part of the manual is organised in sections, each covering a particular kind of primitive – The Turtle, Words and Lists, Variables, Defining and Editing. Within these sections, you will find a short description of the relevant primitives, in alphabetical order.

If you are not sure where to find a particular primitive, consult the main index at the back of the manual. With most primitives, you will find one or more examples of the way it is used.

There are some conventions used in describing the primitives. Where a primitive requires a number as its input, FORWARD, for example, we write FORWARD *n*. Similarly LEFT *n* or SIN *n*.

In these cases, we are describing the kind of input a primitive requires; we are not speaking about the way the

input is written when you type it into the keyboard. When you come to use one of these primitive procedures, you replace the *n* with a number. Where a primitive requires 2 numbers, REMAINDER, for example, we write REMAINDER *a b*.

Similarly: PROD *a b*. Where a list of instructions follows the primitive, as with REPEAT, for example, we write REPEAT *instructionlist*. The word *list* indicates that the instructions should be enclosed in square brackets [ ].

Where a primitive takes the name of a procedure or a file, as in SAVE, we write SAVE *filename procname*. When you use such a primitive, you replace *filename* with the name of the file you are creating, and *procname* with the name of the procedure you wish to save.

In the case of the THING attached to a NAME, we speak of an *object*. For example, LPUT *object list*. A Logo *object* can be a word, a number, or a list.

Where a conditional is involved. For example:

```
IF :X = :Y [PRINT "OKAY]
```

we would write IF *pred instructionlist*. Pred stands for predicate. It must be a condition which the computer can decide to be TRUE or FALSE. It often involves deciding whether one number is bigger than another, or whether two numbers are equal, or whether a list has anything in it, or whether the current pen colour is one colour or another.

another example:

```
IF AND EQUAL? :X :Y LIST? :X [PR [YOU'VE GOT IT]] [PR [TRY AGAIN]]
```

we would write: IF AND *pred1 pred2 instructionlist1 instructionlist2*

This gives us a convenient notation for describing the Logo procedures. If you find it bothers you, when you begin using the reference manual, find someone else who can explain it to you. It's really easy once you have got hold of the idea.

Logo, like other languages, has a grammar. The conventions described above, allow us to refer to that grammar or syntax in a consistent way.

Procedures

The basic building blocks of Logo are the procedures it has in its memory from the moment you switch on your computer. These are the PRIMITIVES, which really is short for primitive procedures, the roots from which other procedures are derived.

You can discover what they are by typing PRIMITIVES.

These can be used directly by typing on the keyboard. When you are using Logo in this way, we say you are at TOPLEVEL. For example, type FORWARD n, where n is a number, and the turtle will immmediately move forward. But the PRIMITIVES can also be used to build up other PROCEDURES, which can then be used by name at TOPLEVEL as if they were PRIMITIVES. These PROCEDURES, which you create, are built in the part of the memory, which will be referred to throughout this manual as **workspace**. It is the space in which you work.

Using the BBC micro, the amount of available workspace changes according to which MODE you are in. This is inevitable, as the BBC micro uses a variable amount of memory to manage the screen. If your workspace overflows, you will receive a Logo Message on your screen, saying OUT OF SPACE.

Procedures are defined between the words TO and END, and have the form:

TO *name inputlist* **CR**
instructionlist **CR**
END

for example

    TO HELLO :NAME
    PR (SE "HELLO WORD :NAME ". [THERE MUST BE MORE TO LOGO THAN
    THIS.])
    END

    HELLO "RUTH
would elicit the response:

    HELLO RUTH. THERE MUST BE MORE TO LOGO THAN THIS.

Note: The first line of a procedure is called the **title line**. It always begins with TO, followed by the name of the procedure, folowed by the names of any variables which are required as inputs to the procedure. Examples are:

```
TO SQUARE :SIDE

TO BOX :HEIGHT :WIDTH

TO HELLO :NAME

TO STAR :POINTS :SIZE :COLOUR
```

The last line must always consist of END by itself.

Since procedures work just like extra primitives, procedures can in turn be used to build new procedures. For example:

```
TO SQUARES :SIDE :NUMBER
REPEAT :NUMBER [SQUARE :SIDE RT 360 / :NUMBER]
END
```

The procedure SQUARE is used to build the new procedure SQUARES. SQUARES could be referred to as the superprocedure, and SQUARE as the subprocedure. When a subprocedure is called from inside a superprocedure, you are no longer at TOPLEVEL. Logo is working outside your direct control. You could only intervene by stopping it, pressing the ESCAPE or BREAK keys.

If you type a word that has not been defined as a procedure, you will get a message. For example, type JEAN. Logo will respond: I DONT KNOW HOW TO JEAN

Objects

Logo objects are words or lists used as inputs or outputs from procedures. A word is a series of alphanumeric characters. A word is contained between two delimiters (see next subsection, which defines delimiters). Each character in a word is said to be an element of that word.

A double quotes mark (") at the beginning of a word enables Logo to distinguish words from primitives and procedure names. There is also a word with no characters in it, called the empty word. It is written with a double quotes mark " . Try
PR "R2D2

PR "WELCOME

PR "

"R2D2 "WELCOME and " are all words in Logo. Numbers (eg 23 134.567 1000) are also words in Logo, but they can be written without the quotation marks, as can the boolean values, TRUE and FALSE.

PRINT 25

PRINT "25

produce the same response.

A list consists of a series of Logo objects usually enclosed in square brackets [ ]. The objects will either be words (or numbers) or other lists. The individual elements of a list are separated by blank spaces. There is also an empty list, written [ ]. [CAT 123 MOUSE HOUSE] is a list containing four elements. [[CAT 123] [MOUSE HOUSE]] is a list containing two elements, each of two elements.

A Logo object may also be the THING of a variable NAME. For example:

MAKE "FRUIT [APPLES AND PEARS]

:FRUIT is a Logo object; so is THING "FRUIT.

Delimiters

The word delimiter is one of those awful bits of computer jargon, which strike terror into the first time user. Never fear. You already use delimiters when you read and write English. You call them punctuation marks. A sentence begins with a capital letter and ends with a full stop. Quoted speech is enclosed in quote marks. A question ends with a question mark. Words are separated from one another by leaving blank spaces. These are all delimiters in ordinary written English.

When a computer program scans along a line of symbols typed in from the keyboard, or fed into memory off a disk or cassette, it relies on delimiters, its own form of punctuation, to know where it is and what to expect.

For example a left-handed square bracket [tells it with absolute certainty to expect a LIST. The other bracket] tells it the LIST has now finished. A carriage return at the end of a line says that the line is finished. In Logo, as in English, we leave spaces between words, and Logo gets confused if you run two words together, unless they are separated by

some other delimiter. These other delimiters are:

    [ ] ( ) = > < + − ✳ /

But even when one of these does appear in a Logo line, it is usually clearer to leave a space on one side or the other:

    MAKE "FRUIT[APPLES]

is quite correct, but

    MAKE "FRUIT [APPLES]

is a lot easier to read. Logo takes no notice of extra spaces.

Be careful with the Minus sign as PRINT 7 –6 would produce:

    7
    YOU DON'T SAY WHAT TO DO WITH –6

You should have written PRINT 7 – 6

Inputs

Some **Procedures** and **Primitives** need inputs to enable them to work. Inputs are Logo objects (words or lists). They may either be given explicitly at TOPLEVEL or be passed to the procedure at the point at which a procedure is called inside a running procedure, as **output** from another procedure. For example, at TOPLEVEL:

    PRINT COUNT [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]
    26

    TO CHECK :LIST :NUMBER
    IF :NUMBER = COUNT :LIST [PR "OKAY]
    END

    CHECK [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] 26

    OKAY

The procedure takes the :NUMBER as 26 and the :LIST as the letters of the alphabet, COUNTS the latter, compares the two values, and prints OKAY. If CHECK is used without two inputs, LOGO will complain

    NOT ENOUGH INPUTS TO CHECK

In the following procedure, CHECK is a subprocedure, and one of its inputs is the output of the procedure DICE.

```
TO EVEN.THROW
CHECK REMAINDER DICE 2 0
END

TO DICE
OP 1 + RANDOM 6
END
```

Quotes, Dots and Brackets

Unless you specifically indicate otherwise, using QUOTES ("),
DOTS (:) or BRACKETS ([ ]), Logo interprets every word as a
primitive or a procedure. The only exceptions are numbers
(written with digits 0. . .9). If it does not find the word in its
lists of primitives and defined procedures, it sends the
message

```
I DON'T KNOW HOW TO . . . .
```

The QUOTES (") indicate to Logo that the sequence of
characters immediately following, and ending with a blank
space, is a word. Even if it is the name of a procedure or a
primitive, it will be treated simply as a word. For example:

```
PRINT "PRINT

PRINT
```

The DOTS (:) tell Logo that the sequence of characters
immediately following, and ending with a blank space, are
the name of a Logo object to be evaluated. The DOTS tell
Logo to refer to the THING attached to that name. For
example:

```
PRINT :FRUIT
APPLES AND PEARS
```

Unless Logo is expecting a list of instructions, as it does
after the primitives REPEAT, RUN and IF, the words
enclosed in BRACKETS ([ ]) are treated as a list of Logo
words, each preceded by QUOTES. For example:

```
PRINT [PRINT FORWARD 100]
```
would simply print out the words contained in the square
brackets, it would not attempt to execute them.

Commands and Operations

In Logo, **primitives** and **procedures** can be conveniently
divided into two categories: **commands** and **operations**.

A command never outputs a value, whereas **an operation always outputs a value**. The value output by an operation must be a Logo object (a word or a list, including numbers, boolean values, and names of primitives, procedures and variables).

Typical commands: FORWARD, SETH, REPEAT, PRINT, SQUARE, ...

Typical operations: SQRT, WORD, EMPTY?, LIST, REMAINDER, REVERSE . . . .

Consequently, any procedure which is an operation can act only as the provider of an input to another procedure. For example, if one wanted to construct an operation to test if a number was even:

```
TO EVEN? :NUMBER
0 = REMAINDER :NUMBER 2
END
```

You then try it out with:

```
IF EVEN? 57 [PRINT "OKAY]
```

and get the Logo message:

```
YOU DONT SAY WHAT TO DO WITH FALSE IN EVEN
```

The fix is the word OP.

```
TO EVEN? :NUMBER
OP 0 = REMAINDER :NUMBER 2
END

IF EVEN 56 [PRINT "OKAY]
OKAY
```

If a procedure is to work as an operation, **it must include the command, OP**. It then outputs its result to the procedure which calls it. Another example

```
TO MAX :A :B
IF EQUAL? :A :B [OP :A ]
IF :A > :B [OP :A] [OP :B]
END

PRINT MAX 7 19
19

PRINT MAX 6 6
6
```

If you try to use a command as input to another command, you get a Logo message. For example

```
PRINT FD 100
FD DIDN'T OUTPUT TO PRINT
```

You can turn a operation into a command by changing the word OP and substituting some other command. Try PRINT in the examples given above.

This distinction is so important that when we come to list the primitives, we indicate in each case whether it is a command or an operation.

Variables

Variables are created in two ways in Logo. First, as inputs to procedures, declared in the title line, as in TO SQUARE :SIDE, for example. And second, through assignment statements, using the primitive MAKE.

Where variables are created as inputs to a procedure, they are LOCAL to that procedure and any subprocedures. For example:

```
TO STAR :SIDE
REPEAT 36 [SQUARE :SIDE RT 10]
END
```

If you then run STAR 200 (assuming SQUARE has been previously defined), the turtle will draw a star of the size required, but then :SIDE will immediately lose its value. This can be demonstrated by typing:

```
PRINT :SIDE
:SIDE HAS NO VALUE
```

The variable :SIDE was LOCAL to those procedures (in this case STAR and SQUARE) to which it was an input, and has no effect on any other procedures. Note that STAR was able to pass the variable :SIDE on to its subprocedure SQUARE :SIDE.

The LOCAL character of these inputs allows one to use the same variable names :X :SIZE :LIST :NUM over and over again as inputs to different procedures, all of which may be in the memory simultaneously.

When you create a variable using the Logo assignment word

MAKE, for example:

```
MAKE "X 3

MAKE "FRUIT [APPLES AND PEARS]

MAKE "ALPHABET
      [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]
```

That variable is GLOBAL, and will exist independently of any procedure which calls it, unless it is specifically ERASEd. Like procedures, global variables can be EDITed, ERASEd, Printed Out or SAVEd.

This is a major innovation in SOLI's implementation of Logo for the BBC micro. To refer to a variable by name, and to distinguish it from a procedure name, it must be enclosed in a list and preceded by quotes. For example:

```
ED ["FRUIT]
```

would move a variable called FRUIT to the EDITOR. Whereas:

```
ED "FRUIT  or  ED [FRUIT]
```

would lead Logo to look for a procedure called FRUIT. You will find more about this in the sections dealing with each primitive.

A variable can contain any Logo object, words (including numbers), lists, and another variable. For example:

```
MAKE "ABC "ALPHABET

MAKE "ABC :ALPHABET
```

In the first case, :ABC stands for the word "ALPHABET. In the second, :ABC stands for the list of letters created previously with the name "ALPHABET. When creating a variable, one does not have to declare the data type as one does in some other computer programming languages. So long as the THING attached to the name is a Logo object, it is valid assignment.

There are two ways of getting at the value assigned to a particular variable name:

```
PRINT :FRUIT
APPLES AND PEARS

PRINT THING "FRUIT
APPLES AND PEARS
```

The second form is particularly useful where you have a variable name stored without DOTS as an element of a list. Work through the following examples:

MAKE "HEATHER [CALCIFUGE LOW-GROWING SHRUB]

MAKE "CALCIFUGE [LIME-HATING PLANT]

MAKE "LOW-GROWING [PLANTS WHOSE HEIGHT RARELY EXCEEDS 50 CM.]

MAKE "SHRUB [PLANT WITH WOODY STEMS PERSISTING FROM ONE
    YEAR TO ANOTHER]

MAKE "H [EIGHTH]

MAKE "EIGHTH [BETWEEN SEVENTH AND NINTH]

PR :HEATHER

PR THING "HEATHER

PR THING FIRST THING "HEATHER

PR THING THING FIRST "HEATHER

PR FIRST THING THING "HEATHER

PR THING FIRST "HEATHER

PR THING FIRST BF :HEATHER

PR THING LAST :HEATHER

Logo Lines

A Logo line can be much longer than a line on your monitor screen, no matter what mode you are in. It ends when you press the RETURN key, but cannot be more than 255 characters. Before you reach the limit, a warning bleep tells you to stop. Here is a complex Logo Line:

IF EQUAL? :LETTER FIRST :ALPHABET [MAKE "WORD1 FPUT :LETTER
:WORD1] [SEARCH :LETTER BF :ALPHABET

Here are some guidelines to help you interpret a complex Logo line:

1. When you see a procedure or primitive name, be sure you know:

   a. whether it is a command or an operation;
   b. how many inputs it should take.

2. The first procedure of a Logo line must always be a command.

3. Be sure to account for every input to a procedure.

4. When all the inputs to a command have been accounted for, the next procedure must be another command.

For example:

```
MAKE "WD "HAPPY

PRINT SE [I AM] WORD BL :WD "IER
```

Analysing this line, we see that PRINT is a command with a single input. This must be the output of SE, which is an operation with two inputs.

The first input to SE is the list [I AM]. The second is the output of the operation WORD. The latter is once again an operation with two inputs. The first is the operation BL, which has a single input :WD. The second input to WORD is "IER.

Since there are no more procedure names to account for on the line, we have finished. The following diagram summarises what we have done:
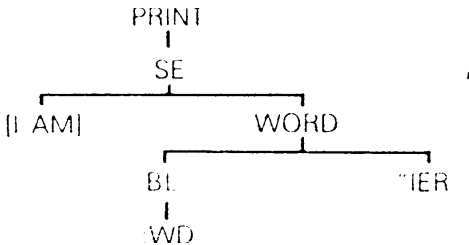
In the instance above, Logo would print HAPPIER.

Arithmetic

Numbers are a special kind of Logo word. You don't have to put quote marks in front of a number, but it's fine if you do.

```
MAKE "A "20
MAKE "B 20

PR :A
20

PR :B
20
```

The following priority is given to infix **arithmetic operations**, all of which take priority over other operations appearing to the left of them in a Logo line (see below and Section 24).

	Infix
Division	/
Multiplication	∗
Subtraction	–
Addition	+
Equality/inequality	=
	<
	>

So division is executed before multiplication; both are executed before subtraction, which is executed before addition. All infix operations performed before prefix operations appearing to their left in the same Logo line. The order can be changed by using parentheses ( ).

```
PR 4 + 6 / 2
7
PR (4 + 6) / 2
5
```

If you consider that these infix operations are also primitive operations, you will quickly see that they don't behave like other Logo operations. The normal way for a Logo operation to work is

operation input input . . . .

taking inputs from the right, and outputting the result to the procedure on the left. For example:

```
PRINT REMAINDER 14 3
2
```

REMAINDER takes two inputs, divides the second into the first, is left with a remainder of 2, which is passed back to print.

With infix operations you have instead:

input operation input

Logo copes with this confusion by always dealing with certain arithmetic operations first. These take priority over all

other operations appearing to their left in a Logo line. For example:

    PR RANDOM 2 + 3

is read by Logo as

    PR RANDOM (2 + 3)

and not

    PR (RANDOM 2) + 3

Another example

    IF COUNT :ALPHABET = 26 [PRINT "OKAY]

Logo will try to compare :ALPHABET to 26, and will output the boolean value FALSE to COUNT, which will output 5 to IF, at which point, there will be a Logo message

    IF DOESN'T LIKE 5 AS INPUT

If the line had been written:

    IF (COUNT :ALPHABET) = 26 [PRINT "OKAY]

the Logo interpreter would have been perfectly happy. Equally it could have coped with:

    IF 3 + 4 = 7 [PRINT "OKAY]

because it would have found two infix operations = and +, and would have dealt with them in the order of precedence described above, first the + then the =.

    IF SUM 3 4 = 7 [PR "OKAY]
    SUM DOES'NT LIKE FALSE AS INPUT

The confusion arises when infix operations are mixed with prefix operations. Remember, the infix arithmetical operations (listed above) are always evaluated before operations to the left of them in a Logo line, unless you have indicated otherwise by judicious use of parentheses ( ).

Note: this is a controversial area in the design of Logo, and is poorly documented in most books. Some Logo interpreters get rid of the confusion by not allowing infix operations.

Screens, Modes and Prompts

When you type instructions to Logo, you have a choice of three different parts of the system, each of which behaves in a different way.

At the TOPLEVEL, every instruction you type in is interpreted and executed inside the workspace immediately you press

the RETURN key. You are at TOPLEVEL as soon as you switch on the computer. The prompt at this level is a question mark at the beginning of the line. Whenever you see that questionmark, you know Logo is waiting to receive your instructions at TOPLEVEL.

When a procedure is running, the workspace is under the control of Logo, and not under your direct control, as it is at toplevel, but the workspace is the same. Logo can only attend to one set of instructions at a time.

Then there is a reserved area of memory called the **Logo Editor**, where you can write new procedures or change old ones. You enter the Editor by typing
EDIT or just ED.

In the Editor, the prompt is a solid square cursor, instead of the question mark. Inside the Editor, Logo does not carry out any instructions, it just waits and records instructions in its own portion of the computer's memory, which is called the **Edit Buffer**. When you have the procedure just right, you type CTRL C, and the Editor writes the new procedures into the workspace ready for use. Special commands are available to allow you to move the cursor freely around the screen for ease in changing procedures.

Finally, you can define procedures without going into the Editor by just typing the title line of a new procedure.

    TO name input1 input2 ....
As soon as you hit the RETURN key, you will see a new kind of prompt

>

This tells you you are defining a new procedure. You are no longer at TOPLEVEL. It will disappear as soon as you type the line:

    > END
and return you to TOPLEVEL with the message

    name DEFINED
Using the BBC Micro, you have several choices of MODE, and this will affect the things you can do in LOGO. You should read about the 8 different MODES in the BBC User Guide (p. 160).

You will discover that graphics are available in some MODES (0 1 2 4 5) and not in others (3 6 7). In MODE 2, you can have 16 colour combinations; in MODES 1 & 5, four colours, and in MODES 0 & 4, only two colours. In the text MODES (3 & 6), you can choose a text colour and a background colour. MODE 7 is a special kind of text called Teletext is a subject on its own.

The problem for Logo of changing MODES stems from the fact that the monitor demands different amounts of memory for different MODES. When you enter Logo you are in MODE 4. You change MODE by typing SETMODE *n*.

You can soon see that this effects the amount of memory you have available for writing procedures by typing PR NODES.

When you want to change MODES, you should save your current procedures, either on disk (or cassette) or in the Editor. Type: EDALL; Press ESCAPE to leave the Editor; then ERALL, which erases all procedures from the workspace, then SETMODE n; then ED to enter the Editor; then CTRL C, and all the procedures will be read back from the Editor into the workspace.

If there are too many procedures or variable names to hold in the Editor, which can contain 1,000 characters, you will have to SAVE them onto a disk or cassette, and then LOAD them back into the workspace after you have changed MODE.

If you ever forget which MODE you are in, type PR MODE. Providing you are in a MODE which offers graphics (0 1 2 4 5), you can choose between two screeens, the Graphics Screen or the Text Screen. In the Graphics screen, you can type in commands, or define procedures, only at the very bottom of the screen. The main part of the screen is reserved for the turtle and any pictures you may draw. In the Text Screen, you can write all over the screen. When you turn on the computer you are in Text Screen. To switch to the Graphics Screen, either give a command to the Turtle (see **Section 21**) or type CS.

To leave the Graphics Screen and return to Text Screen, type TS. If you do this, any pictures you have drawn will be lost.

So be sure you don't mind, or have saved the picture with SAVEPICT (see Section 28).

Recursion

Logo allows recursive procedures. A recursive procedure is one which calls itself. Here is a recursive explanation of how you walk. To walk, put your left foot in front of the right, then the right foot in front of the left, then walk.

Examples are given throughout this manual of recusive procedures. The most important thing to understand about them is that they never stop without some kind of limiting condition, which brings them to an end.

You may have come across the word recursion before in arithmetic. For example, express one third as a decimal fraction:

1 / 3 = 0.333333333333333333333333 recurring

We usually put a limit on such an expression by stopping it after three repetitions, and writing 0.333.

Here is a typical procedure, including a stop clause:

```
TO SPI :SIDE :ANGLE :INC
IF :SIDE > 300 [STOP]
FD :SIDE RT :ANGLE
SPI :SIDE + :INC :ANGLE :INC
END
```

Recursion is not always the most efficient way of reaching your goal. Here, for example are two ways of generating the nth member of the Fibonacci series (1 1 2 3 5 8 13 . . . .):

```
TO FIB :N
IF :N < 3 [OP 1]
OP (FIB :N − 1) + (FIB :N − 2)
END

TO FIB :N
MAKE "A 1 MAKE "B 1
IF :N < 3 [OP 1]
REPEAT QUOT (:N − 2) 2 [MAKE "A (:A + :B) MAKE "B (:A + :B)]
IF EQUAL? REMAINDER :N 2 1 [MAKE "A (:A + :B) OP :A] [OP :B]
END
```

The two procedures produce identical results. The recursive version (the first one) is far the most elegant and easy to read. But it can gum the computer up for several minutes producing the 20th Fibonacci number. The second procedure is not elegant, but it is efficient. Try them both out for yourself. The recursive version is often used as a benchmarking test, to discover how efficiently a particular implementation of Logo handles recursion. Bad implementations crash when asked to compute FIB 20 the hard way. See how long it takes Logotron Logo.

The BBC Operating system

BBC Micro users soon discover two kinds of command, which have a wide variety of uses. These are command, which begin VDU or *. Both kinds of command are available in Logo. The only way they differ from the way they are described in your BBC User Guide (sections 34 and 42) is that the primitive procedure VDU requires a list as its input. For example:

    VDU [19 1 4 0 0 0]
in MODE 5 would change logical colour 1 (red) to actual colour 4 (blue)

If you want to input a variable value to the list, you have to use the construction

    VDU SE 19 1 :NEWCOLOUR 0 0 0
The Logo interpreter will supply the value of :NEWCOLOUR and SE will then output a list to VDU. If one wrote:

    VDU [19 1 :NEWCOLOUR 0 0 0]
You would get a Logo Message:

    VDU DOESNT LIKE :NEWCOLOUR AS INPUT
In other words, it would simply be treating :NEWCOLOUR as an element in the list, not as the name of a variable to be checked against a value.

Both VDU and * commands can usefully be built into procedures, which can then be given to children as tools. This is particularly useful in cases where the children might find the operating system hard work.

The problem with ∗ or star commands is that the star or asterisk can mean three different things in Logo. It can signal the beginning of a call to the BBC micro's operating system; act as a wild card (see the BBC DFS manual), or act as an infix arithmetic operator in Logo. The rules for using star(∗) commands are as follows.

If they come at the beginning of a Logo line or a Logo list, there is no problem. All subsequent numbers or words on the line, or in the list, will be sent to the operating system.

If you wish to insert a star command in the middle of a Logo line, it is necessary to enclose the star command and its inputs in parentheses. For example:

```
SETBG 9 ( ∗FX 9, 50) ( ∗FX 10,50) FD 100
```

There is one exception. According to the BBC User Guide, if you type ∗.1, you should be given information on the contents of the disc in Drive 1. Logo, however, interprets this as a multiplication by 0.1. The solution is to use the back slash (\), which instructs Logo to treat the next character literally, without reference to its meaning. You type ∗.\1.

If these rules are followed, there should be no problems.

NB. Note the difference between the treatment of VDU and ∗ commands. The former acts like any other Logo primitive, taking a list of inputs, without commas separating them. Numbers larger than 255, which require two bytes of memory, are denoted using quotes ("), eg "1278. Whereas star (∗) commands have to be isolated from Logo.

WARNING: Some calls to the BBC operating system, like ∗COMPACT and ∗FORMAT, can crash Logo. In such cases, CTRL BREAK is required to reboot Logo. This is not a bug in Logotron Logo, it is a feature of the BBC Micro.

SECTION TWENTY-ONE – TURTLE GRAPHICS

This section of the manual, and those which follow, consist of descriptions of each primitive of Logotron Logo for the BBC Micro.

In bold face, at the beginning of each entry, you will find the name of the primitive and its short form, if one exists, followed by the type of each input it requires, in italics. We indicate on the same line whether the primitive is a command, an operation, or an infix operation.

Below this, we provide general information about the primitive and examples of its use.

When you use any primitive or procedure that refers to the turtle, Logo switches to the graphics screen. If you are in a MODE without graphics (3, 6 & 7), you will receive a LOGO message:

NOT POSSIBLE IN THIS MODE

Relatively few examples are given as the operation of Turtle graphics commands is straightforward, and many examples are given in the first sections of the manual. Where a number is required as an input to a Turtle graphics command it can always be a real number (eg 34.13456). The most important idea to get hold of when approaching Turtle Graphics for the first time is that you control the turtle's "state", its heading (the direction in which it is pointing), its position, whether or not it is drawing a line, the colour of the line, and the colour of the background. You also control its domain, and decide whether or not it can ever disappear from view.

BACK (BK) *n*   command

Moves the Turtle *n* steps back (ie in the opposite direction to its heading). Its heading does not change. Note that BACK 0, (with PENDOWN) displays a single dot at the turtle's current position, without moving the turtle. Logo will protest if *n* is greater than 32767.9999 or less than −32767.9999. See FORWARD.

BG   operation

BG outputs the current BackGround colour. See also SETBG, PC and SETPC. PRINT BG will print on the screen a number

---

corresponding to the current BackGround Colour. The numbers of the BackGround colours correspond to the "logical colours" available in the different modes. These logical colours can be changed using VDU commands. See BBC User Guide (pp 160-180 , 377-390), or the SETPAL procedure set out below.
The default colours in each graphics MODE are:

BG No.	**MODES 0, 4**
0	BLACK
1	WHITE

	MODES 1, 5
0	BLACK
1	RED
2	YELLOW
3	WHITE

	MODE 2
0	BLACK
1	RED
2	GREEN
3	YELLOW
4	BLUE
5	MAGENTA
6	CYAN
7	WHITE
8	FLASHING BLACK/WHITE
9	FL. RED/CYAN
10	FL. GREEN/MAGENTA
11	FL. YELLOW/BLUE
12	FL. BLUE/YELLOW
13	FL. MAGENTA/GREEN
14	FL. CYAN/RED
15	FL. WHITE/BLACK

The numbers of the MODE 2 colours are the ones you use if you want to change the default colours in other MODES. The procedure SETPAL is helpful in this regard

```
TO SETPAL :A :B
MAKE "A (SE 19 :A :B 0 0 0)
VDU :A
END
```

where :A is the number of the colour you want to be changed (in the MODE you are in) and :B is the number of

the colour you want (from MODE 2). For example, in MODES 5 or 1, SETPAL 1 4 would remove RED from your list of available colours and replace it with BLUE

You can use the procedure MAP (see toolkit) to change all the colours available, as follows:

```
MAP "SETPAL [0 1 2 3] [2 4 5 6]
```
would change your available colours in MODES 5 or 1 from Black, Red Yellow and White to Green, Blue, Magenta and Cyan.

CLEAN   command

Wipes the graphics screen, without changing the turtle's state (see POS & HEADING), or the displayed text in the text window.

CS   command

Wipes the graphics screen and returns the turtle to position [0 0] in the centre of the screen, and its heading to 0, pointing straight up the screen. CS **does not affect** displayed text, background and foreground colours, or WRAP, FENCE or WINDOW (see below). CS also acts as the switch from text screen to graphics screen. If the entire screen is dedicated to text, that text will be lost in the switch to graphics screen.

DOT *x y*   command

DOT does not exist as a primitive in Logotron Logo. A procedure to leave a dot at a specified position, coordinates *x y*, can easily be created as follows:

```
TO DOT :x :y
MAKE "P POS
PU HT SETPOS SE :X :Y
FD 0
SETPOS :P PD ST
END
```

FENCE   command

Limits the turtle's movements to the screen boundaries. After using FENCE, Logo will not allow you to move the

turtle beyond the limits of the screen. See also WRAP and WINDOW.

>   FENCE BK 1000
>   BK DOES NOT LIKE 1000 AS INPUT

FORWARD (FD) *n*   command

Moves the turtle *n* steps forward (ie in the direction it is HEADING). Like BACK 0, FORWARD 0 (with PENDOWN) displays a single dot at the turtle's current POSition without moving the turtle. Logo will protest if *n* is greater than 32767.9999 or less than –32767.9999. See BACK.

HEADING   operation

Outputs the turtle's heading, a number greater than or equal to 0 and less than 360. This is the same as the system used for compass bearings, where North (conventionally at the top of a map) represents a heading of 0 degrees, East (towards the right) is 90 degrees, South 180, and West 270. When you enter Logo, the turtle's HEADING is 0.

HOME   command

Moves the turtle to the centre of the screen and sets its HEADING to 0. It does not CLEAN the screen. If in PENDOWN, the turtle draws a line from its current position to HOME.

HT   command

Stands for Hide Turtle, which makes the turtle invisible, although it can still draw. This command speeds up the turtle's movements.

LEFT (LT) *n*   command

Turns the turtle left (counterclockwise) *n* degrees. It is an error if *n* is greater than 32767.9999 or less than –32767.9999. For example LEFT 45 or LT 45 turns the turtle 45 degrees to the left. LT –45 would turn the turtle 45 degrees to the right. See RIGHT.

PC operation

PC outputs a number corresponding to the current PenColour. PC is 1 on entering Logo. These numbers correspond to the logical colours available in different modes. See the section above on BG for details of these, and how they can be changed using SETPAL. See also SETPC.

PD command

Lowers the turtle's pen, so it draws a line when it moves. See PU. Stands for Pen Down.

PE command

The turtle erases any previously drawn lines it passes over. For example:

```
TO VANISH
REPEAT 4 [FD 250 RT 90]
PE
REPEAT 4 [FD 250 RT 90]
END
```
PD reverses the PE command.

PU command

Lifts the turtle's pen so that no line is drawn when it moves. For example: PU FD 50. See also PD and SETPOS.

POS operation

Outputs the turtle's position as a list of coordinates *[x y]*. When you enter Logo POS is [0 0].

```
RT 90 FD 50
PR POS
50 0
```

RIGHT (RT) *n* command

Turns the turtle right (clockwise) *n* degrees. It is an error if *n* is greater than 32767.9999 or less than –32767.9999. For example RIGHT 45 or RT 45 turns the turtle 45 degrees to the right. RT –45 turns the turtle 45 degrees to the left.

SCRUNCH operation

Outputs the aspect ratio x:y, the ratio of the length of a horizontal step to the length of a vertical step. See SETSCRUNCH. The default value is 1.0.

SETBG *n* command

Sets the BackGround colour to the colour n. See BG for the table of values and how to change them. **Warning:** SETBG wipes out any graphics currently displayed on the screen. This is not a bug in the software, it is a feature of the BBC Micro.

SETH *n* command

Sets the HEADING of the turtle to *n* degrees if *n* >= 0 and < 360; to REMAINDER *n* 360 if *n* > 359; to 360 − *n* if *n* < 0 and > −360; and to 360 + (REMAINDER *n* 360) if *n* < −359.

.SETNIB *n* command

This allows one to achieve spectacular graphics effects, but it needs to be handled with care, as it is making use of the BBC micro's operating system. It corresponds to the BBC BASIC command PLOT, and variations can be found on p.319 of the BBC User Guide. The value of *n* corresponds to the value of K. Try

    .SETNIB 85 FD 200 RT 90 FD 200

    .SETNIB 21 FD 300

These two values, 85 and 21, will be most useful in normal use. The dot in front of .SETNIB is there to warn you that Logo cannot protect you from setting incorrect values as the input to .SETNIB, as it is controlled by the BBC operating system, and not by Logo.

SETPC *n* command

Sets the turtle's PenColour to the colour *n*. See BG for the table of values and how to change them. The number of colours available depends on the MODE you are in.

SETPOS *[x y]*   command

Given a list of two numbers (the x and y coordinates, see
XCOR & YCOR), SETPOS moves the turtle to that POSITION.
If PENDOWN, the turtle leaves a trace. For example
compare:

>   PD SETPOS [–189 79]
>   PU SETPOS [123 –90]

If you wish to input a value derived from a variable to
SETPOS, use

>   SETPOS SE :X :Y

>   SETPOS [:X :Y]

will prompt the Logo message

>   SETPOS DOESN'T LIKE [:X :Y] AS INPUT

SETSCRUNCH *n*   command

Sets the aspect ration x:y = *n*, where x is a turtle step
along the horizontal axis and y is a turtle step on the vertical
axis. Try different values for *n* between .5 and 1.5 on
squares and circles. For example

>   SETSCRUNCH .5
>   REPEAT 360 [FD 10 RT 1]

See also SCRUNCH.

SETX *n*   command

Moves the turtle to point *n* on the x-coordinate (XCOR)
leaving the y-coordinate (YCOR) unchanged. If PD the
turtle leaves a horizontal trace.

SETY *n*   command

Moves the turtle to point *n* on the y-coordinate (YCOR)
leaving the x-coordinate (XCOR) unchanged. If PD the
turtle leaves a vertical trace.

ST   command

Makes the turtle visible. See HT. Stands for Show Turtle.

WINDOW   command

Enables the turtle to move outside the screen area, treating the screen as a window, viewing a small rectangle at the centre of its circular field. The TURTLE can move up to 32767 steps in any direction from the centre. See FENCE and WRAP. If you wish to change the size of the graphics window, you must be in WINDOW. Here is a procedure, which will split the screen vertically, giving you space on the right for text and on the left for graphics.

```
TO SPLITSCREEN
CS
VDU [26 12 28 0 31 20 0 24 "700 "0 "1278 "1000 29 "989 "500]
CS WINDOW
END
```

In order to understand fully how this works, it is essential to study the VDU commands in Section 34 (p.377) of the BBC User Guide. See also Section 28 of this manual. Use the command TS to reverse any windowing. This restores all windows (text and graphics) to their default values.

WRAP   command

Makes the turtle's field WRAP around the edges of the screen, When the turtle crosses a screen boundary, it immediately reappears on the opposite side. Topologists will tell you that WRAP maps the turtle's field onto a torus. When you first enter Logo, the turtle's field is in WRAP.

XCOR   operation

Outputs the x-coordinate of the current position of the turtle. SETX XCOR + 20 moves the turtle 20 steps to the right. Draws a line unless you first enter the command PU.

YCOR   operation

Returns the y-coordinate of the current position of the turtle. SETY YCOR – 20 moves the turtle 20 steps down the screen.

SECTION TWENTY-TWO – WORDS AND LISTS

There are two types of object in Logo: words and lists. We discussed them in Section 20 of the manual. Here we look at some primitives to put them together and take them apart. These can be quite confusing. So before defining them one by one, here is a chart, which may help you tell one from another. If you want to try them out, use SHOW instead of PRINT, as PRINT strips off the outer brackets, while SHOW leaves them in place. For example:

```
SHOW FIRST [[JOHN MARY] [SUSAN GEORGE]]
[JOHN MARY]

PRINT FIRST [[JOHN MARY] [SUSAN GEORGE]]
JOHN MARY
```

Operation	Input	Output
FIRST	"JOHN	J
BF	"JOHN	OHN
FIRST	[MARY JOHN BILL]	MARY
BF	[MARY JOHN BILL]	[JOHN BILL]
FIRST	[[MARY JOHN] BILL]	[MARY JOHN]
BF	[[MARY JOHN] BILL]	[BILL]
FIRST	[ ]	Logo Message∗
BF	[ ]	Logo Message∗
First	"	Logo Message+
BF	"	Logo Message+

∗ FIRST/BF DOESN'T LIKE [ ] AS INPUT
+ FIRST/BF DOESN'T LIKE AS INPUT

Operation	Input1	Input2	Output
FPUT	"LOGO	"TIME	Logo Message∗
LIST	"LOGO	"TIME	[LOGO TIME]
LPUT	"LOGO	"TIME	Logo Message∗
SE	"LOGO	"TIME	[LOGO TIME]
WORD	"LOGO	"TIME	LOGOTIME
FPUT	[AND MORE]	[TO COME]	[[AND MORE] TO COME]
LIST	[AND MORE]	[TO COME]	[[AND MORE] [TO COME]]
LPUT	[AND MORE]	[TO COME]	[TO COME [AND MORE]]
SE	[AND MORE]	[TO COME]	[AND MORE TO COME]
WORD	[AND MORE]	[TO COME]	Logo Message+

∗ LPUT/FPUT DOESN'T LIKE TIME AS INPUT
+ WORD DOESN'T LIKE [TO COME] AS INPUT

Note: The empty word " , shown as isolated quotes in a Logo line, merely appears as a blank space in Logo messages.

105

In the case of the empty list, [ ], the delimiters
are shown (see ∗ and + above).

ASCII *character*   operation

Outputs the ASCII code (decimal *n*) for *character*. There is a
full list of ASCII codes in the BBC User Guide. If the input
word contains more than one character, ASCII returns the
code for its first character. See CHAR. For example:

```
TO SECRETCODE :WD
IF EMPTY? :WD [OP "]
OP WORD CODE FIRST :WD SECRETCODE BF :WD
END

TO CODE :LET
MAKE "NUM (ASCII :LET) + 3
IF :NUM > ASCII "Z [MAKE "NUM :NUM – 26]
OP CHAR :NUM
END

PR SECRETCODE "CAT
FDW

PR SECRETCODE "CRAYON
FUDBRQ
```

The next task is to write a procedure which will translate
secret code back into English.

BF *object*   operation

Outputs all but the first element of *object*. BF " or BF [ ] are
impossibilities and prompt a Logo Message. For example:

```
SHOW BF [BRIAN J. SMITH]
[J. SMITH]

SHOW BF "DOGS
OGS

SHOW BF [DOGS]
[ ]

SHOW BF 3456
456

SHOW BF [ ]
BF DOESN'T LIKE [ ] AS INPUT
```

The following procedure strips a word or list, one element at a time:

```
TO TRIANGLE :MESSAGE
IF EMPTY? :MESSAGE [STOP]
PRINT :MESSAGE
TRIANGLE BF :MESSAGE
END
```

```
TRIANGLE "LOGO
LOGO
OGO
GO
O
```

```
TRIANGLE [HOW NOW BROWN COW]
HOW NOW BROWN COW
NOW BROWN COW
BROWN COW
COW
```

But watch out for this

```
MAKE "PAIR [3 5]
```

```
IF 5 = BF :PAIR [PR "OKAY][PR [TRY AGAIN]
TRY AGAIN
```

```
IF 5 = FIRST BF :PAIR [PR "OKAY][PR [TRY AGAIN]
OKAY
```

The reason is that BF list outputs another list. In order to compare the second element of the list :PAIR to 5, you need to use the additional operation FIRST. BF *word* outputs another word.

BL *object*   operation

Outputs all but the last element of the specified *object* (word or list). It is the mirror image of BF.

```
SHOW BL [I YOU SHE WE]
[I YOU SHE]
```

```
SHOW BL "FLOWER
FLOWE
```

```
SHOW BL "
BL DOESN'T LIKE  AS INPUT
```

See BF, which also refuses to accept the empty word or list as input.

CHAR *n* operation

Outputs the character whose ASCII code is *n* (see ASCII), an integer from 0 through 255.

COUNT *object* operation

Outputs the number of elements in the specified *object* (word or list):

```
PR COUNT [A B C D E F G]
7

PR COUNT 2345
4

PR COUNT "PEACOCK
7

MAKE "PERSON [HEAD ARMS LEGS BODY]
PRINT COUNT :PERSON
4

TO PICK :INFO
OP ITEM (1 + RANDOM COUNT :INFO) :INFO
END

PR PICK :PERSON
LEGS
```

EMPTY? *object* operation

Outputs TRUE if the Logo *object* is empty, otherwise outputs FALSE:

```
MAKE "A [ ]
PR EMPTY? :A
TRUE

MAKE "A "CABBAGES
PR EMPTY? :A
FALSE

PR EMPTY? BF [UNICORNS]
TRUE
```

Where a *list* has only one element, as [UNICORN] above, BF *list* is the empty *list* [ ]. The following procedure

matches animal sounds to animals:

```
TO TALK :ANIMALS :SOUNDS
IF OR EMPTY? :SOUNDS EMPTY? :ANIMALS [PR [THAT'S ALL FOR NOW]
STOP]
PR SE FIRST :ANIMALS FIRST :SOUNDS
TALK BF :ANIMALS BF :SOUNDS
END

TALK [DOGS MOSQUITOS WOLVES MONKEYS] [BARK ZZZZZZZZZZZ HOWL
CHATTER]

DOGS BARK
MOSQUITOS ZZZZZZZZZZ
WOLVES HOWL
MONKEYS CHATTER
THAT'S ALL FOR NOW
```

FIRST *object*   operation

Outputs the first element of a word or list. FIRST of a word
is a character; FIRST of a list may be a word or a list:

```
SHOW FIRST "HAPPY.NEW.YEAR
H

SHOW FIRST [HAPPY NEW YEAR]
HAPPY

SHOW FIRST [[H A P P Y] [N E W] [Y E A R]]
[H A P P Y]
```

The Primitive ITEM already exists, but the following
procedure shows how you could create it from other Logo
procedures.

```
TO ITEM1 :N :OBJECT
IF :N = 1 [OP FIRST :OBJECT]
OP ITEM1 :N – 1 BF :OBJECT
END

PR ITEM1 3 [CUP PUT TUB BUD]
TUB
```

This illustrates an important point: beginning with LIST,
FIRST and BF, you can create most other Logo
procedures. You will find other instances of this truth in the
toolkit.

FPUT *object list*   operation

Stands for First PUT. Outputs a new list, formed by putting
the specified *object* at the beginning of the specified *list*.
See the chart at the beginning of this section comparing
FPUT with other operations that combine words and lists.
Example:

```
TO REV :LIST
IF EMPTY? :LIST [OP []]
OP FPUT LAST :LIST REV BL :LIST
END

PRINT REV :ALPHABET
Z Y X W V U ...........
```

Note: FPUT requires a list as its second input. It will not
bind two words together.

ITEM *n list*   operation

Outputs the nth ITEM of a specified *list*. See FIRST above.

```
TO ITEMISE :OBJECT :LIST :COUNTER
IF NOT MEMBER? :OBJECT :LIST [PR (SE :OBJECT [IS NOT AN ITEM OF]
:LIST) STOP]
IF EQUAL? :OBJECT ITEM :COUNTER :LIST (OP (SE :OBJECT [IS ITEM]
:COUNTER "OF :LIST)]
OP ITEMISE :OBJECT :LIST :COUNTER + 1
END

TO PICKRANDOM :L
OP ITEM 1 + (RANDOM COUNT :L) :L
END
```

LAST *object*   operation

Outputs the LAST element of a word or list. LAST of a word
is a character; LAST of a list may be a word or a list.

```
SHOW LAST "HAPPY.NEW.YEAR
R

SHOW LAST [HAPPY NEW YEAR]
YEAR

SHOW LAST [[H A P P Y] [N E W] [Y E A R]]
[Y E A R]
```

LAST or FIRST of the empty word or empty list is an impossibility and prompts a Logo message. Example

> PR LAST "
> LAST DOESN'T LIKE AS INPUT

LIST *object1 object2*   operation

Outputs a list, whose elements are *object1* and *object2*.

> MAKE "LINE LIST [ONE TWO] [THREE FOUR]
>
> SHOW :LINE
> [[ONE TWO][THREE FOUR]]

Note 1: Where the input *objects* consist of two lists, the lists remain as separate lists. See SE, which would combine the two lists into a single list.

Note 2: LIST can take only two inputs. It is unlike SE or WORD, which can be enclosed in parentheses and given any number of inputs. If one wishes to use LIST with more than two inputs one must repeat the operation:

LIST *object1* LIST *object2 object3*

For Example

> MAKE "LINE LIST [ONE TWO] LIST [THREE FOUR] [FIVE SIX]
>
> SHOW :LINE
> [[ONE TWO] [[THREE FOUR] [FIVE SIX]]]

Whereas:

> MAKE "LINE (SE [ONE TWO] [THREE FOUR] [FIVE SIX])
>
> SHOW :LINE
> [ONE TWO THREE FOUR FIVE SIX]

LIST? *object*   operation

Outputs TRUE if the *object* is a list, otherwise FALSE.

> PR LIST? [6 ABC LOGO]
> TRUE
>
> PR LIST? 6
> FALSE
>
> PR LIST? BF [CATS]
> TRUE

LPUT *object list*   operation

Stands for Last PUT. Outputs a new list which places the
object at the end of the *list*. LPUT is the exact counterpart
of FPUT. Like FPUT, it must have a *list* as its second input.

        PR LPUT "STONE [WOOD IRON BRICK]
        WOOD IRON BRICK STONE

But

        PR LPUT "D "STONE
        LPUT DOESN'T LIKE STONE AS INPUT

MEMBER? *object list*   operation

Outputs TRUE if the *object* is an element of the *list*,
otherwise FALSE.

        PR MEMBER? "A [B 20 A ORANGE]
        TRUE

        MAKE "FRUIT [APPLES PEARS PLUMS RASPBERRIES]

        IF MEMBER? "PLUMS :FRUIT [PR "OKAY]
        OKAY

        IF MEMBER? "PLUM :FRUIT
        FALSE

        PR MEMBER? "L [AB [L] Y Z]
        FALSE

        PR MEMBER? "D "ODD
        MEMBER? DOESNT LIKE "ODD AS INPUT

        TO VOWEL :LETTER
        OP MEMBER? :LETTER [A E I O U]
        END

        IF VOWEL "I [PR [THAT'S A VOWEL]]
        THAT'S A VOWEL

NUMBER? *object*   operation

Outputs TRUE if the object is a number; otherwise FALSE.

        PR NUMBER? 3
        TRUE

        IF NUMBER? [7] [PR "OKAY] [PR [TRY AGAIN]
        TRY AGAIN

In this case the test failed because [7] is a list not a number.

```
IF NUMBER? FIRST [7] [PR "OKAY] [PR [TRY AGAIN]
OKAY
```

Here, FIRST extracts 7 from the list.

```
TO READNUMBER
MAKE "CHECKNUM RL
IF NUMBER? :CHECKNUM [OP :CHECKNUM] [PR [THAT'S NOT A NUMBER
     TRY AGAIN] READNUMBER]
END
```

Readnumber is often a useful variation on READLIST when constructing games and quizzes, to force entry of a number rather than a string of letters.

SE *object1 object2*
(SE *object1 object2...objectn***)**   operation

Outputs a list composed of all the *objects* in the input. See LIST for difference. SE is extremely useful in Logotron Logo for providing inputs to VDU commands. For example:

```
TO TEXTCOL :N
MAKE "N (SE 19 1 :N 0 0 0)
VDU :N
END

TO PAPERCOL :X
MAKE "X (SE 19 0 :X 0 0 0)
VDU :X
END

TO TEXT'N'PAPER :N :X
TEXTCOL :N
PAPERCOL :X
END
```

See also SETPOS for a similar use of SE.

WORD *word1 word2*
(WORD *word1 word2 ... wordn***)**   operation

WORD outputs a word consisting of its inputs, which must themselves be words. WORD will not accept a list as input.

For example:

```
PR WORD "ASTON "ISH
ASTONISH
```

```
PR (WORD "ASTON "ISH "ING)
ASTONISHING
```

```
TO WEEK :DAYS
IF EMPTY? :DAYS [STOP]
PR WORD FIRST :DAYS "DAY
WEEK BF :DAYS
END
```

```
MAKE "DAYS [MON TUES WEDNES THURS FRI SATUR]
```

```
WEEK :DAYS
MONDAY
TUESDAY
WEDNESDAY
THURSDAY
FRIDAY
SATURDAY
```

WORD? *object*   operation

Outputs TRUE if the object is a word, otherwise FALSE.

```
PR WORD? "123ABC
TRUE
```

```
PR WORD? BF "123ABC
TRUE
```

```
PR WORD? FIRST [ 123ABC ]
TRUE
```

```
PR WORD? 123
TRUE
```

```
PR WORD? [ XYZ ]
FALSE
```

```
PR WORD? [ ]
FALSE
```

SECTION TWENTY-THREE – VARIABLES

Any Logo word can be used to name a variable. A variable names a thing. The THING which is named is a Logo object and can be a word or a list. The THING is also referred to as the value of the variable. See Section Twenty.

MAKE *name object*   command

Creates the variable *name* and gives it the value *object*. Once the variable has been created, you can recall its value with dots (:). See also THING. For example:

```
MAKE "NATIONS [FRANCE GERMANY RUSSIA ENGLAND CHINA]

PRINT :NATIONS
FRANCE GERMANY RUSSIA ENGLAND CHINA

PRINT "NATIONS
NATIONS

MAKE "X 8
PR :X
8

MAKE "X COUNT :NATIONS
PR :X
5
```

Variables created using MAKE are global in their scope. See the Summary of Logo Grammar. For example:

```
MAKE "SIZE 200

TO SQ
REPEAT 4 [FD :SIZE RT 90]
END

TO SQUARES
SQ
PR (SE [THE VALUE OF "SIZE IS NOW] :SIZE)
MAKE "SIZE :SIZE / 2
PU SETPOS (SE XCOR – 300 YCOR) PD
SQ
PR (SE [THE VALUE OF "SIZE IS NOW] :SIZE)
END
```

Compare this with

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

TO SQUARES
SQUARE 200
PU SETPOS (SE XCOR – 300 YCOR) PD
SQUARE 300
PR (SE [THE VALUE OF "SIDE IS NOW] :SIDE)
END
```

In the second case, unless there is already a global variable
called SIDE, you will get a Logo Message, saying

```
SIDE HAS NO VALUE IN SQUARES
```

In other words, it is local to SQUARE, and has no value
anywhere else.

You may sometimes want to create a global variable using
MAKE, but nevertheless keep it local to one procedure.
Logotron Logo does not have LOCAL as a primitive, but we
do make it possible to ERASE a variable:

```
TO SQUARE
MAKE "SIZE 300
REPEAT 4 [FD :SIZE RT 90]
ER ["SIZE]
END
```

But

```
TO SQUARE :SIZE
MAKE "SIZE 100
REPEAT 4 [FD :SIZE RT 90]
END
SQUARE 50
PRINT :SIZE
SIZE HAS NO VALUE
```

It would also be possible to

```
ED ["SIZE]
PO ["SIZE]
SAVE "VAR ["SIZE]
```

This is an outstanding new feature of Logotron Logo.

NAME? *object*   operation

Outputs TRUE if the *object* has a value, otherwise FALSE.
NAME? can be used to test for the existence of a global
variable (See MAKE above).

    PR NAME? "SIZE
    TRUE

    PR NAME? "SIDE
    FALSE

This last example assumes the user still has the previous
examples in the workspace.

    MAKE "FRUIT "APPLE
    PR NAME? "FRUIT
    TRUE

THING *name*   operation

Outputs the value associated with *name.* THING "X is the
same as :X.

Note: THING :X is legal, but ::X is not.

    MAKE "MARY "HAPPY
    MAKE "HAPPY [A BIRTHDAY PARTY]

    PR THING "MARY
    HAPPY

    PRINT THING :MARY
    A BIRTHDAY PARTY

    TO INC :X
    MAKE :X 1 + THING :X
    END

    MAKE "TOTAL 7
    PR :TOTAL
    7

    INC "TOTAL
    PR :TOTAL
    8

    INC "TOTAL
    PR :TOTAL
    9

SECTION TWENTY-FOUR – ARITHMETIC

Logo uses both integers and real numbers. For example, 6 is an integer; −6 is an integer, whereas 3.435 is a real number. Some arithmetic operations, however, always return integers: INT, RANDOM, ROUND, QUOT.

Logo provides primitive procedures for you to add, subtract, multiply and divide numbers. You can find sines, cosines and square roots. Other procedures, to raise a number to a power, for example, can easily be created, (**see** below).

Real numbers with more than six digits are converted to standard form (exponential or scientific notation):

2E+6 means 2 times 10 to the power of 6, or 2,000,000

2.59E−2 means 2.59 times 10 to the power of −2 or 0.0259. Logo truncates a a real number in standard form if it contains more than 9 digits.

For example, 2718281828459 045 is converted to 2.71828183E+12.

The difference between infix and prefix operations is discussed in detail in the Section 20 of this manual. But to repeat briefly: Logotron Logo allows both infix and prefix arithmetical operations. Infix operations take precendence over prefix operations appearing to their left in a Logo line. If there are expressions to the left of the infix operation requiring prior evaluation, enclose them in parentheses.

	Infix	Prefix
Division	/	
		QUOT
		REMAINDER
Multiplication	∗	
		PROD
Subtraction	−	
Addition	+	SUM
Equality/inequality	=	EQUAL?
	<	
	>	

Prefix Operations

ARCTAN *n*   operation

Outputs the value in degrees of the arctangent of *n.*

```
PR ARCTAN 1
45
```

Arcsines and arcosines may easily be derived as follows:

```
TO ARCSINE :X
OP ARCTAN :X / (SQRT 1 − :X ∗ :X)
END

TO ARCCOSINE :X
OP ARCTAN (SQRT 1 − :X ∗ :X) / :X
END
```

COS *n*   operation

Outputs the Cosine of *n* degrees.

```
PR COS 60
.5
```

EQUAL? *object1 object2*   operation

Outputs TRUE if *object1* and *object2* are identical numbers,
identical words, or identical lists; otherwise outputs FALSE.
Equivalent to the infix operation: *object1* = *object2*. EQUAL?
is both a logical and an arithmetic operation, as it accepts
boolean values (TRUE and FALSE) as well as numbers as
inputs.

```
PR EQUAL? FIRST "ORANGE FIRST "OGLE
TRUE

PR EQUAL? ITEM 3 [SHERGAR SECRETO TROY TULYAR MORSTON] ITEM
2 [ROME TROY CARTHAGE]
TRUE

PR EQUAL? (7 ∗ 3) (2 ∗ 11)
FALSE

IF EQUAL? 27 (COUNT "TEA) ∗ (SQRT 81) [PRINT TRUE] [PRINT FALSE]
TRUE
PR EQUAL? 10E2 1000
TRUE
```

INT *n*   operation

Outputs the next whole number (INTeger) below real number
n by removing any decimal fraction. Note watch out for the
operation INT on negative numbers.

```
PR INT 5.2129
5
PR INT 5.6
5
PR INT –5.5
–6
```

PROD *a b*
(PROD *a b . . . n***)**   operation

Outputs the product of the inputs. It is equivalent to the infix
operation ∗. If PROD has more than two inputs, parentheses
must enclose PROD and its inputs.

```
PR PROD 5 5
25
PR (PROD 5 5 2)
50
TO CUBE :X
OP (PROD :X :X :X)
END
PR CUBE 2
8
```

QUOT *a b*   operation

Outputs the integer quotient obtained by dividing *b* into *a*,
and removing any decimal fraction.

```
PR QUOT 6 3
2
PR QUOT 5.3 1.7
3
PR 5.3 / 1.7
3.11764706
PR 6.3 / 1.7
3.70588235
```

```
PR QUOT 6.3 1.7
3

PR QUOT 25 0
QUOT DOESN'T LIKE 0 AS INPUT
```

RANDOM *n*   operation

Outputs a RANDOM integer between 0 and (*n* − 1). The input *n* must be an integer. RANDOM 6 could output 0 1 2 3 4 or 5.

```
TO DICE
OUTPUT 1 + RANDOM 6
END

IF DICE = 6 [START]
```

Note: RANDOM 6 + 1 would return any of 0 1 2 3 4 5 6, because the + takes precedence over RANDOM and would be evaluated first, giving RANDOM 7. The alternative to the form shown would be OUTPUT (RANDOM 6) + 1

REMAINDER *a b*   operation

Outputs the integer REMAINDER when *a* is divided by *b*. If the REMAINDER is a real number it is ROUNDED to output the nearest integer.

```
PR REMAINDER 16.7 4.3
4

PR REMAINDER 16.7 4.1
0

PR REMAINDER 16.7 4.2
4

PR REMAINDER 88 9
7
```

Here is a procedure to test whether one **integer** is exactly divisible by another:

```
TO DIVISOR? :INT1 :INT2
OP 0 = REMAINDER :INT1 :INT2
END

PR DIVISOR? 6 4
FALSE
```

ROUND *n*   operation

Outputs the nearest integer to *n*. Compare these examples
with INT:

    PR ROUND 5.219
    5

    PR ROUND 5.5
    6

    PR ROUND –5.3
    –5

SIN *n*   operation

Outputs the sine of *n* degrees.

    PR SIN 30
    .5

SQRT *n*   operation

Outputs the square root of *n*. *n* must be positive.

    PR SQRT 4567
    67.5795827

See the procedure DIST in the toolkit.

SUM *n*
(SUM *a b* . . . *n***)**   operation

Outputs the sum of the inputs. It is equivalent to the infix
operation +. If SUM has more than two inputs, parentheses
must appear around SUM and its inputs.

    PR (SUM 5 2 3)
    10

    PR SUM (4 / 2) (7 ∗ 3)
    23

TAN *n*   operation

Outputs the tangent of *n* degrees.

Infix Operations

Although it is not necessary, it is good practice to leave a space both before and after an infix operator. Take especial care with the minus sign (–).

---

a + b   operation

The plus sign (+) outputs the sum of two inputs *a* and *b*.

```
PR 5 + 2
7

PR −5 + 2
−3
```

---

a − b   operation

The minus sign (–) outputs the difference between the inputs *a* and *b*.

```
PR 7 − 2
5

PR −7 − 3
−10

PR −7 − −2
−5
```

Note: Be careful of the minus sign (–). The same character is used to represent three different things.:

1. Part of a number to indicate that it is negative, as in –3, with no space between the sign and the digit.

2. A procedure taking one input, called unary minus, which outputs the additive inverse of its input, as in –XCOR or –:NUM.

3. An infix procedure of two inputs, as here, outputting the difference between its first input and its second.

---

a ∗ *b*   operation

The multiplication sign (∗) outputs the product of two inputs *a* and *b*.

```
PR 6 ∗ 2
12

PR 6 ∗ –2
–12

PR 2 + 3 ∗ 4
24

PR (2 + 3) ∗ 4
20
```

---

a / *b*   operation

The division sign (/) outputs the dividend of *a* and *b* (*a* divided by *b*). The output is a real number with no remainder.

```
PR 5 / 2.5
2
PR 5.3 / 2.1
2.52380952

PR 6 / 0
/ DOESNT LIKE 0 AS INPUT
```

---

a < *b*   operation

The "less-than" sign (<) outputs TRUE if *a* is less than *b*. The inputs must be numbers.

```
PR 8 < 9
TRUE

PR 9 < 9
FALSE
```

If one wished to use the less-than and greater-than signs to sort words by their initial letters, one might use the construction:

```
IF (ASCII :LET1) < (ASCII :LET2) [ . . . .
```

---

a > b operation

The "greater-than" sign (>) outputs TRUE if *a* is greater than *b*. Both inputs must be numbers:

```
PR 6.789 > 6.788
TRUE
```

---

object1 = object2 operation

The equals sign (=) outputs TRUE if *object1* is equal to *object2*. Unlike > and <, the inputs need not be numbers, they can be any logo objects (words or lists). It is equivalent in every way to the prefix operation EQUAL?

```
PR 80 = 100 − 20
TRUE

PR 80 = 100 −20
FALSE
YOU DON'T SAY WHAT TO DO WITH −20
```

Further Operations For Toolkit

```
TO MAX :A :B
OP IF :A > :B [:A] [:B]
END

TO MIN :A :B
OF IF :A < :B [:A] [:B]
END

TO BALRAN :DEL
OP :DEL − RANDOM (2 ∗ :DEL + 1)
END

TO FACTORIAL :NUMBER
IF :NUMBER = 1 [OUTPUT :NUMBER]
OUTPUT :NUMBER ∗ (FACTORIAL
     :NUMBER − 1)
END

TO EXPONENT :NUMBER :POWER
IF :POWER = 0 [OUTPUT 1]
OUTPUT :NUMBER ∗ (EXPONENT
     :NUMBER :POWER − 1)
END
```

```
TO XOR :PRED1 :PRED2
OP NOT EQUAL? :PRED1 :PRED2
END

TO MULT :NUM :LIST
IF EMPTY? :LIST [OP []]
OP FPUT :NUM * FIRST :LIST
     MULT :NUM BF :LIST
END

TO LINEMULT :LIST1 :LIST2
IF EMPTY? :LIST1 [OP []]
OP FPUT MULT FIRST :LIST1 :LIST2
     LINEMULT BF :LIST1 :LIST2
END

TO ABS :NUM
IF :NUM > 0 [OP :NUM]
 [OP –:NUM]
END

TO PI
OP 3.1415927
END
```

Computers do not do arithmetic quite as we do, especially when they are dealing with real numbers, with decimal fractions. They are constantly truncating, and rounding numbers. Computer arithmetic is accurate only to a limited degree of precision. For example, Type

```
PR EQUAL? (SIN 30) * (SIN 30) + (COS 30) * (COS 30) 1
```

This will sometimes return TRUE and sometimes FALSE. This is true of BASIC as well as Logo, but we think it is important to recognise the fact, and not to regard it as a disgraceful or shameful bug. It is the nature of finite computer arithmetic.

SECTION TWENTY-FIVE – EDITING

There are two ways of defining procedures, as we saw in the introduction. At toplevel, working directly into the workspace, you can define procedures using the primitive TO name. (for a third method, which is occasionally needed, see SETWRITE in Section 28 dealing with the Outside World).

It is often more convenient to define procedures inside the EDITOR, which reserves a special area of the computer's memory, the edit buffer, for building and changing procedures. In the EDITOR you can move the cursor keys about, write words onto the screen, and press the return key, without bringing any procedure to life. Logo does not execute instructions when in the EDITOR.

You will also find the keys work differently when you are in the EDITOR. The arrow keys, for example, can be used to drive the cursor around the screen. Because the EDITOR has its own reserved part of the computer's memory, the edit buffer, it can also be used as a temporary storage place for procedures, when changing MODE, for example.

EDALL   command

This command moves everything currently in your workspace into the EDIT buffer, which holds 1,500 characters. If there is an overflow, EDALL moves in as much as it can, before reporting OUT OF SPACE. You can then SAVE onto a disk if necessary. For a detailed description of the editor, see below.

EDIT (ED)
EDIT (ED) *name*
EDIT (ED) *namelist*   command

Typing EDIT takes you into the EDITOR. The screen changes, and you see a line across the bottom of the screen.

<div align="center">

LOGO EDITOR
^C <exit> ESC <abort>

</div>

The cursor (now a flashing square) will be at the top left-hand corner, ready for you to begin typing. If you have

not used the EDITOR before, there will be nothing written on the screen. If you have used the EDITOR previously, you will see exactly the same words as were there when you left it.

If you use the form

EDIT *name*

you clear the edit buffer of whatever was stored there. For example,

ED "SQUARE

and, provided you had previously defined SQUARE or loaded into your workspace from disk or cassette, Logo will have the definition waiting for you in the EDITOR. If there is no procedure SQUARE in the workspace, you will read:

TO SQUARE
END

at the top of the screen, all ready for you to type the definition. You will find it very easy to change lines, insert words, and generally mess around with your procedures.

If you have made a global variable, say SHIPS and then want to add some more, you can type

EDIT ["SHIPS]

Logo will take you into the Editor and you might find:

MAKE "SHIPS [CANBERRA QE2 SHEFFIELD BELGRANO]

Remember, Logo knew you were referring to a variable name and not a procedure name, because ["SHIPS] was written with quotes (") inside a list.

The four ARROW keys move the cursor one space in each direction, up, down, right and left. SHIFT ARROW keys (ie press the SHIFT key and the ARROW key simultaneously) move the cursor to the top of the page, to the bottom of the page, to the beginning of the line, and to the end of the line.

The delete key deletes the character to the left of the cursor, just as it does at top level, but there are other function keys, which wipe out whole lines, move the cursor to the end of the buffer, or to the beginning, if you have more than one page in the EDITOR at the time.

The most novel feature of the Logotron Logo's EDITOR is the FIND & REPLACE function. Press the F9 key, the rightermost of the red keys at the top of the keyboard. The words

FIND:
REP:

will appear at the bottom of the screen below the line of dashes, with the cursor positioned in front of the FIND. You type in the word or string of words you want replaced, press the RETURN key, and the cursor moves down a line. Type in the word or words you want to insert instead of the first string. Press RETURN and the substitution will be made. For example, imagine the procedure SQUARE was in the EDITOR.

```
TO SQUARE :SIDE
REPEAT 4 [FD 200 RT 90]
END
```

You realise that you should have written FD :SIDE instead of FD 200. So you press F9.

```
FIND: FD 200
REP: FD :SIDE
```

Press RETURN and you will see

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

If there had been further instances of the same substitution required, you would have executed the first, and then pressed the COPY key, followed by F9.
The F8 key performs a simple FIND without a REP line. In other respects it works in the same way.

Note. Before pressing the F8 or F9 keys, be sure the cursor is at the beginning of the text to be searched as Logo scans forward from the cursor.

From the example given above, it might seem hardly worthwhile. The real benefit can be seen when you have a large number of linked procedures in the EDITOR simultaneously, and want to change variable names, for example, or the name of a procedure which is called a number of times.

SUMMARY OF EDITOR COMMANDS

LEFT ARROW (←)   moves the cursor one character position to the left;

RIGHT ARROW (→)   moves the cursor one character position to the right;

DOWN ARROW (↓)   moves the cursor down one line;

UP ARROW (↑)   moves the cursor up one line;

SHIFT LEFT ARROW (←)   moves the cursor to the beginning of the current line;

SHIFT RIGHT ARROW (→)   moves the cursor to the end of the current line;

SHIFT DOWN ARROW (↓)   moves the cursor to the end of the current page;

SHIFT UP ARROW (↑)   moves the cursor to the beginning of the current page;

DELETE   erases the character to the left of the cursor;

F0   erases the character at the cursor position;

F1   erases text from the cursor position to the end of the current line and places it in the kill buffer;

F2   inserts the text from the kill
                                buffer at the cursor position;

F3   moves cursor to next page;

F4   moves cursor to previous page;

F5   scrolls screen to place current
                                line at its mid point;

F6   moves cursor to the beginning
                                of Edit Buffer;

F7   moves cursor to the end of
                                the Edit Buffer;

F8   FIND

COPY F8   repeats the last FIND command;

F9   FIND and REPLACE

COPY F9   repeats the last FIND
                                and REPLACE command;

CTRL C   Exit from Editor, executing
                                changes in the workspace;

ESCAPE   Aborts editing, leaves workspace
                                unchanged, and contents
                                of Editor intact

To load a number of procedures into the edit buffer together, type:

    ED [PROC1 PROC2 PROC3 . . . PRONn]

This is particularly useful when you want to move procedures temporarily into the edit buffer when changing modes. For example:

    PR MODE

    4

    ED [POLYSPI SQUARE CIRCLE FACE FOREVER MOVETO FLOWER]

    Ctrl C

    ERALL

    SETMODE 2

    ED

    Ctrl C

    PR MODE

    2

This is an example. You could not try it out without first writing a collection of procedures to switch in and out of the EDITOR.

However, the EDITOR isn't only for procedures. You can also write and edit variable assignment statements:

    MAKE "CITIES [ROME PARIS BONN]

    MAKE "SQS [1 4 9 16 25 36 49 64]

or Logo commands

    REPEAT 10 [FD 100 RT 90 FD 100 LT 90]

When you leave the EDITOR by typing Ctrl C, Logo reads each line in the Edit buffer. If it is part of a procedure, enclosed between TO . . . END, it will be incorporated into the Logo workspace, and Logo will tell you which procedures have been DEFINED. These may include some which already existed but have been modified in the EDITOR.

Logo will also tell you which variable names have been defined or redefined in the EDITOR.

If it comes across a list of instructions in the edit buffer, it will simply RUN them, as if they had been typed in at top level.

If for any reason, you do not want to incorporate the contents of the edit buffer into your workspace, you can leave the EDITOR by pressing the ESC key, which aborts the edit without making any changes to your workspace.

END   special word

END is necessary, when you are using TO, to tell Logo that you are done defining the procedure. It must be on a line by itself. However, if you leave off END when writing a procedure in the EDITOR, Logo will put it on for you. END is neither a command nor an operation. It is really just a signal to Logo that it has finished executing a procedure. If you just type END, Logo complains:

    YOU ARE AT TOP LEVEL

TO *name input1 input2 . . . inputn*   command

TO tells Logo that you are defining a procedure called name, with inputs (if any) as indicated. At toplevel, the prompt changes from a question mark (?) to a greater-than sign (>) to remind you that you are defining a procedure. This special prompt will remain on the screen, every time you press the RETURN key, until you have written a line with the special word END on its own, to tell Logo that the procedure is complete.

If you have already defined a procedure, say REVERSE, and then type: TO REVERSE :L   You will receive a Logo message:

    REVERSE IS ALREADY DEFINED
You get the same message if you try to define a procedure with the name of a primitive, for example:

    TO FORWARD

SECTION TWENTY-SIX – FLOW OF CONTROL

Logo is an interpreted language, which means that it reads procedures line by line, following the instructions as it meets them. If a procedure contains a call to a subprocedure, Logo reads the lines of the subprocedure before continuing to finish the superprocedure.

For example, let us imagine a call to SUPERPROCEDURE. This is executed as follows: **line1 line2 line3 ...** , where it meets a call to SUBPROCEDURE, which it executes as follows: **line1 line2 END**, it then returns to line3 of the SUPERPROCEDURE and continues **... line3 line4 END**

The phrase Flow of Control refers to the order in which Logo follows instructions. There are times when you want to interfere with Logo's normal way of doing things. There are several ways of doing this. Let's look at them:

Conditional instructions tell Logo to execute a particular instruction or list of instructions, **IF** a particular condition is **TRUE**.

REPEAT instructions tell Logo to execute a list of instructions 2 or more times.

The STOP, OP or TOPLEVEL instructions tell Logo to STOP the current procedure without continuing to the END. STOP and OP only halt the procedure in which they appear. If that procedure is a subprocedure, the superprocedure continues to run. TOPLEVEL stops the superprocedure too, and returns control to the user.

IF *pred instructionlist1 instructionlist2*
                          command or operation

IF is a very powerful Logo primitive. The first input it needs is a predicate. A predicate is a statement which can be tested by the computer to be either TRUE or FALSE. Here are some Logo primitives which can be tested in this way:

EMPTY? *object*
EQUAL? *object1 object2*
LIST? *object*
MEMBER? *object list*
NAME? *word*

NUMBER? *object*
WORD? *object*
num1 > num2
num1 < num2
object1 = object2

You can create your own procedures which perform similar functions. We have shown in this manual (Section 24), VOWEL? *letter* and DIVISOR? *num1 num2*.

All that is needed is a procedure which will OUTPUT either TRUE or FALSE to IF.

If the *predicate* outputs TRUE, then Logo executes the list of instructions enclosed in square brackets immediately following, *[Instructionlist1]*. If the *predicate* outputs FALSE, Logo looks to see if there is a second list of instructions on the line. If there is, it executes *[Instructionlist2]*.

If there is no second list, then Logo passes on to the next line. There are many examples of the use of IF in the sample procedures in this manual. It is hard to write procedures without the word, especially when it comes to stopping recursive procedures.

IF you are familiar with other computer programming languages, **THEN** you will recognise the **IF . . . THEN . . . ELSE** construction, **ELSE** this paragraph may help you understand it.

IF is one of only two Logo primitive procedures, which sometimes work as a command, and sometimes as an operation. As we saw in the introduction, a procedure is an operation if it outputs a value, a command if it does not output a value. Look at these three versions of the DECIDE procedure. In every case the procedure returns the answer YES or NO, at random. Every parent needs one.

If as a command:

```
TO DECIDE
IF 0 = RANDOM 2 [OP "YES]
OP "NO
END

TO DECIDE
IF 0 = RANDOM 2 [OP "YES] [OP "NO]
END
```

If as an operation:

```
TO DECIDE
OP IF 0 = RANDOM 2 ["YES] ["NO]
END

PR DECIDE
NO

REPEAT 5 [PR DECIDE]
YES
YES
NO
YES
NO
```

OP *object*   command

Unlike most other commands, OP cannot be used at
top level (STOP and TOPLEVEL are the other examples), only
inside a procedure. This is not really surprising because the
effect of all three procedures is to interrupt a running
procedure, and they could therefore have no meaning at top
level. OP makes *object* the OutPut of the running
procedure, and returns control to the caller.

Note: OP itself is a command, but the procedure
containing it is an operation because the procedure outputs a
value to another procedure, which could be a primitive, typed
at top level, or it could be another running procedure.

This can be clearly seen in the DECIDE procedures shown in
the discussion of IF above.

If one simply typed DECIDE, A Logo message would
complain:

```
YOU DON'T SAY WHAT TO DO WITH YES
```
DECIDE has to OP to a command like PRINT. Once a
value has been OP, the procedure has done its work,
and the flow of control goes back, either to top level, or to a
superprocedure, which called the subprocedure containing
OP. For this reason, it has a similar effect to STOP.

OP can be used to return any Logo object as its value.

Examples:

```
TO FRANCE
OP [PARIS IS THE CAPITAL OF FRANCE. FRANCOIS MITTERRAND IS THE
PRESIDENT OF THE REPUBLIC]
END

PR FRANCE
PARIS IS THE CAPITAL OF FRANCE. FRANCOIS MITTERRAND IS THE
PRESIDENT OF THE REPUBLIC

TO CUBE :N
OP (PROD :N :N :N)
END

CUBE 6
YOU DONT SAY WHAT TO DO WITH 216
```

REPEAT *n instructionlist*   command

Repeats a list of instructions *n* times; *n* must be a positive
integer, so a decimal fraction is truncated to an integer. Note
n can be output from an operation:

```
REPEAT 4 [FD 100 RT 90]

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

TO LETTERSQUARE
REPEAT COUNT :ALPHABET [PRINT :ALPHABET]
END
```

RUN *instructionlist*   command or operation

With a Logo list as its input, RUN executes the list as if it
were a Logo line. If the instruction list is an operation, then
RUN behaves as an operation and outputs whatever has
been output to it by the instructionlist.

```
TO CALCULATOR
PRINT RUN READLIST
PRINT [ ]
CALCULATOR
END
```

```
CALCULATOR
2 + 3
5

17.5 * 3
52.5

42 = 8 * 7
FALSE
```

The ESCAPE key takes you out of the otherwise endless loop.

```
RUN [PR [GOOD MORNING]
GOOD MORNING
```

The procedure WHILE runs a list of instructions while a specified condition is true.

```
TO WHILE :CONDITION :INSTRUCTIONLIST
IF NOT RUN CONDITION [STOP]
RUN :INSTRUCTIONLIST
WHILE :CONDITION :INSRUCTIONLIST
END
```

```
WHILE [YCOR < 100] [FD 25 PR YCOR]
25
50
75
100
```

You will sometimes find a use for the procedure FOREVER (see BOXES and PHRASEBOOK in **Section 18** of the manual).

```
TO FOREVER :INSTRUCTIONLIST
RUN :INSTRUCTUCTIONLIST
FOREVER :INSTRUCTIONLIST
END
```

```
FOREVER [FD 10 RT 1]
```

sends the turtle around an endless circle.

```
FOREVER [PR RUN RL PR []]
```

is the equivalent of the CALCULATOR procedure.

STOP   command

Stops the procedure running and returns control to the caller. The command STOP works only from inside a procedure, and has no effect on other procedures. It is often used as

the brake on recursive procedures (see also OP and TOPLEVEL):

```
TO POLYSPI :SIDE :ANGLE
IF :SIDE < 10 [STOP]
FD :SIDE RT :ANGLE
POLYSPI :SIDE – 5 :ANGLE
END

TO COUNTDOWN :NUM
PR :NUM
IF :NUM = 0 [PR [BLAST OFF!] STOP]
COUNTDOWN :NUM – 1
END

COUNTDOWN 4
4
3
2
1
0
BLAST OFF!
```

TOPLEVEL   command

TOPLEVEL works in exactly the same way as STOP, except that it returns control to top level, and does not just stop the currently running procedure, but also any superprocedures. Once Logo meets TOPLEVEL in a program, it returns control to the user. The next thing you see is the ? prompt, and Logo is waiting for you to do something. For example, compare:

```
TO LOOKFOR1 :X :L
IF EMPTY? :L [STOP]
IF :X = FIRST :L [MAKE "OBJ :L STOP]
LOOKFOR1 :X BF :L
END

LOOKFOR1 "Z "AZBCZXY
PR :OBJ
ZXY
```

with:
```
       LOOKFOR2 :X :L
       IF EMPTY? :L [STOP]
       IF :X = FIRST :L [MAKE "OBJ :L TOPLEVEL]
       LOOKFOR2 :X BF :L
       END
```

```
LOOKFOR2 "Z "AZBCZXY
PR :OBJ
ZBCZXY
```

The difference is that once LOOKFOR2 has found a condition which satisfies the predicate in the third line (IF :X = FIRST :L) it makes the required assignment and returns control to the user. LOOKFOR1 hunts on for another instance.

TRACE *n*   command

The TRACE command allows you to look right into the workings of a Logo procedure, showing you how the values of variables change, and the inputs and outputs of each operation. For example, enter the following procedure:

```
TO REPLACE :NEW :OLD :OBJ
IF EMPTY? :OBJ [IF WORD? :OBJ [OP " ] [OP []]]
IF LIST? :OBJ [IF EQUAL? :OLD FIRST :OBJ [OP FPUT :NEW REPLACE
 NEW :OLD BF :OBJ] [OP FPUT FIRST :OBJ REPLACE :NEW :OLD BF :OBJ]]
IF WORD? :OBJ [IF EQUAL? :OLD FIRST :OBJ [OP WORD :NEW REPLACE
 NEW :OLD BF :OBJ] [OP WORD FIRST :OBJ REPLACE :NEW :OLD BF
 :OBJ]]
END
```

Then type

```
TRACE
PRINT REPLACE "E "A "BEAR
PRINT REPLACE "PLUM "APPLE [ORANGE PEAR APPLE LEMON]
```

The TRACE command is activated by typing TRACE. After that, all Logo operations are traced on the screen. Typing TRACE a second time switches it off and returns Logo to normal working. This is very useful when debugging a procedure. It is also helpful when learning about how recursion works on lists.

WAIT *n*   command

Tells Logo to WAIT for *n* 60ths of a second before executing the next instruction. Example:

```
TO SLOWFD :DIST
REPEAT :DIST [FD 1 WAIT 2]
END

SLOWFD 80
```

SECTION TWENTY-SEVEN – AND, OR, NOT

In the description of the primitive IF in the last Section, we discussed the concept of a predicate, which had to output either TRUE or FALSE to IF, in order to establish whether Logo should next execute *[instructionlist1]* or *[instructionlist2]*. We are now going to discuss three predicates, whose inputs, like their outputs, must be the boolean values TRUE or FALSE. They are AND, OR and NOT. They are used to combine predicates into logical expressions.

This is similar to the way arithmetic operations are combined to form arithmetic expressions. Just as arithmetic operations have numbers as both their inputs and their outputs, so logical operations have only TRUE or FALSE as their inputs and outputs. In fact, TRUE and FALSE behave very like numbers, and can easily be represented as 1 and 0, ON and OFF, and so on. This has made them especially valuable to people who make computers or write computer programs.

The Logo primitives, which, with their inputs, form predicates are listed in Section 26 dealing with Flow of Control in the description of IF. It is slightly artificial to confine our discussion of Logical Operations to these few primitives. Most Logo operations are "logical", and one needs to consider the relationship of AND, OR and NOT to the Arithmetical Operations and the Flow of Control.

AND *pred1 pred2*
(AND *pred1 pred2 pred3 ....predn***)**   operation

Requires two or more inputs. AND outputs TRUE if all its inputs are TRUE, otherwise FALSE. If there are more than 2 inputs, AND and its inputs must be enclosed in parentheses. For example:

```
PRINT AND TRUE TRUE
TRUE

PRINT AND TRUE FALSE
FALSE

PRINT 4 * 4 = 16
TRUE

PRINT 12 / 3 = 4
TRUE
```

```
PRINT AND (4 * 4 = 16) (12 / 3 = 4)
TRUE
```

Note: the parentheses in the last example are unneccessary, but make for easier reading.

```
TO EVEN? :OBJ
OP AND NUMBER? :OBJ CHECK :OBJ
END

TO CHECK :OBJ
OP 0 = REMAINDER :OBJ 2
END
```

This pair of procedures could be used to check whether a Logo object was an even number. It is quite a useful convention to label procedures which can be used as predicates by ending their names with a ?. They all output TRUE or FALSE (eg EMPTY? MEMBER? NUMBER? etc).

Some teachers believe children find ALL.OF *list* an easier concept than AND with more than two inputs. It is easy to construct:

```
TO ALL.OF? :L
IF RUN SE SE [ ( AND ] :L " ) [OP TRUE] [OP FALSE]
END

PR ALL.OF? [TRUE (4 / 2 = 2) (5 * 5 = 25) FIRST "GREEN = "G]
TRUE
```

FALSE   *boolean value*

FALSE, like TRUE, is a *boolean value*. It is neither a command nor an operation, but is the input/output of logical operations. It behaves rather like a number, and therefore does not require QUOTES in front of it. As with numbers QUOTES are optional.

```
PR FALSE = "FALSE
TRUE
```

NOT *predicate*   operation

Outputs TRUE if the *predicate* is FALSE and FALSE if the *predicate* is TRUE

```
PR NOT EQUAL? "A "Z
TRUE
```

```
PR NOT 4 > 3
FALSE

PR NOT VOWEL? "A
FALSE
```

OR *pred1 pred2*
(OR *pred1 pred2 pred3 . . . predn***)**   operation

Outputs true if any of the *predicates* is true. If OR has more than two inputs, parentheses must enclose the primitive and its inputs. Examples:

```
PRINT OR TRUE TRUE
TRUE

PR OR TRUE FALSE
TRUE

PR OR FALSE FALSE
FALSE

PRINT OR 4 * 4 = 16 2 = 3
TRUE

PRINT (OR FIRST "BEE = "B 3 * 5 = 12 8 = 4 / 3)
TRUE
```

Again, some teachers would prefer to formulate (OR *pred1 pred2 ...predn*) as ANY.OF *list*. This is easily constructed:

```
TO ANY.OF? :LIST
IF EMPTY? :LIST [OP FALSE]
IF FIRST :LIST [OP TRUE]
ANY.OF BF :LIST
END
```

TRUE   *boolean value*

TRUE is a *boolean value*, acting as input/output for logical operations. See FALSE.

SECTION TWENTY-EIGHT – OUTSIDE WORLD

When your BBC micro is running Logo it is really a different "machine" from when it is running BASIC. When people used the word machine before computers were invented, they meant a physical object, made of metal, wood or plastic, usually with moving parts, which had physically measurable inputs and outputs.

Computer folk use the word machine in a rather different way. They are talking about a combination of physical objects, which you can see and touch (hardware), and non-physical objects, which exist, but cannot be seen or touched (software). This machine needs some physical inputs like energy and may produce physical outputs like marks on paper, but the most important inputs and outputs are abstract, words and numbers. You know what the number five is; you can write it down, but you cannot see the thing which is represented by the name 5.

The Logo machine consists of a series of instructions, which are written in a 16K ROM chip inside your computer, interacting with the hardware of the BBC Micro. As you use it the Logo machine grows, occupying the workspace with new procedures (instructions) written by you. It also grows inside your head, as you grow more fluent in the language.

The "**Outside World**", which we are talking about in this section heading, consists partly of other programs (sets of instructions) living in the BBC Micro (the operating system and the disk filing system), and partly of the hardware itself, the computer, the monitor, disc drives or cassette recorders, printers, floor turtles, robots, sprite boards, or any other devices you invent or buy.

In fact, we have already looked at part of the outside world in the first section dealing with turtle graphics. Turtle graphics are a good way of learning to program because your programs are working on a microworld, consisting of the monitor screen, with a single inhabitant, the turtle, and you can quite easily understand and control its state and behaviour (changing states).

∗suffix command

As well as being an infix arithmetic operation, the asterisk (∗) is the prefix to a whole series of commands which allow you to control the BBC operating system. These are summarised on pages 416 & 417 of the BBC User Guide.

Perhaps the most important for you are

∗LOGO, which takes you from BASIC or, say, WORDWISE into Logo, and ∗B., which takes you out of Logo into BASIC. ∗W. takes you into WORDWISE. ∗LOGO cannot be abbreviated to ∗L. as the computer would confuse the command with ∗LOAD. ∗CAT displays a catalogue or directory of files stored on your disk or cassette. Watch our for ∗.1, which in BASIC would also give you details of the contents of a disk on a particular drive. You must type ∗.\1. The backslash ensures that Logo simply passes the number to the operating system.

You will find many uses for other STAR commands as you explore the system. Here are two examples:

Teachers of young children can often help them better if they know exactly what they are doing at the keyboard. The following procedure (which existed as a primitive on the original mainframe Logo systems) allows you to capture every keystroke of a Logo session as a file which can be printed out afterwards:

```
TO DRIBBLE :FILENAME
∗SPOOL :FILENAME
END

TO UNDRIBBLE
∗SPOOL
END
```

The following procedure is a useful one because you cannot SAVE a file to disk with the same name as an existing file. This is a nuisance if you simply want to replace an old version with a new version. I use

```
TO REFILE :FILENAME :PROCNAMELIST
∗DELETE :FILENAME
SAVE :FILENAME :PROCNAMELIST
END
```

One omission from the list of primitive procedures is SAVEPICT, together with its partner LOADPICT. They were left out, partly because of lack of space in the 16k ROM, and partly because the BBC operating system makes it very easy for you to write your own SAVEPICT and LOADPICT. See p.392 in the BBC User Guide and p.460ff in the Advanced User Guide for the BBC Micro. The procedures which follow will do the job for you. Please note, if you make a picture in one mode, SAVE it, and LOAD it in another MODE, you will get some strange but often interesting results. One way of keeping track of the original MODE is to include in the name of :PICT, say TREE5 or HOUSE4, to show that the TREE was originally drawn in MODE 5 and HOUSE in MODE 4.

```
TO SAVEPICT :PICT
IF (OR EQUAL? MODE 3 EQUAL? MODE 6 EQUAL MODE 7)
     [PR [SAVEPICT MAY NOT BE USED IN THIS MODE]]
IF (OR EQUAL? MODE 0 EQUAL? MODE 1 EQUAL? MODE 2)
     [*SAVE :PICT 3000 8000]
IF OR EQUAL? MODE 4 EQUAL? MODE 5
     [*SAVE :PICT 5800 8000]
END

TO LOADPICT :PICT
IF (OR EQUAL? MODE 3 EQUAL? MODE 6 EQUAL MODE 7)
     [PR [LOADPICT MAY NOT BE USED IN THIS MODE]]
HT
IF (OR EQUAL? MODE 0 EQUAL? MODE 1 EQUAL? MODE 2)
     [*LOAD :PICT 3000]
IF OR EQUAL? MODE 4 EQUAL? MODE 5
     [*LOAD :PICT 5800]
RUN :PICT
END
```

Warning: Some * commands can crash Logo. In particular, *COMPACT and *FORMAT, which load programs off disk into RAM, overwrite areas of memory used by Logo. This will not only wipe out procedures in the workspace, it actually crashes Logo. SAVE your Logo workspace to disk before using these commands. Logo can be restored by pressing CTRL BREAK. This resets the system, and you can load your workspace back from disk. This is not a bug in Logo; it is an

inherent limitation of the BBC Micro. It should not bother you unless you are not expecting it.

∗FX *num1 num2* . . . . almost deserves a section on its own, but again, it is better to go straight to the source, and consult the BBC User Guide (p. 418 ff.). ∗FX is not for novice users, and certainly not essential for enjoyable and productive use of the computer.

The rules for incorporating ∗**suffix** or ∗**FX** calls in Logo procedures are as follows. If they come at the beginning of a Logo line or list, the remaining contents of that line or list are passed to the operating system. If they are inserted in a line, the command and its inputs should be enclosed in parentheses. Inputs to ∗**FX** calls must be separated by commas, as they would be in BASIC, (not by spaces). Logo variables can be passed to these operating system calls. For example ∗DELETE :NAME or ∗FX :A.

CT   command

CT, for Clear Text, is the counterpart for the text screen of CLEAN, which clears the graphics screen (see Section 21). It clears the screen of text and places the cursor and prompt (?) in the top lefthand corner of the screen, ready for entering text. If you have a graphics screen displayed, CT clears the text only from the text window.

CURSOR   operation

CURSOR outputs the current position of the text cursor. As with the graphics operation POS, *position* is a pair of numbers. The first element of CURSOR gives the column number, 0-79 in MODE 0, 0-39 in the other 7 MODES. The second element gives the line number, 0-31 on the text screen, and 0-7 below the graphics screen, in MODES 0-6, 0-24 in MODE 7. See SETCURSOR.

ENVELOPE *num1 num2* . . . . *num14*   operation

See SOUND, below.

EOF?   operation

The predicate EOF? stands for End Of File, and is used in conjuction with SETREAD. It outputs TRUE if the end of file has been reached, FALSE otherwise. For example:

```
TO CHECK
IF EOF? [STOP]
END
```

KEY?   operation

Outputs TRUE it there is at least one key waiting to be read on the keyboard or any other device set by SETREAD, otherwise FALSE. The following procedure allows a child to drive the turtle around the screen using just two keys Q and P.

```
TO STEER
FD 2
IF KEY? [TURN RC]
WAIT 4
STEER
END

TO TURN :DIR
IF DIR = "Q [LT 30]
IF DIR = "R [RT 30]
END
```

LOAD *filename*   command

LOADs the contents of filename into the workspace, as if they were typed in directly from the keyboard. If filename doesn't exist, you will receive a Logo Message, advising you of the fact. As procedures are loaded in, Logo will confirm their presence by printing, for example:

```
SQUARE DEFINED
MOVETO DEFINED
MAP DEFINED
```

PRINT (PR) *object*   command

PRINT prints its inputs on the screen (or on any other device set by SETWRITE). When the object is a LIST, PRINT strips

off the outermost brackets. When the object is a word, the QUOTES (") are not printed.

Note. PRINT causes a linefeed to occur after the inputs have been printed. This makes PRINT [] a simple way of leaving a space between two lines of print. See SHOW and TYPE:

```
PRINT [GOOD MORNING]
GOOD MORNING

TO GREET :AGE
REPEAT :AGE [PRINT (SE [HAPPY BIRTHDAY TO YOU,] :AGE "TODAY!)]
END

GREET 5
HAPPY BIRTHDAY TO YOU, 5 TODAY!
HAPPY BIRTHDAY TO YOU, 5 TODAY!
HAPPY BIRTHDAY TO YOU, 5 TODAY!
HAPPY BIRTHDAY TO YOU, 5 TODAY!
HAPPY BIRTHDAY TO YOU, 5 TODAY!
```

RC command

Outputs the first character read from a device (set by SETREAD) or the keyboard. This character is not echoed on the screen. If no character is waiting to be read, READCHAR waits until the user types something. The command is frequently used to assign a value to a variable. For example:

```
TO RESPOND
PR [DO YOU WANT AN ICECREAM? TYPE YES/NO]
MAKE "ANSWER RC
IF :ANSWER = "Y [PR [GO TO THE ZOO AND LOOK FOR A POLAR
     COW]] [PR [THAT'S A PITY, THERE'S A FRESH STRAWBERY
     SUNDAE IN THE FRIDGE]]
END
```

See also RL. RC can also be used to build STEER (See KEY? above). RC stands for Read Character.

When Logo meets the backslash (\), it treats the next character literally, without reference to its meaning. This allows you to create a procedure name consisting of two words. TO BIG \ HOUSE, for example, would be accepted by Logo, whereas TO BIG HOUSE would not.

RL   command

Outputs the first line of words read from a device (set by
SETREAD) or the keyboard. This list is echoed on the screen.
If no list is waiting to be read, READLIST waits until the
user types something. If lists have already been typed, it
outputs the first line that has been typed but not read. The
command is frequently used to assign a value to a variable.
RL stands for Read List. For example:

```
MAKE "CAPITALS [[NIGERIA LAGOS][FRANCE PARIS] [INDIA DELHI]
[ARGENTINA BUENOS.AIRES] [ITALY ROME] [SPAIN MADRID]

TO QUIZ :CUE :LIST
WELCOME
MAKE "PAIR PICKRANDOM :LIST
PR (SE :CUE FIRST :PAIR ", WORD :CONTESTANT "?)
MAKE "ANSWER RL
IF :ANSWER = FIRST BF :PAIR [PR (SE [WELL DONE.] :CONTESTANT [DO
YOU WANT ANOTHER ONE?]) AGAIN]
PR (SE [BAD LUCK.] :CONTESTANT) [THE ANSWER IS ] FIRST BF :PAIR)
AGAIN
END

TO PICKRANDOM :LIST
OP ITEM 1 + (RANDOM COUNT :LIST) :LIST
END

TO AGAIN
PR [DO YOU WANT ANOTHER ONE?]
PR [ANSWER YES/NO]
MAKE "ANSWER RC
IF :ANSWER = "Y [QUIZ]
END

TO WELCOME
PR [HELLO. WHAT IS YOUR NAME?]
MAKE "CONTESTANT RL
PR []
PR (SE "WELL WORD :CONTESTANT ", [LET'S GET ON WITH THE GAME!])
END

QUIZ [WHAT IS THE CAPITAL OF] :CAPITALS
```

HELLO. WHAT IS YOUR NAME?
HARRY

WELL HARRY, LET'S GET ON WITH THE GAME!

WHAT IS THE CAPITAL OF NIGERIA?
NAIROBI

BAD LUCK HARRY. THE ANSWER IS LAGOS. DO YOU WANT ANOTHER
ONE?

ANSWER YES/NO

This structure could be used to create an infinite number of
quiz programs. Please also note a stylistic point. The
WELCOME and AGAIN subprocedures could be included in
the QUIZ procedure, but AGAIN would create very long lines,
which would be hard to read. If WELCOME was part of the
procedure, it would be repeated if :CONTESTANT wanted
another turn. If a section of a procedure can be turned into a
subprocedure, it is usually worth it.

SAVE *filename*
SAVE *filename procname*
SAVE *filename proclist*
SAVE *filename varnamelist*   command

SAVEs the contents of *filename* onto disk or cassette from
the workspace. If SAVE *filename* is used without specifying a
procname, *proclist* or *varnamelist*, the entire workspace will
be SAVED. For example, SAVE "SUSAN might be used to
save all the work in progress of a girl called Susan at the
end of her Logo session.

If Susan wanted only three of her procedures, she might
type SAVE "SUSAN [POLYSPI DRAGON WHILE]. On the
other hand, SAVE "SUSAN "HOUSE. saves a single
procedure called HOUSE. If *filename* already exists, you will
receive a Logo Message, advising you of the fact. (See ∗,
above, for the REPLACE procedure.)

In most other Logos, global variables could be saved only in association with the procedures in which they were used, or as part of a whole workspace. Logotron Logo allows you to save global variables, using the same syntax as EDIT, ERASE, and PO. For example.

> MAKE "LANGUAGES [ALGOL FORTRAN PASCAL ADA FORTH LOGO LISP
> POPLOG SNOBOL COBOL MAD MANIAC]

> SAVE ["LANGUAGES]

You could save a mixture of procedures and variables in a single list:

> SAVE [SQUARE "LANGUAGES WELCOME "SHIPS]

You would have saved two procedures, and two variables. If one of these names was incorrect, the whole SAVE would abort, and you would be asked to start again.

SETCURSOR *list*   command

SETCURSOR sets the cursor to *list*. As with the graphics command SETPOS, the first element of *list* gives the column number, 0-79 in MODE 0, 0-39 in the other 7 MODES. The second element gives the line number, 0-31 on the text screen, and 0-7 below the graphics screen, in MODES 0-6, 0-24 in MODE 7. For example, in MODE 4, textscreen SETCURSOR [20 15] will place the cursor right in the middle of the screen, and when you begin to type, that is where the text will appear. See CURSOR, above.

Note. If you want to write on the graphics screen, SETCURSOR does not work, and you have to use:

> VDU (SE 5 :X :Y)

where :X and :Y are the coordinates on the graphics screen at which you want to start writing, the text cursor and the graphics cursor are joined together.

VDU [4] reverses the effect of VDU [5] and returns control to the text cursor at its normal position below the Graphics Screen.

SETREAD *filename*
SETREAD [ ]   command

SETREAD is used for reading a file from disk or cassette.
Filename can be a program file or a data file created through
SETWRITE or DRIBBLE (see ∗*suffix* above). After the
command SETREAD is given, RC or RL read information
from filename. SETREAD [ ] closes the file being read. For
example:

```
SETREAD "CITIES
REPEAT 4 [PR [ ] PR RL] SETREAD []
DAKAR
DELHI
DJAKARTA
DUBLIN
```

You can only read from one file at a time, but you can open
a file for reading (SETREAD) and writing (SETWRITE) at the
same time.

SETWRITE *filename*
SETWRITE [ ]   command

Opens a file named *filename* and sends a copy of all
characters appearing on the screen to that file. SETWRITE [ ]
closes the file. Used together, SETWRITE and SETREAD can
be used to create databases for quiz games, address lists
and telephone numbers. They can also be used to define
procedures, which sometimes come as primitives in Logo
systems, but which could not be fitted into the Logotron 16k
ROM:

```
TO COPYDEF :NEWDEFINITION :OLDDEFINITION
MAKE "OLDDEFINITION TEXT :OLDDEFINITION
DEFINE :NEWDEFINITION (BF BF FIRST :OLDDEFINITION) (BF
:OLDDEFINITION)
END
```

SECTION TWENTY-EIGHT – OUTSIDE WORLD

```
TO DEFINE :NAME :INPUT :LIST
SETWRITE "PROG
PRINT (SE "TO :NAME :INPUT)
PR.OUT :LIST
PR "END
SETWRITE []
LOAD "PROG
ERASEFILE "PROG
END

TO PR.OUT :LIST
IF EMPTY? :LIST [STOP]
PR FIRST :LIST
PR.OUT BF :LIST
END

TO TEXT :NAME
SAVE "PROG :NAME
SETREAD "PROG
OP FPUT BF BF RL READLINE [ ]
END

TO READLINE :TEXT
MAKE "LINE RL
IF [END] = :LINE [ERASEFILE "PROG OP :TEXT]
OP READLINE LPUT :LINE :TEXT
END

TO ERASEFILE :FILENAME
∗DELETE :FILENAME
END


DEFINE "SPIRAL [[:SIZE :ANGLE] [FD :SIZE] [RT :ANGLE] [SPIRAL :SIZE +
15 :ANGLE]
```

would output

```
TO SPIRAL :SIZE :ANGLE
FD :SIZE
RT :ANGLE
SPIRAL :SIZE + 15 :ANGLE
END
```

While TEXT "SPIRAL would output:

```
[[ :SIZE ANGLE] [FD :SIZE] [RT :ANGLE] [SPIRAL :SIZE + 15 :ANGLE]]
```

Another useful pair of procedures, STARTUP and COPY allow you to create procedures on disk which can be run simply by LOADing them into the workspace. The procedures also demonstrate how SETREAD and SETWRITE can work together.

```
TO STARTUP :FILENAME :STARTUPFILENAME :STARTPROC
:SCREENMODE
SETWRITE :STARTUPFILENAME
PR "ERALL
(PR "SETMODE :SCREENMODE)
SETREAD :FILENAME
COPY
PR :STARTPROC
SETWRITE [ ]
END

TO COPY
IF EOF? [STOP]
PR RL
COPY
END
```

SHOW *object*   command

SHOWs *object* on the screen, followed by a carriage return. In fact, it works just like print, except that if *object* is a list, it does not strip away the outermost brackets. For example:

```
SHOW "A SHOW [A B C]
A
[A B C]

PRINT "A PRINT [A B C]
A
A B C

TYPE "A TYPE [A B C]
AA B C
```

These examples clearly show the differences between PRINT, TYPE and SHOW.

SOUND *num1 num2 num3 num4*   command

For a full discussion of the production of sound (music and
other noises) on the BBC micro, the user is referred to
pages 180-187 in the BBC User Guide. The inputs to sound
are numbers, which are not enclosed in a list. They appear
just as they do in BASIC (ie as described in the BBC User
Guide), but without commas between the numbers. eg

    SOUND 2 –3 121 99
num1 (0 to 3) is the channel number

num2 (0 to –15) is the amplitude
num 3 (0 to 255) controls pitch
num4 (0 to 255) controls duration

The command **ENVELOPE** *num1 num2* . . . *num14* gives the
user even greater control over the possible sound effects.
For example:

    TO TRIAL
    ENVELOPE 2 1 2 –2 2 10 20 10 1 0 0 –1 100 100
    SOUND 1 2 100 100
    END
The possibilities are endless, and well explained in the BBC
User Guide. One could easily develop a library of noises,
each a procedure which could be called into a program to
create appropriate sound effects. One could imagine:
TRUMPETS SIREN TAKEOFF BIRDSONG GUNFIRE etc.

You could also use RC to turn the computer into a keyboard
instrument.

TS   command

TS is the switch from the graphics screen to the
full text screen. It clears the screen of graphics, and places
the cursor and prompt (?) in the top lefthand corner of the
screen, ready for entering text. Anything (text or graphics)
displayed on the screen when this command is used will be
lost. TS is also the default value, and is displayed,
when you enter Logo. **TS is therefore used to
clear any windows you may have created (see WINDOW,**

Section 21 and VDU commands below). CS switches you to the normal graphics screen, with a seven-line text window below it.

TYPE *object*
(TYPE *object1 object2* . . . *objectn***)**   command

Prints *object* or *objects* to the screen or other device, but there is no linefeed once the printing is completed. If TYPE has two or more inputs, then TYPE and all the inputs must be enclosed in parentheses.

>   TYPE [A B C]
>   A B C
>
>   (TYPE [A B] [D E F] [G H I])
>   A B CD E FG H I

VDU *list*   command

VDU is another primitive (like ∗*suffix*) which gives you direct access to the BBC micro's own operating system. The VDU commands, as you might expect, all have to do with the monitor (or Visual Display Unit). You may have already met some VDU commands in this manual. You will certainly have met VDU commands if you have been using the BBC microcomputer for any length of time. They are described in detail is Chapter 34 of the BBC User Guide (pp 377-389).

From a Logo users point of view, the important thing to remember is that VDU takes a list of numbers as its input. If you want one of those numbers to be generated by evaluating a variable (:X :NUM :COL :LINE for example), you must create the list with the operation SE. For example:

>   VDU SE (19 :X :Y 0 0 0)

SE outputs a list to VDU with :X and :Y properly evaluated as numbers. If a number required as an input is greater than 255 and therefore occupies two bytes of memory, prefix it with quotes ("), and Logo will do the rest. For example "1278. Here is a list of some of the more useful VDU

commands with examples of how they might be used in Logo:

Printing

VDU [1] is used to send a character or string of characters to the printer only, and not to the screen. This is used to send control codes, which change the print style. For example, VDU [1 14] would instruct an Epson MX-80 to print double width characters. Of course, this assumes the printer is in place and turned on.

VDU [2] is used to turn on the printer (assuming you have a printer in place and plugged in) while VDU [3] turns it off. If children are turned off (as I am) by VDU commands, then here is an example of how a VDU command can be incorporated into a command which is more faithful to the spirit of the language. This way you get the power of the VDU commands, combined with the friendliness of Logo:

```
TO PRINTER :SWITCH
IF :SWITCH = "ON [VDU [ 2 ]]
IF :SWITCH = "OFF [VDU [ 3 ]]
END
PRINTER "ON
PRINTER "OFF
```

Labelling the Graphics Screen

VDU [5] is a wonderful tool which allows you to write text on the graphics screen. For example:

```
CS VDU [ 5 ] PRINT [TURTLE AT HOME]
```
The words TURTLE AT HOME will be printed in the middle of the screen, just below the turtle. VDU [4] returns the text cursor to its normal position, and disables VDU [5].

VDU [5] can be used for labelling pictures or diagrams. For example, to teach a child the screen coordinates, you could

use the following little program.

```
TO NAVIGATE
PU
MAKE "DRIVE RC
IF :DRIVE = "L [LT 10]
IF :DRIVE = "R [RT 10]
IF :DRIVE = "F [FD 20]
IF :DRIVE = "B [BK 20]
IF :DRIVE = "X [MAKE "X POS VDU [ 5 ] PR SE [X = ] ROUND XCOR
SETPOS :X]
IF :DRIVE = "Y [MAKE "Y POS VDU [ 5 ] PR SE [Y = ] ROUND YCOR
SETPOS :Y]
IF :DRIVE = "P [MAKE "P POS VDU [ 5 ] PR (SE [POS = ] ROUND
FIRST POS ROUND FIRST BF POS) SETPOS :P]
IF :DRIVE = "S [VDU [ 4 ] STOP]
NAVIGATE
END
```

Some of you will recognise a new use for the INSTANT
procedure. The best way to understand this procedure is to
type it in and use the controls, L R F B X Y P and S. It will
give you lots more ideas for using VDU [5]. **Warning:** It
doesn't work very well in MODES 2 and 5 as the characters
are too large.

Carriage Return

VDU [10] and **VDU [13]** together allow you to build a
carriage return into your programs. It is the equivalent of the
Logo command PRINT [ ].

Changing Colours

VDU [17] VDU [18] VDU [19] and VDU [20] all
affect the colours available in different modes. They are
discussed in some detail in Section 21 of this manual. See
especially the discussion of BACKGROUND.

Redefining Characters

VDU [23] allows you to redesign characters. It is not for the
novice.

Changing the Turtle's field

VDU [24] allows the user to change the size of the graphics window. This can only be done when the turtle field is a window. If it is in WRAP or FENCE, you cannot change the size of its field. Note if you do this, you also need to change the origin of the turtle, using VDU [29]:

    VDU (SE 29 :X :Y)

where :X and :Y are the coordinates of the new origin of the graphics cursor. Note: These coordinates can be mapped from the planning sheet on page 494 of the BBC User Guide. Once again, this is probably not one for the novice programmer. The procedure for doing it is given in Section 21 under WINDOW. It may be very useful in cases where children want to be able to see more of their commands as they run. This could be particularly true of children and teachers who have worked with the DART turtle graphics program.

Control Codes

The VDU drivers can be replaced at TOPLEVEL by using the CTRL characters, by simultaneously pressing the CTRL key and the key required by a given code. For example, CTRL U deletes the current line. CTRL E allows you to write on the graphics screen, just beneath the turtle.

A full list of these CTRL codes is provided on p. 378 of the BBC User Guide.

Important note: The only exception to the information you will find there, is that in the LOGO EDITOR, CTRL C is used to move you from the Editor, back into your workspace.

SECTION TWENTY-NINE – WORKSPACE

Part of the memory of the computer is reserved for the variables and procedures that you have written or loaded in from a disk or cassette. This is called your **workspace**.

The available space is measured in NODES, each of which is five bytes long. You can discover how many NODES are free at any one time by typing PR NODES. The number of NODES returned by this operation will range from 4,631 in MODE 7 with an empty workspace down to 755 in MODES 0, 1 & 2 with an empty workspace.

The difference is due to the memory taken up by the BBC in mapping the screen. This is as little as 1k bytes in MODE 7 and as much as 20k bytes in MODES 0, 1 and 2.

As procedures run, they use up memory space, sometimes temporarily, and sometimes permanently. The memory held temporarily is freed by a little procedure you cannot see called "the garbage collector", which bustles round clearing memory for reuse.

The garbage collector works automatically, but you can force an extra collection by typing RECYCLE. If you write lots of procedures and never SAVE any of them on disk or cassette, you will run out of memory, especially if you want to play with all the colour combinations available in MODE 2, for example.

Moving from one MODE to another with SETMODE adds an extra dimension to programming in Logo with the BBC Micro. We have made it as easy as possible by providing an EDITOR which acts as a temporary store for procedures, when you are switching from one MODE to another.

If you switch MODE without storing your procedures in the EDITOR, you may get a message saying LOGO NOT FRESH. This means there are procedures in the memory, which Logo does not want to destroy, and therefore cannot adjust the size of the workspace, as it must if it is to allocate more memory to screen management, when moving, say, from MODE 5 to MODE 2.

The maximum number of NODES available in each MODE is

as follows:

MODE 7	4,631
MODE 6	3,203
MODE 5	2,795
MODE 4	2,795
MODE 3	1,571
MODE 2	755
MODE 1	755
MODE 0	755

If you are lucky enough to have a second processor, you have the same number of NODES in each MODE, 5,500.

As we have seen, Logo will not let you move from a MODE with more available NODES to a MODE with fewer available NODES, unless you first erase all procedures from your workspace, with the command ERALL. Needless to say, you should be careful first to protect your procedures and variable names, either by SAVEing them to disk or cassette, or more conveniently, by typing EDALL and stashing them in the EDITOR, where you have room for 1,500 characters. Then leave the Editor by using the ESCAPE key before changing MODES.

Once you have changed MODES it is a simple matter to move your procedures and variable names back into the workspace, by first typing ED or EDIT without any procedure names following the command, and then CTRL C.

If you want to move from a MODE with fewer NODES to a MODE with more NODES, Logo does not protest, but it does not take up the extra available memory unless you have gone through the process of moving procedures into the EDITOR and ERasing them from the workspace.

The system was designed in this way to give you the maximum possible number of NODES to work with in all MODES. We stress this because it is different from other Logos you may have seen, both on other computers or on the BBC micro. But compare the number of procedures you can fit in your workspace, or in your editor, with those other Logos.

If you are familiar with other Logos, you will notice some changes. We have got rid of POPS, PONS, POTS, ERN, ERPS and ERNS. The main reason for this was that we believed they were confusing. There are now two basic commands PO (for Print Out) and ERASE (ER) and two operations, OPPS (for OutPut ProcedureS) and OPNS (for OutPut NameS). Combining the commands and operations, you can do everything the old systems did, and more. This change was conceived in conjunction with the new syntax for SAVEing, EDITing, Printing Out and ERASEing named variables, using ["name]

Here are the primitives you will need to manage your workspace, know your way around it, and generally keep it in good order. They include some which affect the Logo system itself. You can use them to access the computer memory directly. The more dangerous primitives start with a dot (.). Before using them, be sure to SAVE all your work on disk or cassette.

.CALL *n*   command

Transfers control to a machine language subroutine starting at address *n* (decimal). For advanced programmers only.

.CONTENTS   operation

Outputs a list of everything contained in the Logo workspace, including procedure and variable names, but NOT the names of primitive procedures. It will also include odds and ends of unfinished or cancelled business which the garbage collection cannot reach. You can get rid of this rubbish by saving your workspace to the Editor or to disk, and then reloading it into your workspace. This can sometimes help with space problems.

DEFINED? *word*   operation

Outputs TRUE if the specified word is the name of a procedure, otherwise outputs FALSE. If *word* is a primitive procedure, DEFINED? outputs FALSE (see PRIMITIVE?). For

example:

```
PR DEFINE? "SQUARE
TRUE
```

assuming you do have SQUARE in your workspace.

```
PR DEFINE? "RANDOM
FALSE
```

even though RANDOM is a primitive procedure. Compare with the operation of PRIMITIVE? on the same two words.

.DEPOSIT *n byte*   command

Writes the specified *byte* into machine address *n* (decimal). This command is provided for the use of experienced programmers. It is the equivalent of POKE in most dialects of BASIC (not BBC BASIC, see p409 of the BBC User Guide). See .EXAMINE below.

ERALL   command

ERrase ALL. Erases all the procedures and variables in your workspace. This command also frees up all the nodes available to you. Make sure that all the procedures you want to keep are stored safely in the EDITOR or on a disk or cassette before you use this file. Note especially the use of ERALL when changing MODE. It allows Logo to take full advantage of the extra workspace in a different MODE.

ERASE (ER) *name namelist*   command

Erases the named procedure(s) or variables from the workspace. Neither of the ERase commands (ERALL ERASE) affect procedures in the EDITOR or SAVEd on disk or cassette. The command works differently with procedures and variables. For example ER "TRIANGLE erases a procedure named TRIANGLE.

```
ER [TRIANGLE POLYGON SQUARE]
```

erases a list of procedures, TRIANGLE, POLYGON and SQUARE. ER ["TRIANGLE], on the other hand, erases a variable named TRIANGLE.

```
ER ["TRIANGLE "SQUARE SHAPES]
```

erases variables named TRIANGLE and SQUARE and a

procedure named SHAPES. If one of these variables or procedures cannot be found, Logo aborts the ERASE and returns you to toplevel. This protects you from erasing variables or procedures in error.

The variables you can erase in this way are the global variables, created by using the Logo primitive MAKE. For example:

```
MAKE "FLOWERS [ROSES PANSIES BUTTERCUPS DAISIES]

MAKE "TRIANGLE [REPEAT 3 [FD 200 RT 120]]

PRINT :FLOWERS
RUN :TRIANGLE
```

Now type

```
ER [FLOWERS TRIANGLE]
```

and try again. They are still there. Now type

```
ER ["FLOWERS "TRIANGLE]
```

and try again. Logo has wiped them out. Now type:

```
TO FLOWERS
PR [ROSES PANSIES BUTTERCUPS DAISIES]
END

TO TRIANGLE
REPEAT 3 [FD 200 RT 120]
END
```

First test these procedures by typing first FLOWERS, then TRIANGLE. Then type:

```
ER ["FLOWERS "TRIANGLE]
```

and they will, of course, survive. Type

```
ER [FLOWERS TRIANGLE]
```

and Logo forgets them. The same principle applies to the command EDIT (ED) where names can either apply to procedures or variables, also PO and SAVE.

Note 1: If you want to refer to a single variable name, it has to be a list with one element. For example ER ["FLOWER]. Logo would treat "FLOWER, without square brackets around it, as a procedure name. It also means you cannot have procedure names beginning with QUOTES. Logo would treat them as if they were variable names.

Note 2: ERNS and ERPS do not exist as primitives in Logotron Logo. It is easy to create them:

```
TO ERNS
ERASE OPNS
END

TO ERPS
ERASE OPPS
END
```

.EXAMINE *n*   operation

Outputs the contents of address n (decimal). See .DEPOSIT. Provided for experienced programmers.

MODE   operation

MODE outputs the number of the current MODE. When you enter Logo, you are in MODE 4. So when you switch on the computer, type:

```
PR MODE
4
```
See also SETMODE

NODES   operation

Outputs the number of free NODES. This is very useful when calculating precisely how to fit procedures into your workspace. Try running the following procedure:

```
TO FIB :NUM
IF :NUM < 3 [OP 1]
PR NODES
OP (FIB :NUM − 1) + (FIB :NUM − 2)
END
```

This is an extravagant way of generating Fibonacci numbers. This is the series which runs 1 1 2 3 5 8 13, in which each number is the sum of the preceding two. The output from FIB is the nth Fibonacci number. So FIB 3, outputs 2, FIB 7 outputs 13, and so on. In making this calculation, Logo uses up NODES. The line PR NODES has been included to show this actually happening. Try PR FIB 10, then RECYCLE PR NODES. The RECYCLE procedure frees up all the NODES which were temporarily used by FIB.

OPNS operation

Here is a brand new Logo primitive. It outputs all the names of
global variables contained in your workspace as a list. Using
OPNS (it stands for OutPut NameS), you can easily create ERNS
or PONS (see below under ERASE and PO).

OPPS operation

Here's another brand new Logo primitive, the counterpart of
OPNS (see above). It outputs the names of all procedures
contained in your workspace as a list. Using OPPS (it stands for
OutPut ProcedureS), you can easily create ERNS, ERPS, POPS or
POTS (see above and below under ERASE and PO).

PO *name namelist* command

Stands for Print Out the definitions of the named
procedure(s) and variables. You cannot PO Logo primitive
procedures.

```
PO "SQUARE
TO SQUARE :SIDE
REPEAT 4 [FD :SIDE LT 90]
END

PO [SPINCOIN DICE "FRUIT]
TO SPINCOIN
IF EQUAL? RANDOM 2 0 [OP "HEADS] [OP "TAILS]
END

TO DICE
OP 1 + RANDOM 6
END

MAKE "FRUIT [ORANGES BANANAS PINEAPPLE]
```

POALL command

Stands for Print Out ALL. Prints the definition of every
procedure and the name and value of every variable in the
workspace.

Note PONS POPS and **POTS** do not exist in Logotron Logo.
PONS and POPS can be created exactly like ERNS and ERPS
(see ER above). POTS is rather different, as it only prints out

the titles. It needs a combination of two small procedures:

```
TO POTS
POTS1 OPPS
END

TO POTS1 :L
IF EMPTY? :L [STOP]
(PR "TO FIRST :L)
POTS1 BF :L
END
```

PRIMITIVE? *word* operation

Outputs TRUE if the specified word is a primitive procedure, otherwise outputs FALSE. See DEFINED? above. For example:

```
PR PRIMITIVE? "RANDOM
TRUE

PR PRIMITIVE? "SQUARE
FALSE
```

PRIMITIVES command

Prints out a list of all the PRIMITIVE procedures. In order to inspect them, either print them out, using VDU [2] or slow them down by using CTRL N, then page through them using the SHIFT key.

RECYCLE command

Performs a garbage collection (see NODES above), freeing as many NODES as possible. Garbage collections happen automatically where necessary, but each one takes at least a second. Running RECYCLE before a time-dependent activity prevents the automatic garbage collector from slowing things down at an awkward moment. In fact the Logotron garbage collector is very efficient, and you will not often be aware of its existence. But if you ever see the turtle pause in the middle of a highly recursive procedure, it is likely to be the garbage collector at work.

SECTION THIRTY – LOGO MESSAGES

Sometimes, even when you know a good deal about Logo, the system will fail to understand you. When this happens, Logo sends you a message. We don't call them "error messages", because that suggests you have made an error, whereas it often means only that the poor old computer isn't as smart as you. Where we leave dots . . . . . ., Logo will fill in the name of the procedure and/or the word which is bothering it.

I DON'T KNOW HOW TO . . . .

You will see a lot of this when you begin. Often because you have mistyped the name of a procedure, or left off quotes (") or dots (:), so that Logo thinks a word is a procedure, when really it is a word to be printed, or a variable name to be evalutated.

YOU DON'T SAY WHAT TO DO WITH . . . .

Here's another common message. Usually you should have typed PRINT in front of a procedure which is an operation, which outputs a value. Operations need to be preceded by commands. Operations don't know what to do with the values they output, unless you tell them.

. . . . HAS NO VALUE

Logo has come across a word preceded by dots :, (:SIDE for example), and cannot find a value. Check that you have given it a value, either as an input to the procedure, or by creating a global variable with MAKE.

NOT ENOUGH INPUTS TO . . . .

Perhaps you have provided a procedure with only one input, where it expects two or more. For example:

    IF EQUAL? SQRT 16 [PRINT "OK]
would get the message:

    NOT ENOUGH INPUTS TO EQUAL?

. . . . DIDN'T OUTPUT TO . . . .

This may mean that you have tried to create a procedure to work as an operation, but forgotten to include the command OP to make sure it OUTPUTS a value to the calling procedure. Or that you have put two commands together.

For example

    PR FD 100

The turtle draws a line on the screen, but doesn't output to PRINT, which expects at least one input. So Logo sends a message:

    FD DIDN'T OUTPUT TO PRINT

.... DOESN'T LIKE .... AS INPUT

Logo sends this message when a procedure requires inputs, but gets the wrong kind.

    PRINT SQRT "APPLE
    SQRT DOESN'T LIKE "APPLE AS INPUT

    SETPOS [:X :Y]
    SETPOS DOESN'T LIKE [:X :Y] AS INPUT

Go back to the reference sections, look at the Logo words you have used, and discover what kind of inputs they expect. It isn't always obvious what is wrong. For example, SETPOS expects two numbers. Since :X and :Y are enclosed as a list in square barackets, they cannot be evaluated. You should have written

    SETPOS SE :X :Y

We have indicated, wherever possible, the traps of this kind.

OUT OF SPACE

This frustrating message is sent when Logo cannot fit another definition into its workspace, or when a procedure uses up all the available nodes while running and cannot finish its work. The solution is to tidy up your workspace, getting rid of unwanted variables and procedures, or maybe rewriting your procedure so that it uses less memory. Some recursive procedures are particularly greedy. Another solution may be to move to a MODE where you have more space. You can sometimes win space by freshening your Logo. You do this by saving you entire workspace:

    SAVE :FILENAME
    ERALL
    LOAD :FILENAME

This gets rid of some unwanted bits and pieces that the garbage collector may miss. You can do the same thing more quickly by moving everything into the Editor with EDALL, but if your problem is that you are OUT OF SPACE, there may not be room in the Editor for all the procedures you have to save.

LOGO NOT FRESH

This message warns you when you try to change MODE with procedures in the workspace. You have to clear them out with ERALL (saving them in the Editor or on Disk before you erase).

NOT POSSIBLE IN THIS MODE

This one is quite obvious and will be sent if you try to use graphics commands in MODES 3, 6 or 7, where text only is available.

. . . . ALREADY EXISTS

Occasionally, you try to define a procedure using a name which you have already given to another procedure, or you use the name of a primitive procedure, or you try to SAVE a file under a filename which is already used. In all these cases, Logo will stop you. For example:

```
TO COUNT :LIST
COUNT ALREADY EXISTS
```

STOPPED!!!

This is the message Logo sends when a procedure is stopped while it is running by the user pressing the ESCAPE key.

Other Logo Messages are:

WORD TOO LONG (more than 255 characters)
TOO MUCH INSIDE ( )
UNEXPECTED)
NUMBER TOO BIG
BAD FILE NAME   (eg. more than seven characters.)
YOU ARE AT TOP LEVEL   (eg. type STOP or END at top level)

SECTION THIRTY-ONE – GLOSSARY

The following is an alphabetically arranged glossary of all the primitives contained in Logotron Logo for the BBC Micro. It is designed for quick reference. More detailed descriptions are given elsewhere in the manual. Consult the index for page references. The # sign indicates a procedure is "greedy". It can handle any number of inputs, provided you enclose them in parentheses ( ). See Section 20 for an ex planation of the italicised *Inputwords*.

#AND *pred1 pred2*	Outputs TRUE if all its inputs are TRUE.
ASCII *char*	Outputs ASCII code (decimal) for *char*.
ARCTAN *n*	Outputs ARCTAN of *n* in degrees.
BACK (BK) *n*	Moves turtle *n* steps back.
BG	Outputs number representing background colour.
BF *object*	Outputs all but last element of *object*.
BL *object*	Outputs all but last element of *object*.
.CALL *n*	Transfers control to a machine code subroutine starting at address *n*.
CHAR *n*	Outputs character whose ASCII code is *n*.
CLEAN	Erases graphics without affecting the turtle's state.
CS	Erases graphics, restores turtle to home position [0 0] and heading to 0.
CT	Erases text.
.CONTENTS	Prints out list of workspace contents, including mistypes etc.
COUNT *object*	Outputs the number of elements in *object*
CURSOR	Outputs the current position of the text cursor.
COS *n*	Outputs the cosine of *n* degrees.

DEFINED? *name*   Outputs TRUE if *name* is defined
                     procname

.DEPOSIT *n byte*   Writes *byte* to address *n*.

EDALL   Writes all procedures and variables in
            the workspace to the editor.

EDIT (ED) *name/list*   Writes named procedures and/or
                           variables to the editor.

EMPTY? *object*   Outputs TRUE if *object* is empty, "
                     or [ ].

END   Special word indicating end of
          procedure definition.

ENVELOPE   Controls output of SOUND, see BBC
  num1 . . . num14   User Guide.

EOF? *filename*   If End Of File outputs TRUE, used in
                     conjunction with SETREAD and
                     SETWRITE.

EQUAL?   Outputs TRUE if its inputs are equal.
  object1 object2

ERALL   Erases everything in workspace, does
            not affect contents of editor.

ER *name/list*   Erases named procedures and/or
                    variables from workspace.

.EXAMINE *n*   Outputs contents of address *n* (see
                  .DEPOSIT).

FALSE   Special input for AND, IF, NOT, OR,
            output by predicates.

FENCE   Limits turtle's movements to the
            screen boundaries. See WRAP and
            WINDOW.

FIRST *object*   Outputs first element of *object*.

FORWARD (FD) *n*   Moves turtle forward *n* steps.

FPUT *object list*   Outputs new list formed by putting
                        object in front of *list*.

HEADING   Outputs turtle's heading.

HT   Makes turtle invisible.

HOME   Moves turtle to [0 0] and sets heading to 0, but does not clean graphics.

IF *pred list1 list2* IF *pred* is true, THEN run *list1*, ELSE run list2.

INT *n*   Outputs INTeger portion of *n*.

ITEM *n list*   Outputs ITEM *n* of *list*.

KEY?   Outputs TRUE if a key has been pressed but not yet read.

LAST *object*   Outputs LAST element of *object*.

LEFT (LT) *n*   Turns turtle *n* degrees counterclockwise.

LIST *object1 object2* Outputs a LIST of its inputs.

LIST? *object*   Outputs TRUE if *object* is a list.

LOAD *filename*   Loads *filename* from disk or cassette into workspace.

LPUT *object list*   Outputs new list formed by putting *object* at end of *list*.

MAKE *name object* Makes *name* refer to *object*. See THING.

MEMBER? *object list* Outputs TRUE if *object* is included in *list*.

MODE   Outputs number of current MODE (1- 7).

NAME? *name*   Outputs TRUE if *name* has a value (THING).

NODES   Outputs number of free NODES.

NOT *pred*   Outputs TRUE if *pred* is FALSE.

NUMBER? *object* Outputs TRUE if *object* is a number.

OPNS   Outputs NameS of variables currently in workspace.

OPPS   Outputs names of ProcedureS currently in workspace.

#OR *pred1 pred2* Outputs TRUE if any of its inputs are TRUE.

OP *object*   Returns control to caller, with *object* as OUTPUT.

PC   Outputs current pen colour.

PD   Puts turtle's pen down, and drawing.

PE   Turtle erases lines if it draws over them.

PU   Lifts turtle's pen so it is no longer drawing.

PO *name/list*   Prints definitions of named procedures and variables.

POALL   Prints out definitions of all procedures and variables.

POS   Outputs coordinates of turtle's position, x y.

PRIMITIVE? *name*   Outputs TRUE if *name* refers to a primitive procedure.

PRIMITIVES   Prints the list of all primitive procedures included in Logotron Logo.

#PRINT (PR) *object*   Prints *object*, stripping outer brackets and quotes, follows with carriage return.

#PROD *a b*   Outputs *a* multiplied by *b*.

QUOT *a b*   Outputs the INTeger QUOTient obtained by dividing *a* by *b* and truncating the answer.

RANDOM *n*   Outputs random positive integer between 0 and *n* − 1.

RC   Outputs character read by the current device (default is keyboard), waits if necessary. Does not echo output to screen.

RL   Outputs line read by the current device (default is keyboard), waits if necessary. Echoes output to screen.

RECYCLE   Forces garbage collection, freeing available NODES.

REMAINDER *a b* Outputs integer remainder obtained by dividing *a* by *b* and rounding the result.

ROUND *n* Outputs *n* rounded off to the nearest integer.

RIGHT (RT) *n* Turns turtle *n* degrees clockwise.

RUN *list* Runs *list*, outputs what *list* outputs.

SAVE *filename name/list* SAVEs to disk or cassette named procedures or variables, or entire workspace if no names are specified.

SCRUNCH Outputs current ratio of horizontal to vertical turtle steps.

#SE *obj1 obj2* Outputs unified *list* of its inputs.

SETBG *n* Sets background colour to colour *n*.

SETCURSOR [*x y*] Sets text cursor at position indicated by coordinates *x* and *y*.

SETH *n* Sets turtles heading to *n* degrees (0 up the screen).

SETMODE *n* Sets MODE to *n* of BBC Micro's MODES 1-7, see BBC User Guide.

.SETNIB *n* Produces special graphics effects; *n* must be in the range 0-255, equivalent to parameter K in BBC's PLOT command.

SETPC *n* Sets pencolour to colour *n*.

SETPOS [*x y*] Moves turtle to position given by *x* and *y* on graphics screen.

SETREAD *filename* Sets the *filename* from which RC and RL, will receive inputs.

SETREAD [ ] Closes the file opened with SETREAD.

SETSCRUNCH *n* Sets ratio (*n*) of horizontal turtle step to vertical turtle step.

SETWRITE *filename* Opens file and sends copy of all characters displayed on the screen to *filename*.

SETX *x*	Moves turtle horizontally to x-coordinate at *x*.
SETY *y*	Moves turtle horizontally to y-coordinate at *y*.
SHOW *object*	Prints *object* without stripping outer brackets.
SHOWN?	Outputs TRUE if turtle is showing.
ST	Makes turtle visible.
SIN *n*	Outputs the sine of *n* degrees.
SOUND *num1*...*num4*	Provides access to BBC Micro's sound facilities, see User Guide.
SQRT *n*	Outputs square root of positive *n*.
STOP	Stops current procedure and returns control to caller.
#SUM *a b*	Outputs the sum of its inputs.
TAN *n*	Outputs the tan of *n* degrees.
TS	Switches from graphics screen to text screen, clearing graphics and text.
THING *name*	Outputs object referred to by *name*, equivalent to :name.
TO *name inputs*	Signals start of title line of defined procedure.
TOPLEVEL	Stops all procedures and returns control to top level (ie keyboard).
TRACE *procname*	Enables user to trace all inputs and outputs of running procedures, used for debugging procedures.
TRUE	Special input for AND, IF, NOT, OR, output by predicates, see FALSE.
#TYPE *object1* . . .	Prints *object*, but leaves cursor at the end of line, without carriage return.
USE *modulename*	Links external modules containing special primitives or extensions to Logo.

VDU *list*   Gives access to BBC operating system. *List* contains parameters required to control VDU drivers. See BBC User Guide.

WAIT *n*   Causes Logo to WAIT *n* 60ths of a second before executing next instruction.

#WORD *word1 word2* Outputs word made up of its inputs.

WRAP   Maps turtle field onto torus, so that whenever it leaves screen, it reappears on opposite edge.

XCOR   Outputs x-coordinate of turtle's position.

YCOR   Outputs y-coordinate of turtle's position.

∗*suffix* *inputs*   Star commands, like VDU commands, give direct access to BBC Micro's operating system. See section 20 (p.96) and 28 (p.147).

a + *b*   Outputs *a* plus *b*.

a − *b*   Outputs *a* minus *b*.

a ∗ *b*   Outputs *a* times *b*.

a / *b*   Outputs *a* divided by *b*.

a < *b*   Outputs TRUE if *a* is less than *b*.

a > *b*   Outputs TRUE if *a* is greater than *b*.

object1 = *object2* Outputs TRUE if *object1* is equal to *object2*.

\   Tells Logo to treat the next character literally, without reference to its meaning.

SECTION THIRTY-TWO – INDEX

Logo Primitives are listed in **BOLD CAPITAL** letters, other procedures, mentioned in the text, are listed in CAPITAL letters.