

HOW TO WORK THE
LOGO MACHINE

BY
BENEDICT DU BOULAY
AND
TIM O'SHEA

D.A.I. OCCASIONAL PAPER

NO. 4



DEPARTMENT OF ARTIFICIAL INTELLIGENCE
UNIVERSITY OF EDINBURGH

HOW TO WORK THE

LOGO MACHINE:

a primer for ELOGO.*

by

Benedict du Boulay and Tim O'Shea

D.A.I. Occasional Paper

No. 4.

*To be used in conjunction with D.A.I. Occasional Paper No. 5

"Teaching Children LOGO: a metaprimer for ELOGO"

© du Boulay and O'Shea
All rights reserved

November 1976

CONTENTS

	Page
PREFACE	
ACKNOWLEDGEMENTS	
1. INTRODUCTION	1
2. USING THE DRAWING DEVICES	7
3. TYPING TO LOGO	13
4. YOUR OWN PROCEDURES	17
5. TIDY LOGO	24
6. CORRECTING MISTAKES IN YOUR PROCEDURES	26
7. TWO MEMORIES (Part 1)	29
8. PROBLEM BUG	33
9. YOUR OWN "POCKET" CALCULATOR	34
10. CALCULATING RESULTS	37
11. SUPER-PROCEDURES and SUB-PROCEDURES	41
12. BREAKING DOWN PROBLEMS	46
13. PROCEDURES WITH INPUTS (Part 1)	51
14. PROCEDURES WITH INPUTS (Part 2)	55
15. CHANGING PROCEDURES (Part 2)	56
16. TWO MEMORIES (Part 2)	58
17. PROCEDURES WITH INPUTS (Part 3)	61
18. POLYGONS	66
19. PROCEDURES WITH RESULTS	70
20. RECURSION	73
21. SPIRALS	78
22. TRUE or FALSE	79
23. CONTROL PROCEDURES	81
24. QUIZZES	84
25. STOPPING PROCEDURES	86
26. TRACING PROCEDURES	90
27. HOW LISTS WORK	93
28. FINDING THINGS IN LISTS	96
29. COMING BACK OUT OF RECURSION	99
30. WORKING ON LISTS	103
31. CONSTRUCTING LISTS	106
32. VARIABLES	111
33. USING PUBLIC BOXES	115

APPENDICES

		Page
A	PRINTING	118
B	MORE ABOUT DRAWING CIRCLES	120
C	THE TURTLE STATE	121
D	THE LOGO CLOCK	124
E	PAPER TAPE	125
F	AN ABBREVIATION FOR VALUE	127
G	MORE ABOUT DEFINING PROCEDURES	128
H	GLUEING THINGS TOGETHER	129
I	BOTH and EITHER	131
J	THE END OF THE LIST	133
K	WHILE	135
L	AND	137
M	GO	138
N	RUN	139
Index of procedure names (alphabetic)		140
Index of procedure types		143
Index of markers and prompts		144
Errata		145

PREFACE

This LOGO primer consists of

- (a) An ordered set of 33 worksheets designed to introduce the basic concepts of the Edinburgh implementation of LOGO.
- (b) An unordered set of 14 appendices designed to introduce various additional LOGO facilities which a student may need to complete a project. These are much terser than the worksheets.
- (c) A glossary of LOGO primitives.

The primer has been written for use by two specific groups of people working in our LOGO Laboratory. It is designed for a very particular implementation of LOGO* which has particular syntax, error messages, devices and filing system.

These notes have been used by eleven to thirteen year old boys and by trainee and serving school teachers. They have also been used by the Edinburgh undergraduate Artificial Intelligence course and by visitors to the Department.

The way in which the primer is intended to be used in teaching is described in a companion paper**. This paper also lists deficiencies, problems and possible improvements to the primer.

We would be most grateful to receive any critical comments on the content, organisation or style of the primer.

* 'Design Considerations for ELOGO' by McArthur, du Boulay, O'Shea and Howe.

** 'Teaching Children LOGO: a Metaprimer for ELOGO' by O'Shea and du Boulay.

ACKNOWLEDGEMENTS

The content and organisation of this primer owe a great deal to the work, comments and help of Ricky Emanuel, Colin McArthur and Richard Young. We are indebted to Jim Howe for his support and encouragement in the writing of the primer. We thank Jean Parker and Margaret Pithie for their patient hard work in producing this document and Doreen du Boulay for sustaining our efforts.

Many boys and teachers gave enormous help through their work with and comments on earlier versions of these notes.

The work is funded by a Social Science Research Grant No. HR 2981/1. Benedict du Boulay receives a Social Science Research Council Studentship.

INTRODUCTION

These notes are to help you learn and use LOGO. LOGO is a language which the computer understands. These notes assume you have no previous knowledge of computers. Each one explains some new ideas in LOGO. The names of these ideas will be written in capitals and underlined as they are introduced. Some of the words in the notes are words which are part of LOGO. These will be written in capitals but not underlined.

Each of these notes describes things you can do with the computer. Some of these may give you ideas for your own projects. Always explore any ideas, especially if you are not sure that they will work.

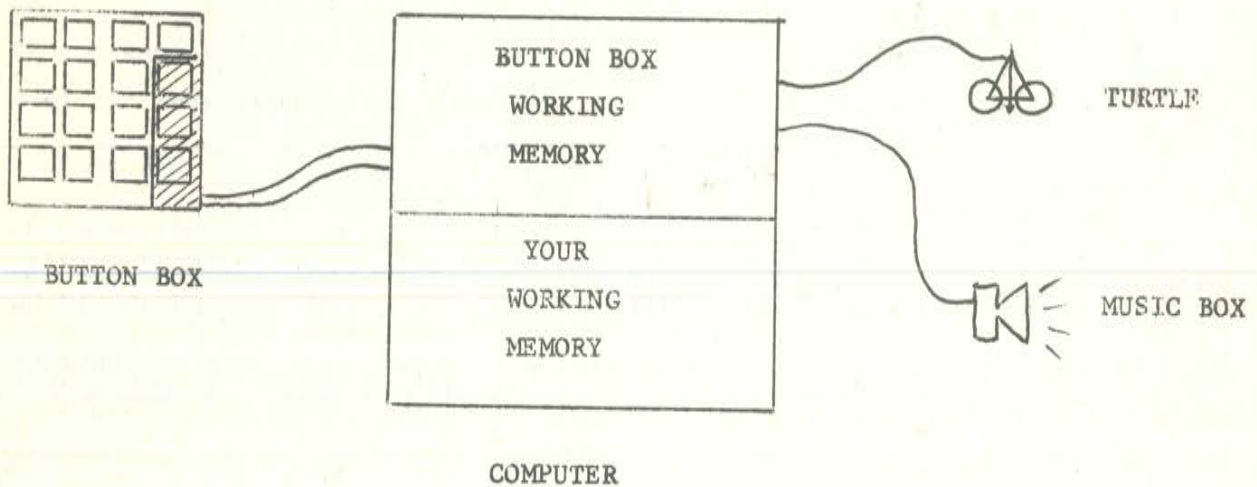
1. THE BUTTON BOX

The button box is a device which is used to send messages to the computer. The button box can be used to control either the turtle or the music-box via the computer.

A. Controlling the turtle

The computer can drive the turtle FORWARD, BACKWARD, LEFT or RIGHT. It can also make it HOOT. When the turtle moves it leaves a line on the floor by means of a pen fixed at its centre.

Pressing a button is COMMAND which the computer will EXECUTE and you will see the EFFECT. When all the lights are off the computer is WAITING for a command from you.



EXERCISE 1: Press every button except those labelled DEFINE, END and RUN.

FORWARD 1	FORWARD 5	FORWARD 20	HOOT
BACKWARD 1	BACKWARD 5	BACKWARD 20	DEFINE
RIGHT 5	RIGHT 15	RIGHT 90	END
LEFT 5	LEFT 15	LEFT 90	RUN

To understand how the turtle will be affected by commands, imagine yourself in the position of the turtle. If you were told FORWARD 5 you would walk 5 paces forward in whatever direction you happened to be facing. If you were told RIGHT 15 you would rotate on the spot towards the right through 15 degrees.

If you had paint on your feet you would see the effect on the floor of a sequence of such commands!

The important things about the turtle are where it is, its POSITION and which way it is facing, its HEADING. The POSITION and HEADING together are called the TURTLE STATE. FORWARD and BACKWARD commands change only the position part of the turtle state. LEFT and RIGHT change only the heading part of the state.

EXERCISE 2: Find out what sequence of commands will make the turtle face the opposite direction.

How many degrees are there in one complete turn?

Use the turtle to draw a triangle with equal size angles and equal length sides.

The computer has a WORKING MEMORY. It is divided into two parts, one for the button box and one for you. In the button box part of working memory are stored instructions of what EFFECT on the turtle each command should have.

You can use your part of the working memory to STORE A SEQUENCE OF COMMANDS, for example, a sequence to draw a triangle. Such a sequence of commands is called a PROCEDURE. Storing a procedure in working memory is called DEFINING A PROCEDURE.

To define a procedure press the DEFINE BUTTON.

EXERCISE 3: Press it.

This sets the computer in a special STATE, called the DEFINING STATE. When the computer is in this state the light in the define button stays on. Each command button that is pressed is added into your part of the working memory as the next command in your procedure. You can continue to add commands until your procedure is complete. While you are defining your procedure, the commands you give will NOT be executed, they will be stored away.

EXERCISE 4: Press command buttons to make a procedure.

EXERCISE 5: When you have finished your procedure press the END BUTTON.

This tells the computer that you have finished defining your procedure. The computer returns to the WAITING STATE when it is waiting to execute any command immediately. The light in the define button goes out.

Now that you have stored a procedure in working memory you can command the computer to execute the whole sequence of commands which make up your procedure. The computer reads this sequence from your part of the working memory. This is called RUNNING A PROCEDURE. You can run your procedure by pressing the RUN BUTTON.

EXERCISE 6: Run your procedure.

While the computer is executing your procedure it is in EXECUTING STATE and will ignore new commands. As the computer executes each command in your procedure the appropriate button will light up.

If you want to run your procedure again, just press the run button again, when the computer is waiting.

EXERCISE 7: Run your procedure a few more times.

If you want to define another procedure just press the store button again. The old procedure will be forgotten and the computer will be put in the defining state to store away your new procedure.

EXERCISE 8: Define a procedure to draw a hexagon (6 sided figure) with equal sides and equal angles. Hint: it takes 360 degrees for the turtle to turn right round once.

Define a procedure to draw an octagon (8 sided figure).

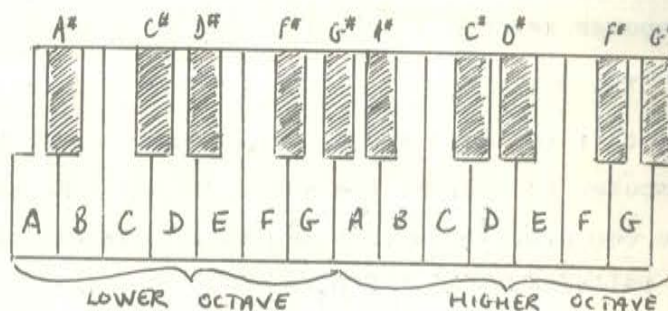
Define a procedure to draw a diamond.

Define some other procedures for your own pictures.

B. Controlling the music-box

In the button box part of working memory there are also instructions for working the music-box.

A	A [#]	B	OCTAVE
C	C [#]	D	DEFINE
D [#]	E	F	END
F [#]	G	G [#]	RUN



Buttons marked A, A[#], ..., G, G[#] play the notes of a single octave.

EXERCISE 9: Play all the notes in the octave.

If you want notes from an octave higher, press the OCTAVE BUTTON and then the note you want. Notes will continue to be in the higher octave until you press the octave button again (like a ballpen, one press for up, another press for down).

EXERCISE 10: Play the higher octave.

Play both octaves.

Play a tune.

The define, end and run buttons do the same job as before.

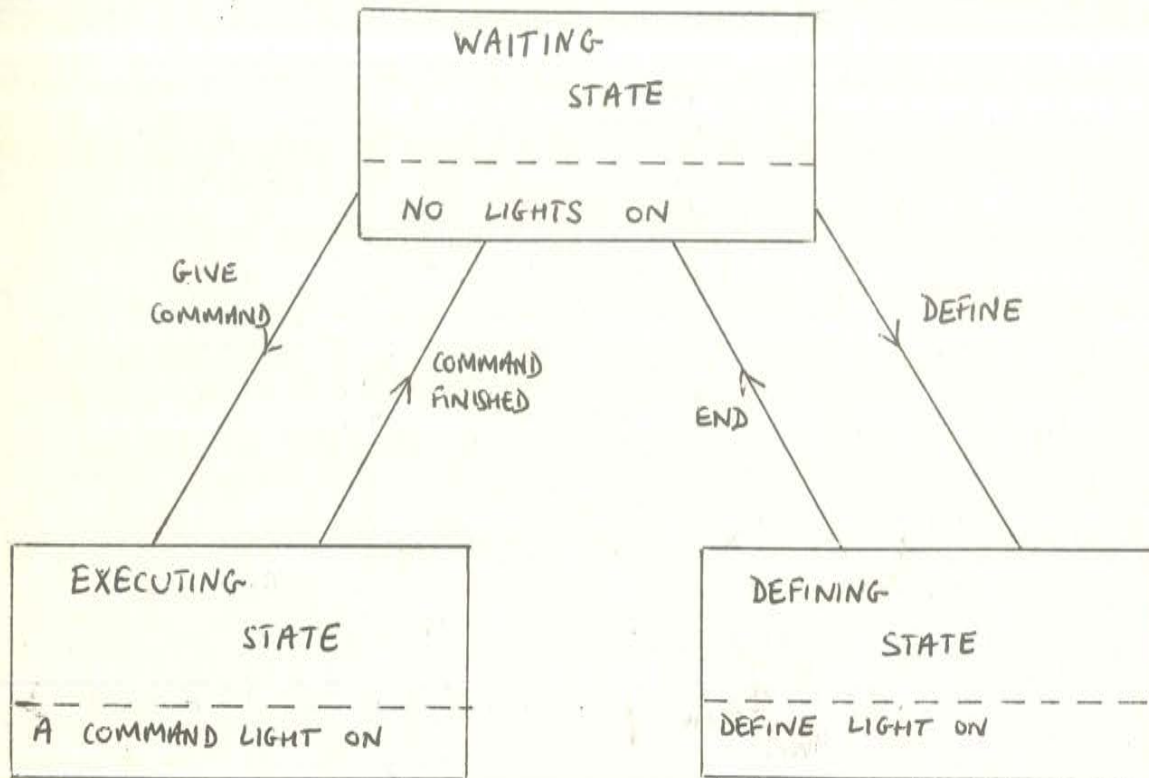
EXERCISE 11: Define a procedure to play Frère Jacques, this is how it starts:

C,D,E,C,C,D,E,C,E,F,G,E,F,G

Can you finish it off?

SUMMARY

The computer can be in one of three states: WAITING, DEFINING, EXECUTING.

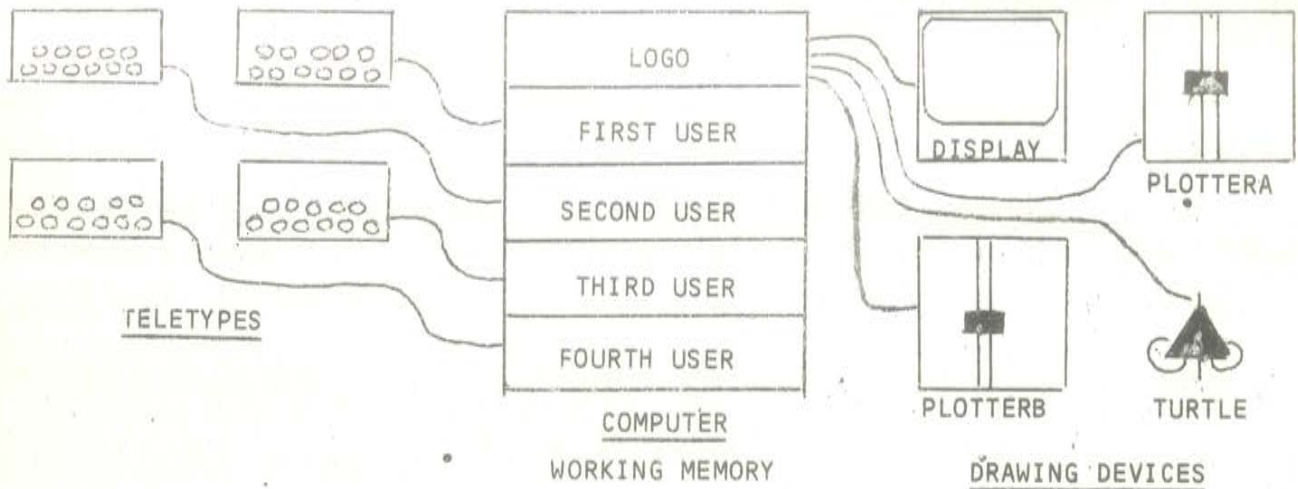


The arrows show you how to change the state of the computer. The boxes represent the three states.

The TURTLE STATE is its position and heading.

When the turtle goes FORWARD it moves in whatever direction it happens to be facing (POSITION changes). When the turtle turns LEFT or RIGHT it rotates on the spot (HEADING changes). When the turtle turns right round it turns through 360 degrees.

2. USING THE DRAWING DEVICES.



The working memory of the computer is divided into five sections, one for LOGO and one for each user.

A. Starting

When LOGO types

WHO ARE YOU:

then type your full name and then press the GREEN COMMAND BUTTON.

You will then be given a section of the working memory. LOGO will then type a PROMPT

W:

which means it is in the WAITING STATE and is waiting for your next COMMAND. LOGO's job is to execute your COMMANDS one at a time. Each command will have an EFFECT.

B. Choosing a drawing device

Just as you used the button box for drawing so you can use LOGO.

But now you have a choice of four drawing devices, two PLOTTERS, the DISPLAY or the FLOOR TURTLE.

You must tell LOGO which drawing device you would like your teletype to be connected to. There are PROCEDURES to connect you to the devices.

The NAMES of the procedures are:

DISPLAY
TURTLE
PLOTTERA
PLOTTERB

To command LOGO to connect your teletype to one of the drawing devices you must type the name of one of these procedures and press the green command button. This button tells LOGO that you have finished typing a command and that this command must be executed at once.

EXERCISE 1: Connect yourself to a drawing device.

If you are using the floor turtle, put it in the middle of its board. The display has an imaginary turtle which draws on its screen. The plotters use their pens as turtles. These imaginary turtles always start in the middle of their drawing area.

The four procedures for connecting to the drawing devices are stored in LOGO's section of the working memory.

When LOGO reads the name of a procedure you have typed, it looks it up in the working memory to find out what should be done, and then executes the sequence of instructions associated with that name.

If you type the name of a procedure which LOGO cannot find in its working memory, LOGO types an appropriate MESSAGE.

EXERCISE 2: Try typing
W: CLOTTERA

C. Drawing

The names of the procedures for drawing were the labels used on the button box:-

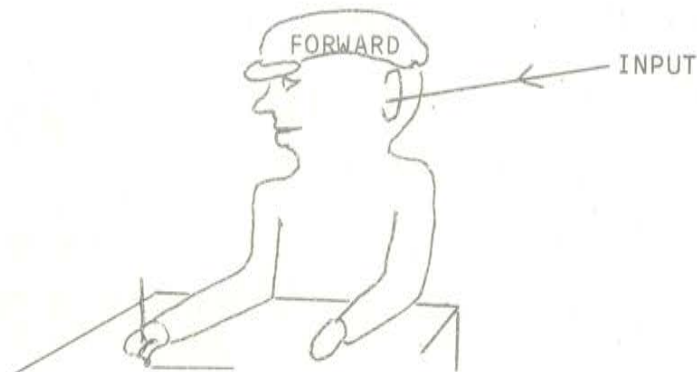
FORWARD
BACKWARD
LEFT
RIGHT

Each procedure is like a WORKER who knows how to do a particular

job and will execute that job when commanded to by having his name called. Some of these workers need information to be able to do their jobs.

EXERCISE 3: Type FORWARD and press the green command button.

It is no good just commanding FORWARD without telling this worker how far to move forward. Giving this information is called giving an INPUT.



We give an input by typing the name of the procedure and then a space and then a number. The number is the number of steps to go forward, in whatever direction the turtle is facing.

EXERCISE 4: Try typing
W: FORWARD 125
W: BACKWARD 16

When LOGO reads the name of the procedure FORWARD it looks it up in its working memory and finds that this worker needs an input. LOGO continues reading your command from left to right looking for the input.

The space between the procedure name and the input is important. It tells LOGO where the procedure name stops and the input starts.

EXERCISE 5: Try typing
W: FORWARD7

The message from LOGO tells you that it could find no procedure called FORWARD7 in its working memory. LOGO could not execute your

command is now WAITING for another command.

The turtle may be rotated ON THE SPOT towards the LEFT or RIGHT.

The two procedures each need an input to tell them how much to rotate.

EXERCISE 6: Try typing
W: RIGHT 62
W: LEFT 33

If you are connected to a plotter, watch its COMPASS!

Try driving the turtle to draw a square for example.

LOGO expects only one command at a time. If you give more than one command before pressing the green command button, only the first command on the line will be executed and the rest ignored.

EXERCISE 7: Try typing
W: FORWARD 24 LEFT 50 BACKWARD 256
W: 256

LOGO expects a command and does not know what to do with the number 256.

There is a procedure for putting the turtle back in the centre of its drawing area. The name of the procedure is:

CENTRE

The turtle will be moved to the centre with its PEN up so that no line is drawn. The turtle will be left facing towards the right, with its pen down again.

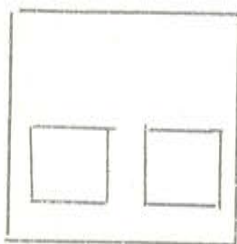
EXERCISE 8: Trying running CENTRE.
Try to draw a triangle with equal sides
Try to draw a hexagon with six equal sides.

You can also move the turtle without drawing a line if you first raise its pen. There are procedures to raise and lower the turtle's pen. Once the pen is raised, no more lines will be drawn until you lower it. The names of the procedures are:-

LIFT
DROP

The pens on the plotters are held a little off the paper except when they are actually moving to stop ink splodges. The effect of running LIFT is to keep the pen off the paper even when it is moving.

EXERCISE 9: Draw some separated shapes.



Sometimes you may lose track of exactly what the TURTLE STATE is. There is a procedure which will draw you the turtle, which you can use when you are connected to the DISPLAY or the PLOTTERS. The name of this procedure is:

WHERE

There are two other special procedures. One for clearing the display screen and one for making the floor turtle HOOT. The names of these procedures are:

CLEAR

HOOT

EXERCISE 10: If you are connected to the display, try running WHERE then moving the turtle FORWARD and run WHERE again. Run CLEAR.

D. Changing your drawing device

There is a procedure to disconnect you from your drawing device. Its name is:

FREE

EXERCISE 11: Swop drawing device with someone, but stay sitting at the same teletype.

E. Finishing the LOGO session

There is a procedure which empties your section of the working memory ready for a new user. The procedure also disconnects you from any drawing device. The name of the procedure is:

GOODBYE

SUMMARY

When LOGO types the PROMPT W: it is in the WAITING STATE, waiting to EXECUTE a single COMMAND. You give a command by typing the NAME of a PROCEDURE, with an INPUT if needed, and then press the GREEN COMMAND BUTTON.

Here is a table of new LOGO procedures.

NAME OF PROCEDURE	INPUT	EFFECT OF PROCEDURE
DISPLAY	no input	connects teletype to display
TURTLE	no input	connects teletype to floor turtle
PLOTTERA	no input	connects teletype to plotter a
PLOTTERB	no input	connects teletype to plotter b
FORWARD	one number	moves forward
BACKWARD	one number	moves backward
LEFT	one number	rotates leftwards
RIGHT	one number	rotates rightwards
LIFT	no input	raises pen
DROP	no input	lowers pen
CENTRE	no input	turtle to centre, facing right
CLEAR	no input	clears display screen
HOOT	no input	floor turtle hoots
WHERE	no input	draws turtle on display or plotters
FREE	no input	disconnects drawing device
GOODBYE	no input	empties working memory.

3. TYPING TO LOGO

This note describes how to use the teletype and how to correct typing mistakes.

LOGO reads a difference between 1 (number) and l (letter). LOGO also reads a difference between 0 (number) and o (letter).

There is a shift key, or button, as on a normal typewriter. This is used when there are two characters printed on one button, e.g.



If you do not touch the shift key and press (2) you will type the lower character 2. Holding the shift key down and pressing (") will type the upper character " .

A. Correcting typing mistakes

If you make a typing mistake you can make it invisible to LOGO, but not rub it off the paper. If the last character you typed was wrong just press (DE
LETE)

This will make that last character invisible to LOGO which will type you +

EXERCISE 1: Connect to a drawing device and type
W: FORWARD 123

If the mistake was not the last character typed, you must press (DE
LETE) a sufficient number of times to make all the characters back to the mistake invisible to LOGO and then continue on again from the mistake

For example: W: GOOFBYE<<<<DBYE

If your mistake is right at the beginning of a long line it may be easier to tell LOGO to ignore the whole line and start again. Hold

the CTRL button down and press X, LOGO will type ↑ and the whole line will be ignored. LOGO will give you a new prompt.

EXERCISE 2: Try typing
W: RACKWARD 556↑
W:

B. PRINTING

There is a procedure named PRINT which needs one INPUT. PRINT has the EFFECT of making the teletype type the input you give it.

EXERCISE 3: Try typing
W: PRINT 79

Find the largest number you can command to be printed.

PRINT can also have a WORD as its input.

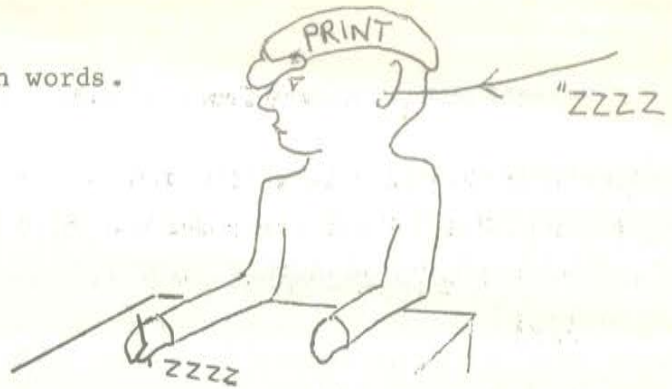
EXERCISE 4: Try typing
W: PRINT CAT

The reason LOGO sent you a MESSAGE and did not type CAT was because LOGO looked for a PROCEDURE named CAT in its working memory and could not find one there.

To mark the difference between the NAME OF A PROCEDURE TO BE EXECUTED and a WORD TO BE USED AS INPUT, we use a special character ". This we will call the quote sign. When LOGO reads this it assumes that the word immediately following is to be used as an input and is not to be executed.

EXERCISE 5: Try the following
W: PRINT "CAT
W: PRINT "RHUBARB
W: PRINT "PRINT
W: PRINT "ZZZZZZZZ

LOGO words do not have to be English words.

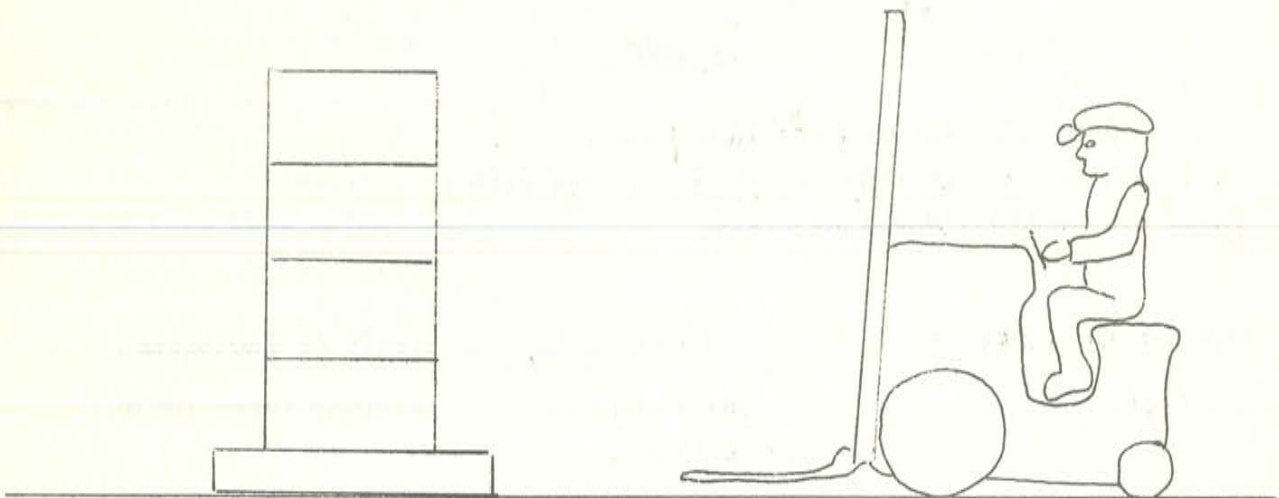


EXERCISE 6: Try

W: PRINT "HELLO "MOTHER "HOW "ARE "YOU

PRINT expects only ONE INPUT so LOGO does not know what to do with the extra words.

To have more than one word as an input you must put all the words into a LIST. A LIST is like a stack of boxes each of which could hold words or numbers. The boxes are all stacked on a pallet so that they can be picked up as one stack.



The list can be the one input for PRINT.

The beginning of a list, the top of the stack, is marked with [and the end of the list, the pallet is marked with]. These two characters are called LIST BRACKETS. We call each box of the stack an ELEMENT of the LIST.

EXERCISE 7: Try typing
 W: PRINT [HELLO MOTHER HOW ARE YOU]
 W: PRINT [ZZZZZZZZZ]
 W: PRINT [I AM 21 TODAY]
 W: PRINT [1 2 3 4 5 6 7]

LOGO does not look in the boxes to find procedures to execute.

EXERCISE 8: Try typing
 W: PRINT [FORWARD 100]

SUMMARY

You can make LOGO ignore typing mistakes.

There are only three types of INPUT you can give procedures.

These are NUMBERS, WORDS or LISTS.

NAME OF PROCEDURE	INPUT	EFFECT OF PROCEDURE
PRINT	one number, or one word, or one list,	teletype types input.

4. YOUR OWN PROCEDURES

A. Defining your own procedures

When you used the button box you were able to store a single sequence of commands in the working memory. This was called DEFINING A PROCEDURE.

This was useful because once the procedure was defined you could run it over and over again without having to remember or press the individual commands.

LOGO allows you to define as many separate procedures as you like and stores them all in your section of the working memory. Because you can have more than one procedure in the working memory it is necessary to give each procedure its own NAME so that you can run the one you want.

Defining a procedure is like telling a WORKER how to do a job. Your instructions to the worker will be an ordered sequence of commands.

For example:-

```
HOW TO BOIL AN EGG
1 GET AN EGG
2 BOIL SOME WATER
3 PUT THE EGG IN THE WATER
4 WAIT A FEW MINUTES
5 TAKE THE EGG OUT OF THE WATER
THATS ALL
```

In order to define a procedure you must put LOGO in the DEFINING STATE. There is a procedure named

DEFINE

to do this which needs one input. This input must be a LOGO word. It is used to make the NAME of your new procedure.

When the computer is in the DEFINING STATE the PROMPT changes to

D:

You can use any LOGO word, e.g. "FRED or "SQUARE which you must make up yourself.

In our example we show how we define a new procedure which will be named SPIKE whose job will be to draw a spike.

W: DEFINE "SPIKE

D: 1 FORWARD 55

D: 2 LEFT 110

D: 3 FORWARD 80

D: 4 LEFT 140

D: 5 FORWARD 80

D: END

When LOGO is in the DEFINING STATE commands will not be executed, they will just be stored away TIDILY in LINE NUMBER ORDER as instructions for the worker named SPIKE.

The procedure named END changes LOGO back to the WAITING STATE and tells LOGO that the worker has been given all his instructions.

Once the procedure is defined it is stored in the working memory until you run GOODBYE which empties the working memory.

EXERCISE 1: Define the procedure SPIKE.

You may name it something else if you wish.



B. Running your own procedure

In the button box you pressed a special button to run the one procedure you could define.

In LOGO we run procedures by typing their names without the quote sign.

LOGO assumes that names without quote signs are procedures to be executed. Running your procedure is like telling the worker to actually carry out the instructions you have already told him.

For example: "Boil an egg, please!".

To run the procedure SPIKE we just type its name WITHOUT THE QUOTE SIGN.

EXERCISE 2: Run the procedure SPIKE

W: SPIKE

W:

Run the procedure a few more times.

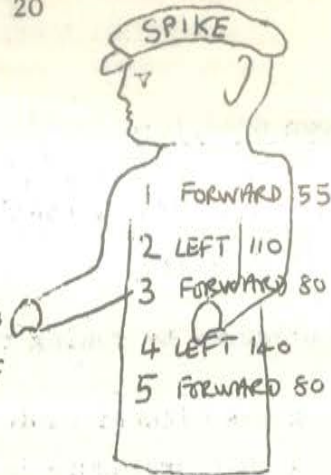
SPIKE'S job is to supervise the execution of his five commands in the right order. Each one of his commands contains the name of one of LOGO'S procedures, or workers. SPIKE will have to supervise these other workers. Executing a procedure is a PROCESS which takes time.

To help explain how SPIKE supervises his workers we will draw some SNAPSHOTS from a MOVIE of him doing his job.

When we run SPIKE by typing

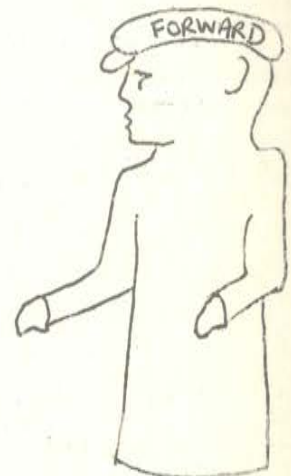
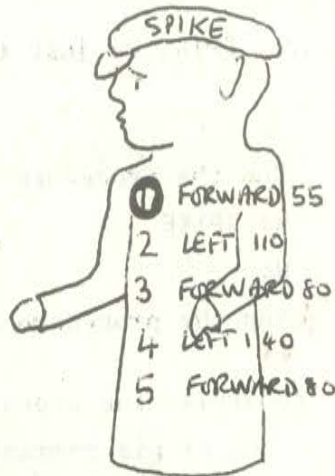
W: SPIKE

LOGO looks up the name SPIKE in its working memory and calls the worker SPIKE.



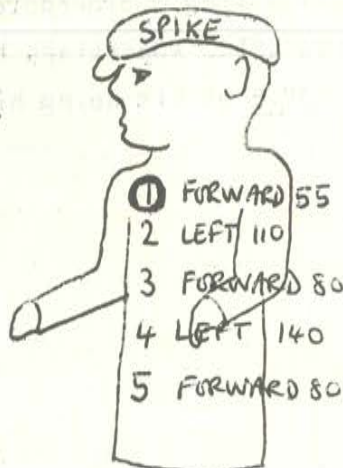
SNAPSHOT 1

SPIKE reads his first command and calls up the worker FORWARD from working memory. SPIKE hands over to FORWARD and leaves a marker on the line he has reached to remember how far he has got.



SNAPSHOT 2

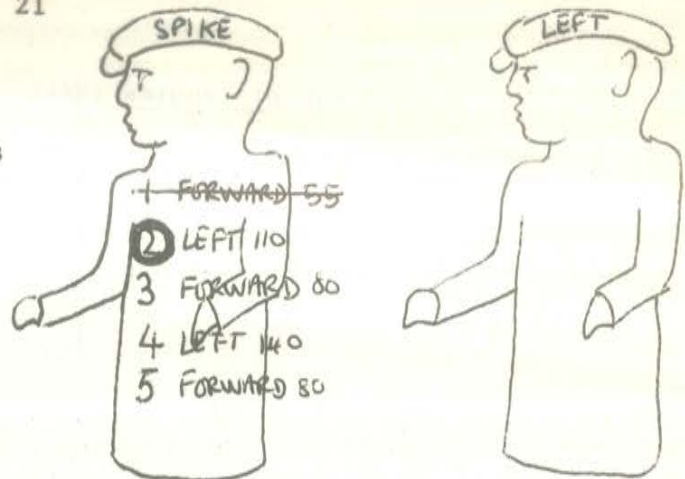
FORWARD needs one input and reads it from the line in SPIKE which called him. FORWARD does his work and then tells SPIKE when he has finished.



SNAPSHOT 3

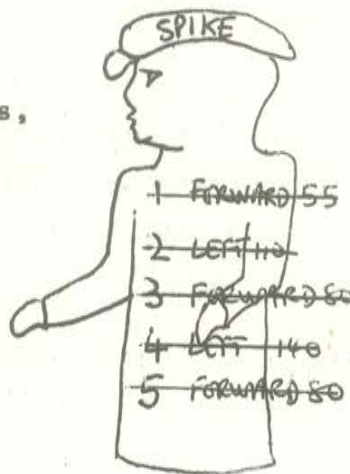
SPIKE looks at his marker and moves to his next command which involves calling up LEFT.

SNAPSHOT 4



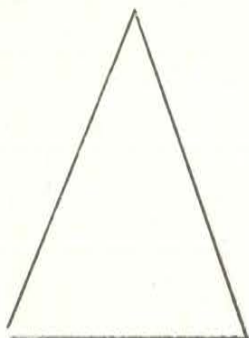
Sometime later, when SPIKE has executed all his five commands, having called up five other workers, SPIKE tells LOGO that he has finished.

SNAPSHOT 5



LOGO types the prompt W: It is waiting for a new command. The EFFECT of SPIKE was to draw a spike.

SNAPSHOT 6



C. Seeing your own procedures typed out by LOGO

There is a procedure you can run to have LOGO type out the whole of one of your own procedures. This procedure needs one input, the LOGO word used to make your procedure name. The name of this procedure is:-

SHOW

EXERCISE 3: SHOW the procedure SPIKE
W: SHOW "SPIKE

EXERCISE 4: Try typing
W: SHOW SPIKE

When you typed this LOGO tried to draw a spike and you got a funny message. This was because SPIKE did not have a quote sign in front of it. So LOGO tried to execute it.

You now know enough about LOGO to make a whole variety of procedures. Here are some ideas:-

EXERCISE 5: You could draw procedures for a diamond, a hat, or any other shape.
You could try this procedure out on your friends.

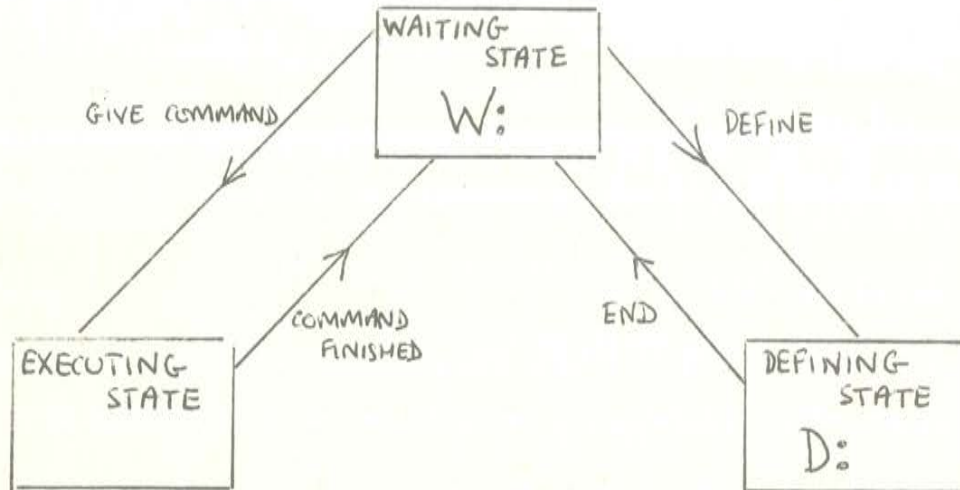
W: DEFINE "SURPRISE

D: 1 PRINT [THE PERSON SITTING AT THIS TELETYPE]
D: 2 PRINT "IS
D: 3 PRINT "VERY
D: 4 PRINT "VERY
D: 5 PRINT "VERY
D: 6 PRINT "P
D: 7 PRINT "A
D: 8 PRINT "T
D: 9 PRINT "I
D: 10 PRINT "E
D: 11 PRINT "N
D: 12 PRINT "T
D: END

In note 6 we will tell you what to do when the EFFECT of running your procedure is not what you intended!

SUMMARY

LOGO can be in one of three states:- WAITING, EXECUTING or DEFINING.



The PROMPT tells you in which state LOGO is.

The new procedures are:

NAME OF PROCEDURE	INPUT	EFFECT
DEFINE	<u>LOGO</u> word to be a procedure name	puts LOGO in the defining state.
END	no input	puts LOGO in the WAITING STATE.
SHOW	<u>LOGO</u> word which is a procedure name	types out procedure.

5. TIDY LOGO

LOGO in the DEFINING STATE is very TIDY. It stores the commands of your procedures in the order of their line numbers even if you type the commands in the wrong order!

The line numbers do not even have to count up in ones. You could have 3,2,7,103,54 or 10,20,30,40 as line numbers. LOGO always puts the commands in the order of size of the line numbers, with smallest first.

EXERCISE 1: Try defining this procedure FUNNY in this wrong order.

```
W: DEFINE "FUNNY
  D: 50 FORWARD 120
  D: 10 FORWARD 120
  D: 33 FORWARD 120
  D: 70 FORWARD 120
  D: 40 LEFT 135
  D: 60 LEFT 45
  D: 20 LEFT 45
  D: END
```

What shape would it draw?

RUN the procedure to see if you were right.

SHOW the procedure to see that LOGO has tidied it.

While you are defining a procedure, and LOGO is still in the DEFINING STATE you may notice that you have typed in a line wrongly. Just type the line again with the same line number. Tidy LOGO will not let a procedure have two lines with the same number. So the old line will be rubbed out and replaced with the new version.

EXERCISE 2: Why do you think LOGO does not allow two lines in a procedure with the same number?

How will LOGO tidy the following procedure AWFUL ?

Check your answer by defining it as it is below.

```
W: DEFINE "AWFUL
  D: 5 FORWARD 80
  D: 5 FORWARD 180
  D: 10 LEFT 90
  D: 15 FORWARD 280
  D: 20 LEFT 90
  D: 5 FORWARD 280
  D: 15
  D: 12 FORWARD 280
  D: END
```

Now SHOW procedure AWFUL.

In future we will usually define procedures with the line numbers counting in tens, 10,20,30 etc. This leaves room for other lines to be INSERTED if we forget them.

SUMMARY

LOGO TIDIES your procedures by putting the lines in order.

6. CORRECTING MISTAKES IN YOUR PROCEDURES

There are two kinds of mistake or BUGS which may make you wish to **CHANGE** one of your procedures:

- (A) the procedure does not have the EFFECT you wanted
(PROBLEM BUG).
- (B) there is a command in the procedure which LOGO cannot execute
(GRAMMAR BUG)

In this note we will show you how to deal with grammar bugs. In a later note we will deal with problem bugs.

EXERCISE 1: Define this procedure with its mistake.

```
W: DEFINE "GROTTY
    D: 10 FORWARD 100
    D: 20 DAFT 90
    D: 30 FORWARD 100
    D: END
```

RUN the procedure.

The MESSAGE tells you:-

```
which command cannot be executed
why the command cannot be executed
which procedure contained the command
which line of the procedure contained the command
```

To correct the bug in this procedure we need to REPLACE line 20.

EXERCISE 2: Try typing a replacement for line 20, e.g.

```
W: 20 LEFT 90
```

LOGO did not know what to do because it did not know which procedure you wanted to change. There are usually a lot of procedures all with line 20 s!

There is a procedure named

CHANGE

which needs one input. This input is the quoted name of the procedure you wish to change. When CHANGE is executed LOGO is put in the DEFINING STATE. The prompt changes to

D:

Once LOGO is in the defining state we can retype any line we want to change. Tidy LOGO throws away the old version of the line and stores the new one.

EXERCISE 3: CHANGE the procedure GROTTY

W: CHANGE "GROTTY

D: 20 LEFT 90

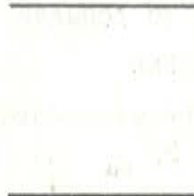
D: END

Lines not mentioned are not changed.

Type SHOW "GROTTY to see how line 20 has been replaced.

In the same way new lines can be INSERTED if need. All we have to do is run CHANGE and type in lines with the right new numbers.

EXERCISE 4: Insert new lines into GROTTY so that it draws an open box



Sometimes it is necessary to DELETE lines from a procedure. Again you just run CHANGE and then run the procedure DELETE. This procedure needs one input which is the line number of the line you wish to delete. The procedure DELETE can only be run after LOGO is in the defining state.

EXERCISE 5: Delete lines 10, 20 of GROTTY so that its effect is now to draw an L shape

W: CHANGE "GROTTY

D: DELETE 10

D: DELETE 20

D: END

Run and SHOW the latest version of GROTTY.

Tidy LOGO will not allow you to store away two procedures with the same name.

EXERCISE 6: Try to DEFINE GROTTY again

W: DEFINE "GROTTY

SUMMARY

You can change one of your procedures by running CHANGE and making the appropriate correction. Lines may be REPLACED, INSERTED or DELETED.

The new procedures are

NAME OF PROCEDURE	INPUT	EFFECT
CHANGE	Quoted name of procedure to be changed	Sets LOGO in DEFINING STATE
DELETE	line number	line is deleted.

7. TWO MEMORIES

ON OFF

LOGO
FIRST USER
SECOND USER
THIRD USER
FOURTH USER

WORKING MEMORY

CALUM'S	SCOTT'S
HANS'	DUGALD'S
SHAUN'S	KENNETH'S
DAVID'S	PHILIP'S
GREGOR'S	GRAEME'S
JASON'S	ROBIN'S

PERMANENT MEMORY

Any procedures you define are stored in your part of the WORKING MEMORY until you finish your LOGO SESSION by running GOODBYE. If you wish to be able to run one of your procedures at another session and do not want to have to define it again you can command LOGO to REMEMBER the procedure. LOGO will REMEMBER it by first making a COPY of the procedure. Then LOGO will transfer the COPY to your part of a special memory called the PERMANENT MEMORY.

Procedures in the permanent memory are safe even if LOGO is switched off or breaks down.

The procedure for remembering is named

REMEMBER

This procedure needs one input. This input should either be the quoted name of one procedure to be remembered or a single LIST of procedure names, for example

"SPIKE or [SPIKE SQUARE TRIANGLE]

LOGO never looks inside a LIST for procedures to EXECUTE.

EXERCISE 1: Command LOGO to REMEMBER the procedures you have defined in WORKING MEMORY today. If you have not defined a procedure yet, define one.

W: REMEMBER "SPIKE

SPIKE REMEMBERED

The permanent memory has a magnetic disc. This works in much the same way as a magnetic tape cassette in a tape-recorder.

It is usually a good idea to command LOGO to REMEMBER your procedures once you have defined them since they will be safe should LOGO break down.

LOGO only makes a COPY of a procedure when it remembers it, like a photograph. If you CHANGE the original procedure in WORKING MEMORY the copy, or photograph in PERMANENT MEMORY will not be affected. Photographs of yourself as a baby do not change as you get older!

EXERCISE 2: Make sure that you have commanded LOGO to
REMEMBER all the procedures you wish to keep.
Then run GOODBYE
Then start a new session by typing ELOGO
Try to run any of the procedures that used
to be in working memory.

When you start a session the working memory is empty. Before you can run one of your remembered procedures you will have to command LOGO to RECALL the procedure from the permanent memory. The name of the procedure which does this is

RECALL

This procedure needs one input, either a single procedure name or a list or names, just like REMEMBER.

EXERCISE 3: Command LOGO to RECALL all of your procedures
from permanent memory.

When LOGO RECALLS a procedure from permanent memory it just makes a COPY of the procedure and transfers the copy to working memory. This means that whatever you do to the copy in working memory you will always have the original version safe in permanent memory.

If you ever give LOGO a command which would make it put a copy of a procedure with a certain name into a memory where there is already a

procedure with the same name then the procedure put in last is the one kept.

This is just like TIDY LOGO in the defining state. If you define a procedure with two lines with the same line number then the line typed last is the one kept.

In the same way TIDY LOGO does not let you DEFINE two procedures with the same name.

EXERCISE 4: Define a procedure named TWIN

W: DEFINE "TWIN

D: 10 FORWARD 100

D: END

TWIN DEFINED

Then try to define TWIN again.

If you want to erase a procedure from working memory so that LOGO no longer knows the definition you must run the procedure

UNDEFINE

which needs one input which should be the name of your procedure to be undefined.

EXERCISE 5: Erase the definition of TWIN

W: UNDEFINE "TWIN

Now you could define a new procedure TWIN
if you wished.

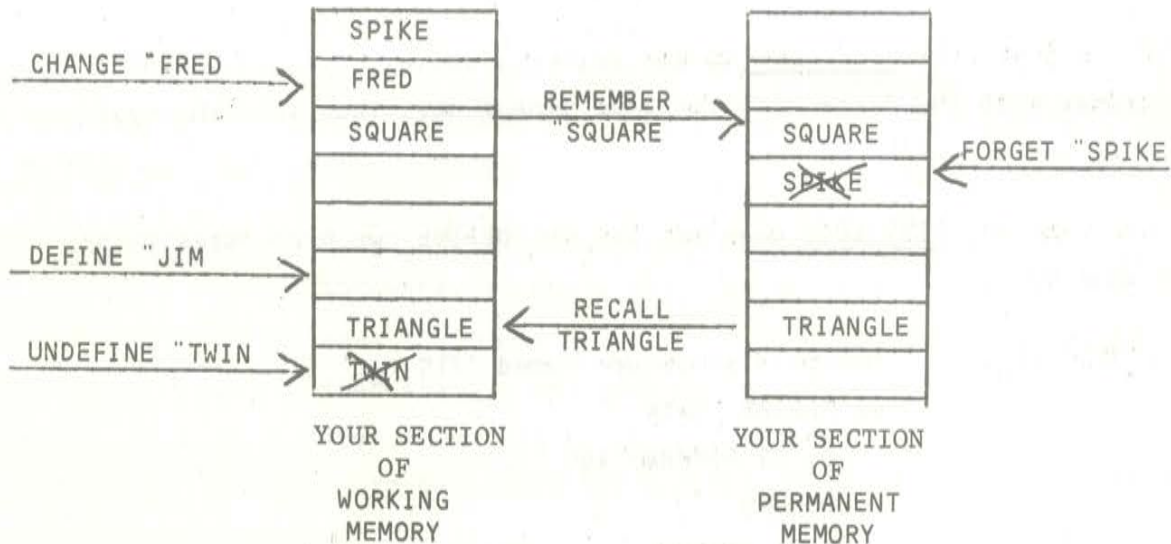
If you wish to erase a procedure from the permanent memory you command LOGO to forget it by running the procedure

FORGET

which needs one input, like REMEMBER.

EXERCISE 6: Command LOGO to FORGET a procedure in permanent memory. Be careful to choose one you do NOT want to keep!

W: FORGET "SPIKE

SUMMARY

Only procedures in working memory may be run or changed or undefined.

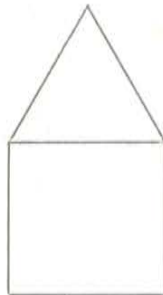
Only procedures in permanent memory may be kept from day to day.

A procedure in working memory may be copied (photographed) into permanent memory by running REMEMBER.

A procedure in permanent memory may be copied into working memory by running the procedure RECALL. It may be forgotten by running FORGET.

The new procedures are:-

NAME OF PROCEDURE	INPUT	EFFECT
REMEMBER	procedure name or list of names	copies from working to permanent memory
RECALL	procedure name or list of names	copies from permanent to working memory
UNDEFINE	procedure name	erases definition from working memory
FORGET	procedure name	erases definition from permanent memory.

8. PROBLEM BUG

When you try the following exercise your work will probably be attacked by a PROBLEM BUG.

EXERCISE 1: Define a procedure named SQUARE which draws a square.
Define a procedure named TRIANGLE which draws a triangle.
Command LOGO to draw a house by running SQUARE and then running TRIANGLE.

You probably found that the roof did not go where you wanted it to. This is caused by a PROBLEM BUG. We call the process of solving such problems DEBUGGING.

HINT: Think about the TURTLE STATE, its heading and position, after it has drawn the square.

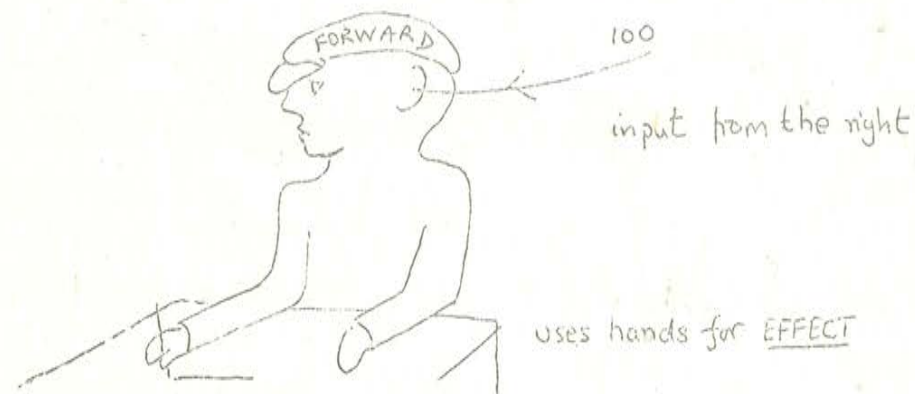
SUMMARY

One of the best breeding grounds for problem bugs is when you use several of your own procedures together.

9. YOUR OWN "POCKET" CALCULATOR

We have described procedures as WORKERS. So far all the workers have been run because they produce some EFFECT. In most cases the effect has depended on an INPUT. The workers are very well disciplined.

They always listen for their input FROM THE RIGHT.



The workers will only carry out their effect once they have the correct number of inputs.

EXERCISE 1: Try running the drawing procedure named

ARC

This procedure needs two number inputs.

Find out what the worker uses each input for.

There are other kinds of workers which do not produce an effect. These workers use their inputs to calculate a RESULT which they speak to the worker ON THEIR LEFT. We are unable to see this RESULT unless we arrange for it to be the input of a worker who produces a visible effect. The name of a procedure which only gives a RESULT is

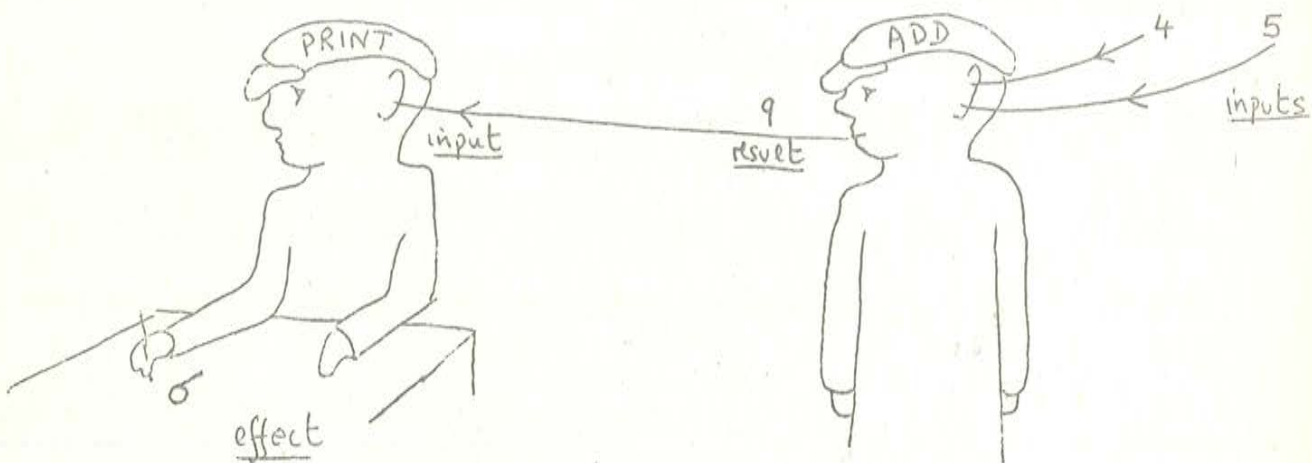
ADD

This procedure needs two inputs, both of which should be numbers.

EXERCISE 2: Try typing
 W: ADD 4 5
 W: PRINT ADD 4 5

In the second line of the exercise two workers were arranged so that the RESULT from ADD was the INPUT to PRINT

Remember all the workers look for their inputs to their right.



There are several other procedures which calculate a result. They each need two number inputs. Their names are

SUBTRACT
 MULTIPLY
 DIVIDE

EXERCISE 3: Command LOGO to do calculations for you using these procedures

Command LOGO to add up three numbers. This is hard as the procedure ADD needs only two inputs!

HINT: Arrange more than one worker ADD in your command.

LOGO only knows about integers (whole numbers) so the procedure DIVIDE calculates its result to NEAREST WHOLE NUMBER below the answer.

EXERCISE 4: Try
 W: PRINT DIVIDE 8 3
 W: PRINT DIVIDE 5 12

There is a procedure named

REMAINDER

which needs two inputs. This procedure gives as its result the remainder obtained when its second input is divided into its first input.

EXERCISE 5: Try
 W: PRINT REMAINDER 22 5

SUMMARY

Some procedures like PRINT or FORWARD produce an EFFECT. Other procedures like ADD or SUBTRACT calculate and give a RESULT. In order to see this result it has to be the input of a procedure which gives an effect.

The names of the new procedures are:-

NAME OF PROCEDURE	INPUT	RESULT
ADD	two numbers	adds numbers
SUBTRACT	two numbers	subtracts second number from first
MULTIPLY	two numbers	multiplies numbers
DIVIDE	two numbers	divides second number into first
REMAINDER	two numbers	remainder when second number divided into first
ARC	two numbers	draws arc curving left of radius given by first input and angle given by second input.

10. CALCULATING RESULTS

In the last note we showed you how you can give LOGO complicated commands. These commands are made by arranging the workers so that each one looks for its inputs on the right and hands on its result to the worker on its left.

The worker at the extreme left must produce an effect rather than hand on a result. Otherwise we will never know what the workers did.

You can give LOGO very complicated commands if you wish. But it is important to understand how LOGO reads your command.

EXERCISE 1: Try to work out what LOGO will print when you command



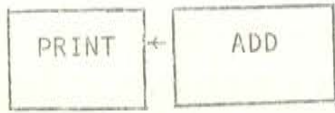


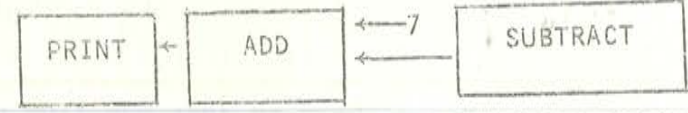
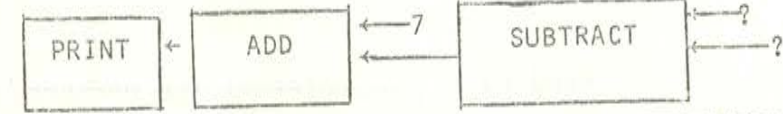
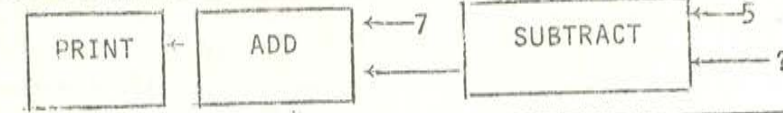
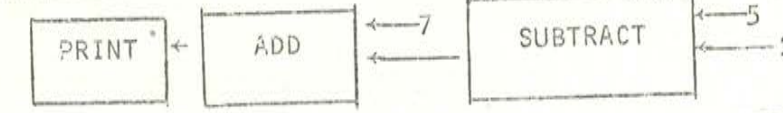
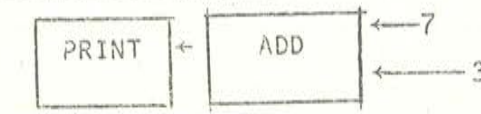

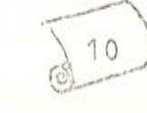
W: PRINT ADD 7 SUBTRACT 5 2

W: PRINT SUBTRACT 7 ADD 5 2

LOGO reads your command from left to right looking for the correct number of inputs for each procedure it finds. On the next page we have drawn a series of SNAPSHOTS from a movie of LOGO executing a SINGLE COMMAND. Remember a single command is just one line of typing.

In the snapshots we have used boxes to represent the workers. The arrows going into a box from the right are inputs. Any arrow coming out of a box on its left is its result. A box with no arrow coming out is a worker who produces an effect, e.g. like PRINT.

W: PRINT ADD 7 SUBTRACT 5 2

SNAPSHOT 1		LOGO tries to execute PRINT
SNAPSHOT 2		but PRINT needs an input
SNAPSHOT 3		LOGO tries to find the result of SUM as the input
SNAPSHOT 4		but ADD needs two inputs itself
SNAPSHOT 5		the first input is 7 but another is needed
SNAPSHOT 6		the result of SUBTRACT will be the second input
SNAPSHOT 7		but SUBTRACT itself needs two inputs
SNAPSHOT 8		5 is the first input for SUBTRACT
SNAPSHOT 9		and 2 is the second input
SNAPSHOT 10		SUBTRACT has enough inputs. Its result is 3
SNAPSHOT 11		ADD has its two inputs. Its result is 10
SNAPSHOT 12		PRINT has its input and has the effect of printing 10 at the teletype

EXERCISE 2: Draw a similar series of snapshots to show how the following command is executed:

W: PRINT SUBTRACT 7 ADD 5 2

Type in the following commands after working out what LOGO will print (can you work it out faster than LOGO does?)

W: PRINT ADD SUBTRACT 2 5 7

W: PRINT MULTIPLY ADD 2 5 7

W: PRINT ADD 2 MULTIPLY 5 7

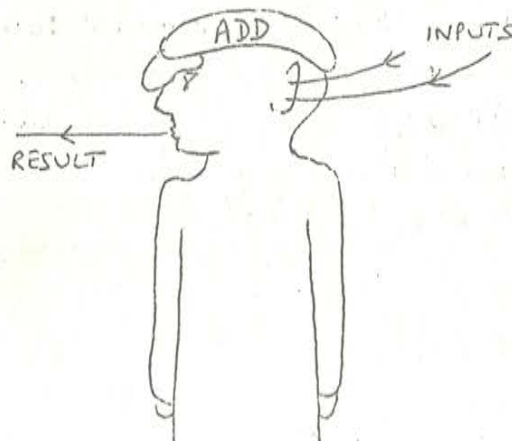
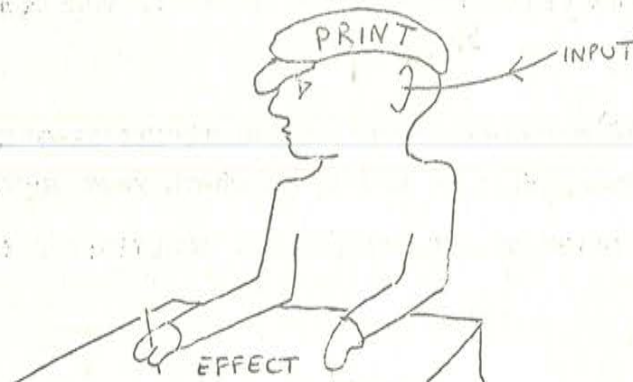
W: PRINT MULTIPLY 2 ADD 5 7

W: PRINT ADD MULTIPLY 2 5 7 9

W: PRINT ADD 1

W: PRINT ADD ADD ADD ADD 10 100 1000 10000 100000

Remember that there is a difference between procedures like PRINT which produce an EFFECT and those like ADD which give a RESULT.



EXERCISE 3: Try using a procedure which produces an effect, like FORWARD as if it gave a result

W: PRINT FORWARD 100

The input for FORWARD can be the result of some other procedure. Try

W: FORWARD ADD 100 200

Using BRACKETS

You can put in ROUND BRACKETS to help you see which inputs belong to which procedure. For example:

W: PRINT ADD (MULTIPLY 2 5) 7

W: PRINT ADD 2 (MULTIPLY 5 7)

Inside any matching pair of round brackets, (), you can put a procedure and its inputs. Any of these inputs could be the result of some other bracketed procedure with its inputs.

You can read LOGO round brackets in the same way as you read brackets in mathematics.

EXERCISE 4: Work out what the effect of the following command will be and then check your answer.

!!: PRINT ADD (MULTIPLY 3 4) (MULTIPLY 2 (ADD 1 5))

SUMMARY

LOGO reads commands from left to right. The command will often be an arrangement of procedures. Each procedure will look for inputs on its right and give results to the procedure on its left. The procedure on the extreme left will always be the last to be executed and should produce an effect rather than give a result.

11. SUPER-PROCEDURES AND SUB-PROCEDURES

Here is a single procedure which will have the effect of drawing a house when run.

```
W: DEFINE "HOUSE
  D: 10 SQUARE
  D: 20 LEFT 60
  D: 30 TRIANGLE
  D: END
```

EXERCISE 1: Define and run the procedure named HOUSE.

If you got an error message when you tried to run HOUSE it may have been because LOGO did not have a copy of the procedure SQUARE in its working memory.

EXERCISE 2: If you have a procedure SQUARE in permanent memory
 RECALL it to working memory.
 Otherwise define a suitable square procedure.
 Do the same for the procedure TRIANGLE.
 Now run HOUSE.

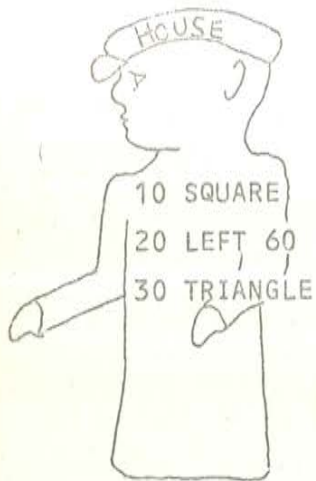
You may have to get rid of some problem bugs before the HOUSE procedure has the effect you want.

Such a procedure HOUSE which has your own procedures inside it is called a SUPER-PROCEDURE.

The procedures inside are called SUB-PROCEDURES.

The worker HOUSE supervises your own workers, SQUARE and TRIANGLE, as well as the LOGO procedure LEFT. This is done in just the same way as the procedure SPIKE supervised his workers in note 4.

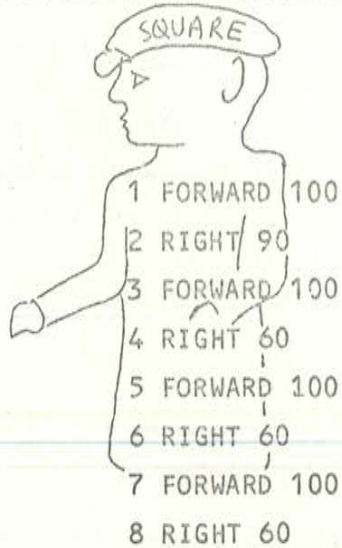
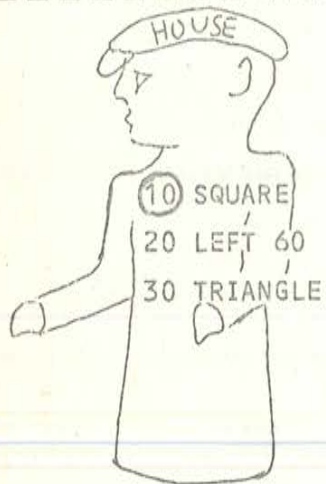
On the next page are drawn some snapshots from a movie of the procedure HOUSE being executed.



SNAPSHOT 1

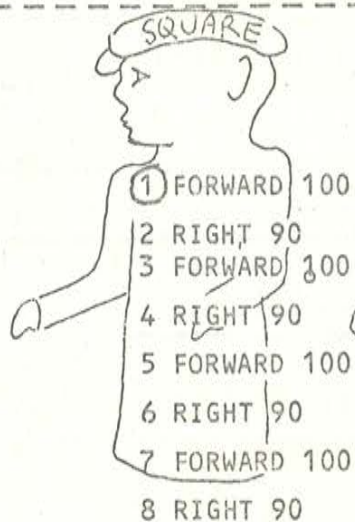
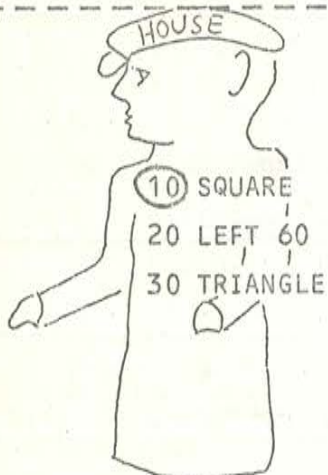
We run procedure HOUSE

W: HOUSE



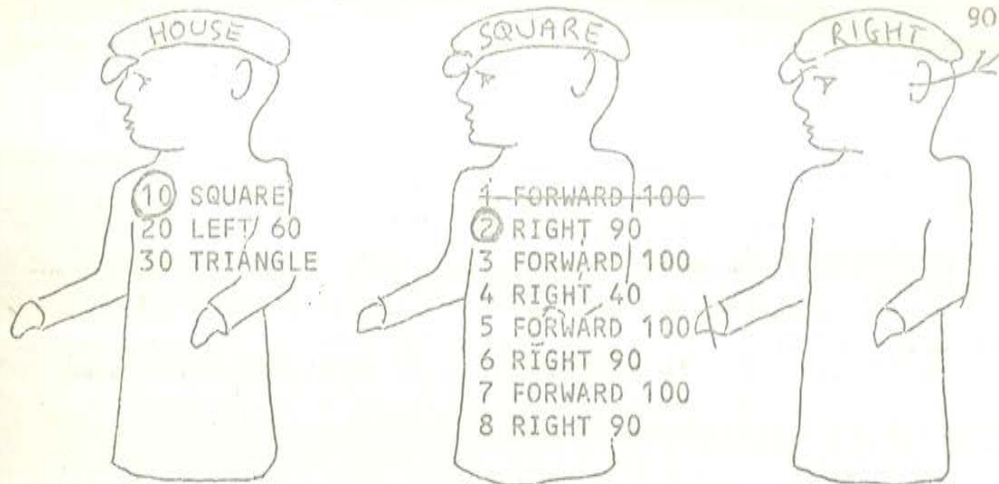
HOUSE calls on worker
SQUARE. HOUSE leaves
a marker on line 10

SNAPSHOT 2



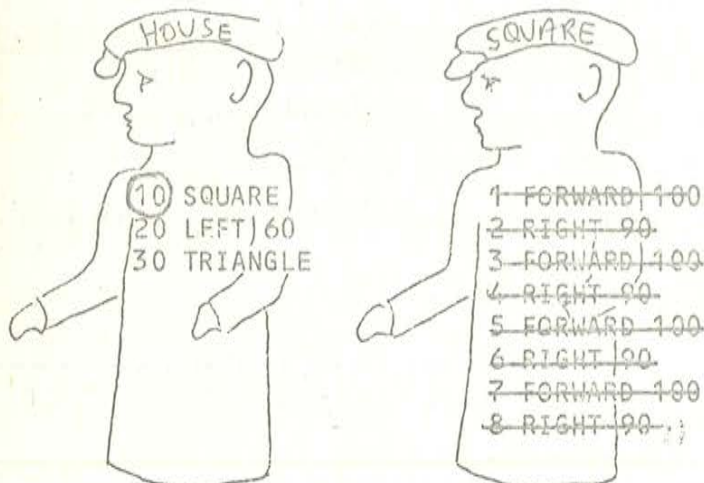
SQUARE calls on
FORWARD providing
him with his input
of 100. SQUARE
leaves a marker on
his line 1.

SNAPSHOT 3



SNAPSHOT 4

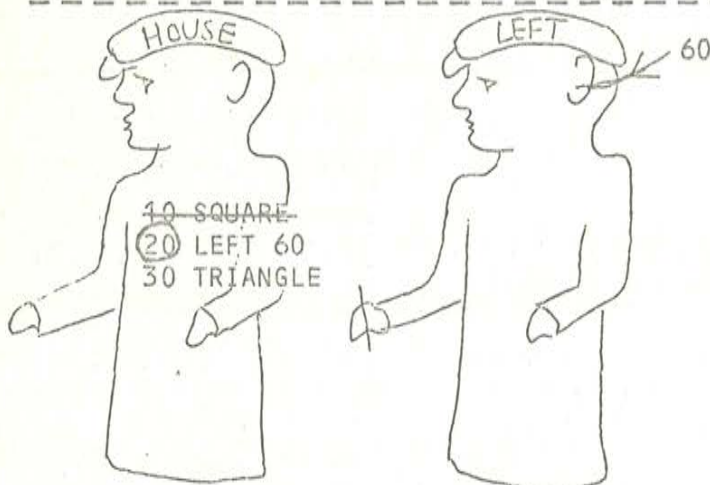
SQUARE next calls
on RIGHT giving him
90 as his input



eventually SQUARE
finishes all his
work. He reports:
to HOUSE that he
has finished.



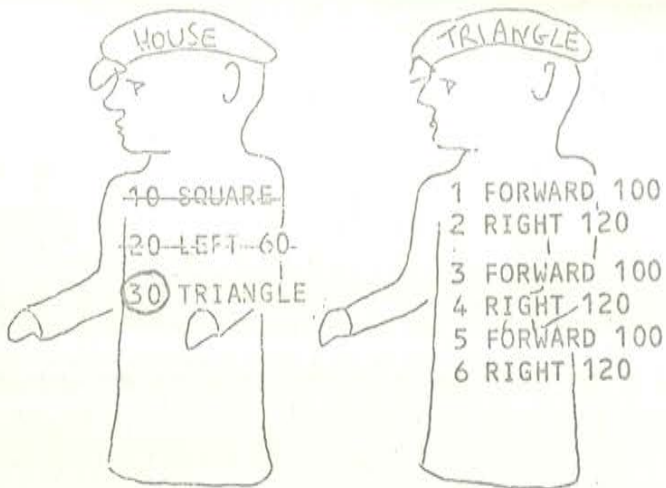
SNAPSHOT 5



HOUSE next calls
LEFT providing him
with his input of 60

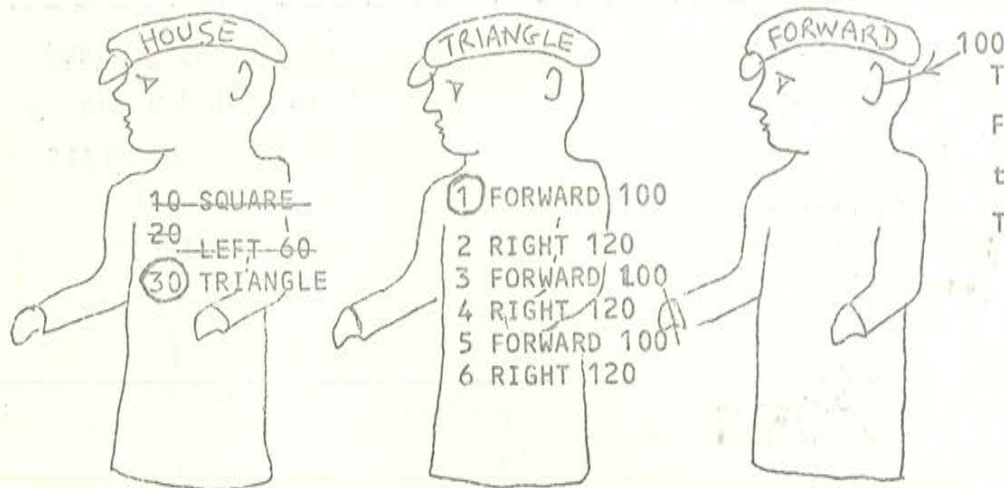


SNAPSHOT 6



SNAPSHOT 7

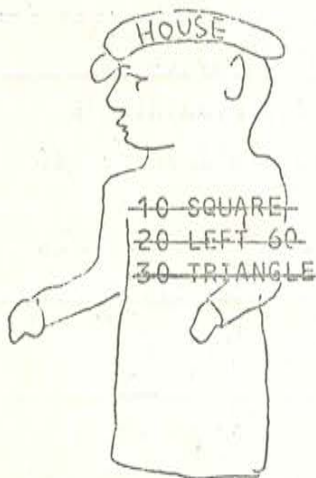
HOUSE next calls on TRIANGLE. HOUSE'S marker is now on line 30



TRIANGLE calls on FORWARD giving him the input 100
TRIANGLE marks line 1.



SNAPSHOT 8



Eventually TRIANGLE finishes all his work. He reports to HOUSE who thus also finishes. The drawing of the house is now complete.

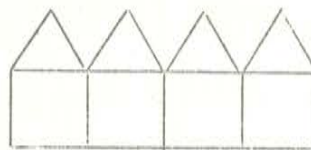


SNAPSHOT 9

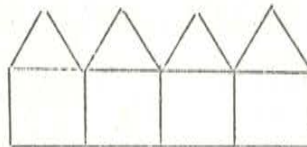
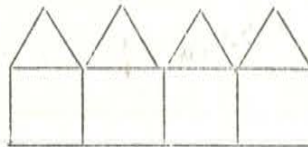
EXERCISE 3: Try to define a procedure named TOWER which uses SQUARE and TRIANGLE as sub-procedures.



Can you define a procedure STREET which uses HOUSE as a sub-procedure?



Can you define a procedure TOWN which uses STREET as a sub-procedure?

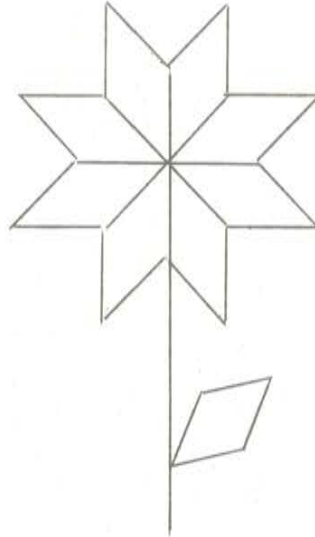


SUMMARY

A plastic LEGO set has a number of different building blocks. These can be assembled into a whole variety of different models.

In LOGO your sub-procedures can be assembled into a whole variety of different super-procedures.

12. BREAKING DOWN PROBLEMS



This picture, designed by a LOGO student could be drawn in one of three ways:

- a) By giving single drawing commands one after the other (about 100, count them). This has disadvantages: it is easy to lose track, if you make a mistake you have to start from the beginning again. If you want to draw the picture again you have to repeat the whole sequence of commands.
- b) By making one procedure of all the long sequence of commands. This has many of the disadvantages of method a), plus the fact that it is hard to debug.
- c) By breaking down the problem of drawing the whole picture into the smaller sub-problems of drawing parts of the picture. In this flower picture one sub-problem is that of drawing the diamond shape which appears nine times.

This last method has several advantages: the big problem of drawing the flower is broken down into smaller problems which will usually be easier to solve.

The different pieces of the flower can be debugged separately and corrected before putting them together to draw the whole flower.

These different pieces can be used to make other pictures.

The sub-procedures you write will be easier to debug because you will be able to match them to the different parts of the whole picture.

We are going to show you how to draw the flower by BREAKING DOWN THE LARGE PROBLEM INTO SMALLER SUB-PROBLEMS.

EXERCISE 1: Define and debug a procedure to draw a diamond.

Hint: you may find it useful to arrange for your procedure DIAMOND to leave the TURTLE in the same state as it found it!

The flower bloom can be made by writing a procedure which draws a petal and turns a bit, keeping going until it has drawn all the petals, as you might draw such a bloom using a stencil cut out in the shape of a diamond.

Using your procedure DIAMOND as a sub-procedure this is how one might define a procedure to draw the bloom:

W: DEFINE 'BLOOM

D: 10 DIAMOND

D: 20 RIGHT 45

D: 30 DIAMOND

D: 40 RIGHT 45

D: 50 DIAMOND

D: 60 RIGHT 45

D: 70 DIAMOND

D: 80 RIGHT 45

D: 90 DIAMOND

D: 100 RIGHT 45

D: 110 DIAMOND

D: 120 RIGHT 45

D: 130 DIAMOND

D: 140 RIGHT 45

D: 150 DIAMOND

D: 160 RIGHT 45

D: END

This is long-winded! It can be shortened because there is a repeated pattern to the commands:- diamond, right, diamond, right etc.

EXERCISE 2: Define the procedure which draws a diamond and turns right.

```
W: DEFINE "BLOOMBIT
D: 10 DIAMOND
D: 20 RIGHT 45
D: END
```

We could then change BLOOM so that it uses the sub-procedure BLOOMBIT.

```
W: DEFINE "BLOOM
D: 10 BLOOMBIT
D: 20 BLOOMBIT
D: 30 BLOOMBIT
D: 40 BLOOMBIT
D: 50 BLOOMBIT
D: 60 BLOOMBIT
D: 70 BLOOMBIT
D: 80 BLOOMBIT
D: END
```

This new version of BLOOM is shorter than before. But because we have named and defined a sub-procedure BLOOMBIT which we wish to have repeated eight times by BLOOM we can use a special LOGO procedure. The name of this procedure is

REPEAT

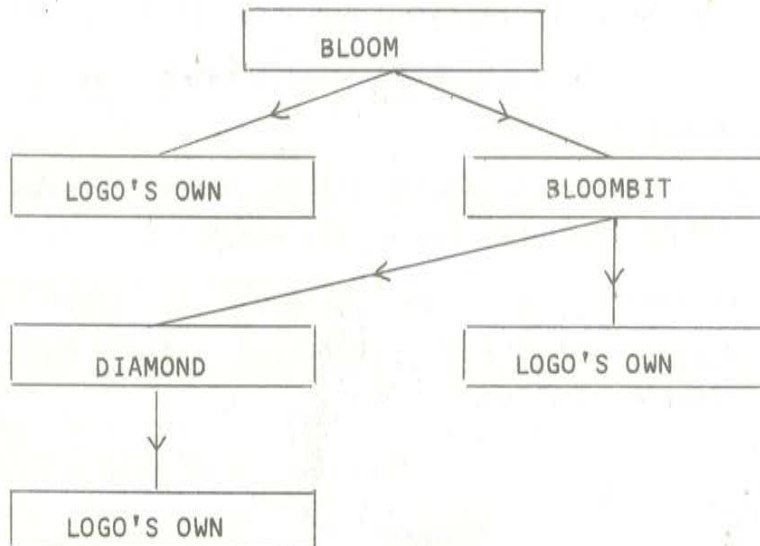
It needs two inputs. The first input must be a number. The second input must be a procedure to be repeated that number of times. A procedure like REPEAT which controls how a command is executed is called a CONTROL PROCEDURE.

EXERCISE 3: Try defining BLOOM as follows:-

```
W: DEFINE "BLOOM
D: 1 REPEAT 8 BLOOMBIT
D: END
```

run BLOOM to see how it draws the bloom.

The following diagram illustrates how the bloom is put together. The boxes represent your different procedures. Boxes marked LOGO'S OWN procedures contain only procedures which LOGO knows already. The arrows out of a box point to the sub-procedures it uses.



EXERCISE 4: Try defining a procedure for the stem of the flower.

For example:

```

W: DEFINE "STEM
  D: 10 FORWARD 160
  D: 20 LEAF.
  D: 30 FORWARD 40
  D: END
  
```

We have referred to a procedure LEAF which we have not yet defined. LOGO accepts this; but if you try to RUN the procedure STEM before LEAF is defined LOGO will not know how to execute LEAF and give you a message.

EXERCISE 5: Try to run STEM.

Now define a suitable LEAF procedure and run STEM again.

EXERCISE 6: When you have debugged your STEM and LEAF procedures assemble these with the sub-procedure BLOOM into a super-procedure FLOWER whose job it will be to draw the whole flower.

EXERCISE 7: Try changing FLOWER to put more leaves on the stem.

EXERCISE 8: Try writing a procedure GARDEN which draws a row of flowers.

EXERCISE 9: Draw a diagram showing all the sub-procedures of FLOWER like our diagram for BLOOM.

EXERCISE 10: REMEMBER all your new procedures. (FLOWER cannot work without its sub-procedures. These must be remembered as well).

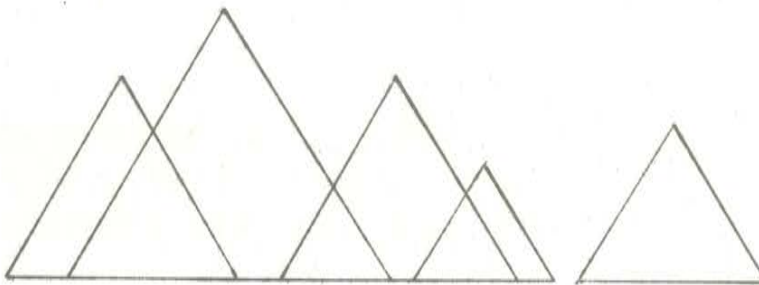
SUMMARY

Problems can be dealt with by breaking them down into sub-problems and writing procedures to solve each of these sub-problems.

The new procedure is:

NAME OF PROCEDURE	INPUTS	EFFECT
REPEAT	number, procedure	a control procedure to repeat the execution of its second input a number of times

13. PROCEDURES WITH INPUTS (Part 1)

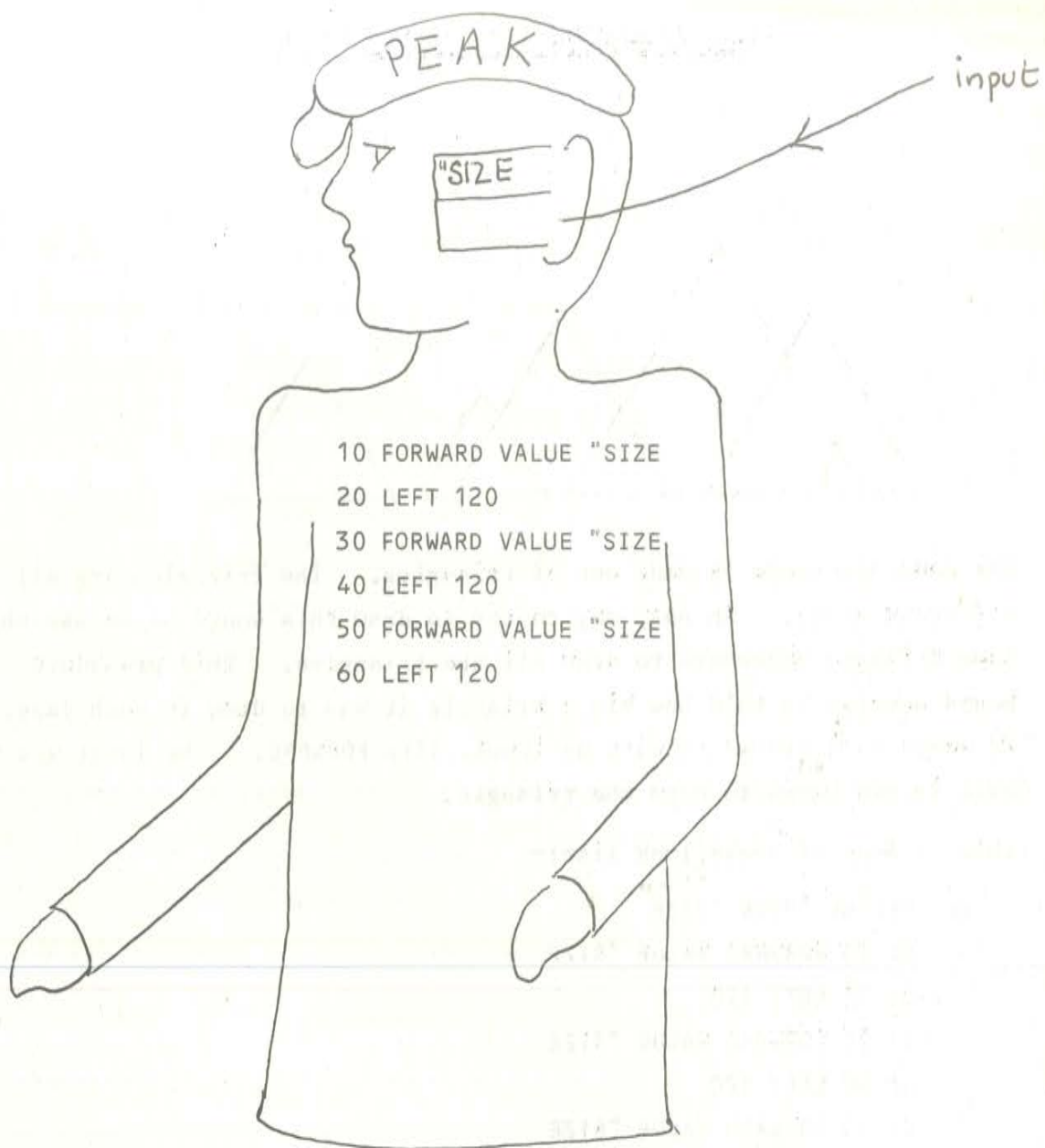


The mountain range is made out of triangles. The triangles are all different sizes. An easy way to try to draw this would be to use the same triangle procedure to draw all the triangles. This procedure would have to be told how big a triangle it was to draw in each case. It would be a procedure with an input, like FORWARD. The input would tell it how large to draw the triangle.

This is what it could look like:-

```
W: DEFINE "PEAK "SIZE
  D: 10 FORWARD VALUE "SIZE
  D: 20 LEFT 120
  D: 30 FORWARD VALUE "SIZE
  D: 40 LEFT 120
  D: 50 FORWARD VALUE "SIZE
  D: 60 LEFT 120
  D: END
```

The title line of the procedure contains the name of the procedure and the name of one input. Procedure "PEAK will now expect a value for its input, named "SIZE, when it is run. We can choose any LOGO word to be the name of the input.



EXERCISE 1: Type in the definition of PEAK

EXERCISE 2: Run the procedure PEAK with different inputs e.g.

W: PEAK 10

W: PEAK 180

EXERCISE 3: Run the procedure with no input

W: PEAK

EXERCISE 4: Run the procedure with two inputs

W: PEAK 10 180

Run the procedure with a word input, rather than a number

W: PEAK "BIG

Your procedures can have more than one input. Here is a procedure which will write thank-you letters. The procedure needs three inputs, the person you are thanking, the present you got and what you used the present for.

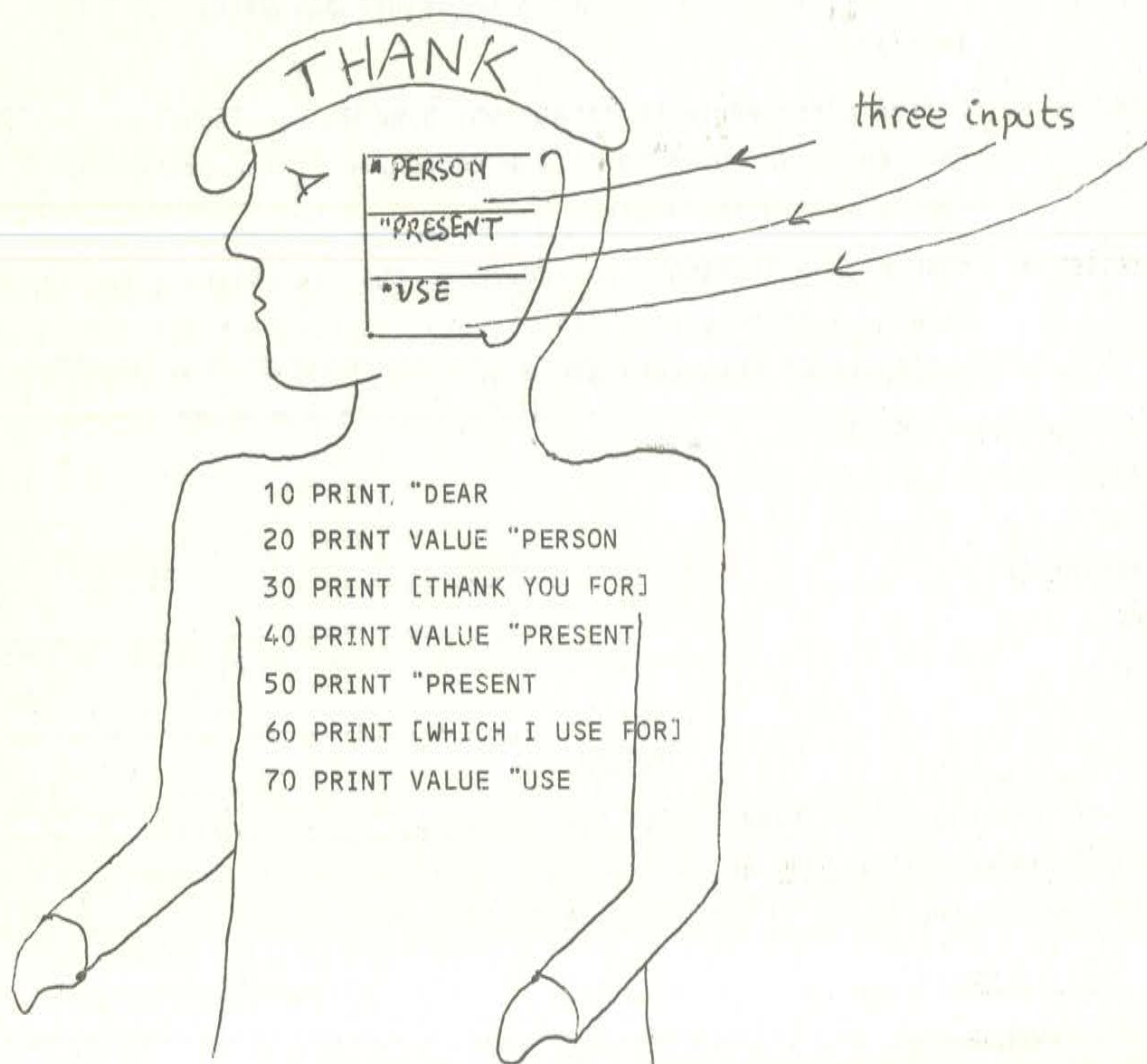
```
W: DEFINE "THANK "PERSON "PRESENT "USE
D: 10 PRINT "DEAR
D: 20 PRINT VALUE "PERSON
D: 30 PRINT [THANK YOU FOR THE]
D: 40 PRINT VALUE "PRESENT
D: 50 PRINT "PRESENT
D: 60 PRINT [WHICH I USE FOR]
D: 70 PRINT VALUE "USE
D: END
```

EXERCISE 5: Type in this definition and then run the procedure with three inputs e.g.

```
W: THANK "GRANNY "WATCH [TELLING THE TIME]
```

```
W: THANK [UNCLE JIM AND AUNTIE MARY] "COMPUTER [SITTING ON]
```

A list is one input.



Both PEAK and THANK use a new LOGO procedure named

VALUE

VALUE needs one input. To show how VALUE works we look at lines 40 and 50 of procedure THANK

```
40 PRINT VALUE "PRESENT
50 PRINT "PRESENT
```

When line 50 is executed the word "PRESENT is printed. When line 40 is executed, procedure VALUE assumes the word "PRESENT is a NAME. In this case "PRESENT is the NAME of the second input for THANK.

So VALUE takes the second number word or list given to THANK when it is run. VALUE gives this number, word or list to PRINT.

EXERCISE 6: Try running THANK with a variety of inputs e.g.

```
W: THANK "DEAR "LOVELY [TARGET PRACTICE]
W: THANK [TELLING THE TIME] "GRANNY "WATCH
W: THANK "ZZZZ 842 75
```

EXERCISE 7: Define a procedure INVITE which prints out party invitations

EXERCISE 8: Define a procedure RECTANGLE which needs two inputs, the length and breadth of the rectangle. The procedure should draw a rectangle.

EXERCISE 9: CHANGE your RECTANGLE procedure so that in addition to drawing a rectangle it also prints out the area and perimeter of the rectangle e.g.

```
W: RECTANGLE 8 3
AREA
24
PERIMETER
22
W:
```

SUMMARY

Your own procedures can have inputs just like LOGO'S procedures. Each input is named with a LOGO word. VALUE is used to get the number, word or list named by the input. The new procedure is:

<u>Name of procedure</u>	<u>input</u>	<u>result</u>
VALUE	quoted LOGO word	fetches value named by word

14. PROCEDURES WITH INPUTS (Part 2)

In the last note you defined a procedure PEAK which needed one input. Such a procedure can be used as a sub-procedure like any other. Here is a super-procedure which uses PEAK as a sub-procedure.

```
W: DEFINE "MOUNTAINS
  D: 10 PEAK 150
  D: 20 PEAK 70
  D: 30 PEAK 20
  D: 40 PEAK 95
  D: END
```

EXERCISE 1: Define a procedure RANGE which draws a mountain range using PEAK as a sub-procedure

EXERCISE 2: Define a super-procedure CONIFER which draws



(Hint: a useful sub-procedure would be ARROW which draws ↗ different sizes)

SUMMARY

Your procedures with inputs can be sub-procedures.

15. CHANGING PROCEDURES (Part 2)

There are various reasons for changing the title line of a procedure, e.g.

1. To change its name.
2. To change the number of inputs the procedure has.

EXERCISE 1: Define and run a procedure which prints out a message, e.g.

```
W: DEFINE "HAPPY
    D: 10 PRINT [GOOD MORNING]
    D: 20 PRINT [WHAT A LOVELY DAY]
D: END
```

This procedure can be changed so that it greets a particular person by name. First we must put LOGO back in the DEFINING STATE and make the changes we want, for example

EXERCISE 2: Give your procedure an INPUT which it can use.

For HAPPY we would type

```
W: CHANGE "HAPPY
    D: RETITLE "HAPPY "WHO
    D: 15 PRINT VALUE "WHO
D: END
```

EXERCISE 3: SHOW your changed procedure.

EXERCISE 4: Run your changed procedure.

We have used a new procedure named

RETITLE

This procedure can only be run when LOGO is in the defining state. RETITLE does the following with its INPUTS which must be LOGO words. The first word is used to make a new name for the procedure. Any other words become the names of the inputs for this changed procedure.

EXERCISE 5: Give your procedure a different name and make it print a different message.

For example, we could change HAPPY to MISERY and it could run as follows:-

W: MISERY "FRED

GOOD MORNING

FRED

ITS RAINY AND FOGGY AGAIN TODAY

W:

SUMMARY

NAME OF PROCEDURE	INPUTS	EFFECT
RETITLE	New procedure name and new input names	Changes <u>TITLE LINE</u>

16. TWO MEMORIES (Part 2)

There is a procedure named

DEFINED

to help you keep track of what is in your working memory. This procedure needs no input.

EXERCISE 1: Try typing
W: PRINT DEFINED

The result of DEFINED is a list of the names of all the procedures in your section of working memory.

There is a similar procedure named

REMEMBERED

whose result is a list of all the names of your procedures in permanent memory.

EXERCISE 2: Try typing
W: PRINT REMEMBERED

REMEMBERED can also be used if you want to copy all your procedures from permanent memory to working memory in one go.

EXERCISE 3: Try typing
W: RECALL REMEMBERED

Notice that RECALL is able to take either a list of procedure names as an input or a single procedure name.

EXERCISE 4: Try typing
W: PRINT DEFINED

The names of all the procedures now in working memory are printed.

DEFINED can also be used at the end of a LOGO session if you want to copy all the procedures in working memory to permanent memory.

EXERCISE 5: Try typing
W: REMEMBER DEFINED

Notice that REMEMBER like RECALL can also have a list as input.

You may want to copy procedures from somebody else's permanent memory. You have to run a procedure named

BORROW

BORROW needs one input. This must be a list containing the name of the person whose procedure you want to copy.

EXERCISE 6: Try typing
W: BORROW [TIM OSHEA]

Now you have been connected to TIM OSHEA's permanent memory.

EXERCISE 7: Try typing
W: RECALL REMEMBERED

and run any of the procedures you get copies of.

EXERCISE 8: Try and REMEMBER one of the procedures in your working memory.

You will get a message telling you that you cannot do this. This is because you are not allowed to remember procedures in other people's permanent memories!

To get connected back to your own permanent memory (and disconnected from TIM OSHEA's) run the procedure

RETURN

which needs no inputs.

EXERCISE 9: Try typing
W: RETURN

EXERCISE 10: Use BORROW and RETURN to put a copy of somebody-else's procedure in your permanent memory.

SUMMARY

NAME OF PROCEDURE	INPUT	RESULT	EFFECT
DEFINED	none	list of procedure names	none
REMEMBERED	none	list of procedure names	none
REMEMBER	word or list	none	copies procedures into permanent memory.
RECALL	word or list	none	copies procedures into working memory.
BORROW	list	none	connects you to another permanent memory.
RETURN	none	none	connects you back to your own permanent memory.

17. PROCEDURES WITH INPUTS (Part 3)

In notes 13 and 14 we showed you how to define a procedure which takes variable inputs. Such procedures could be used as sub-procedures.

EXERCISE 1: RECALL your procedure PEAK, which draws variable sized triangles.
Show the procedure PEAK

It should be

```
DEFINE "PEAK "SIZE
  10 FORWARD VALUE "SIZE
  20 LEFT 120
  30 FORWARD VALUE "SIZE
  40 LEFT 120
  50 FORWARD VALUE "SIZE
  60 LEFT 120
  END
```



hat

EXERCISE 2: Define a procedure HAT which uses PEAK as a sub-procedure. It should draw a hat.

One solution might be

```
W: DEFINE "HAT
  D: 10 FORWARD 20
  D: 20 PEAK 100
  D: 30 FORWARD 100
  D: 40 FORWARD 20
  D: END
```

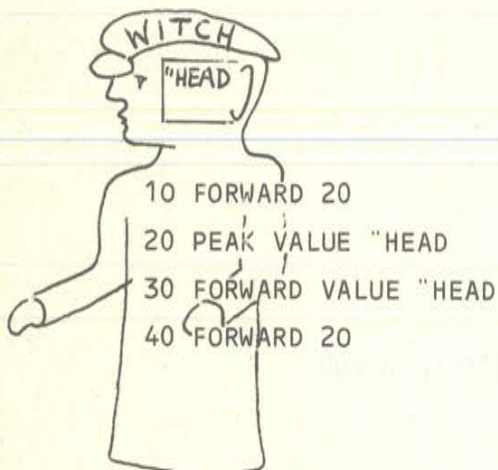

However this HAT only fits one size of head. The procedure HAT can itself have a variable input. The worker HAT will have to tell the worker PEAK about the value of its input so that PEAK draws the right size of triangle.

EXERCISE 3: Define this procedure which can draw hats of any size. We have based it on our original procedure HAT.

```
W: DEFINE "WITCH "HEAD
  D: 10 FORWARD 20
  D: 20 PEAK VALUE "HEAD
  D: 30 FORWARD VALUE "HEAD
  D: 40 FORWARD 20
  D: END
```

Run WITCH with different value inputs.

The following snapshots are from a movie of the worker WITCH supervising its other workers.

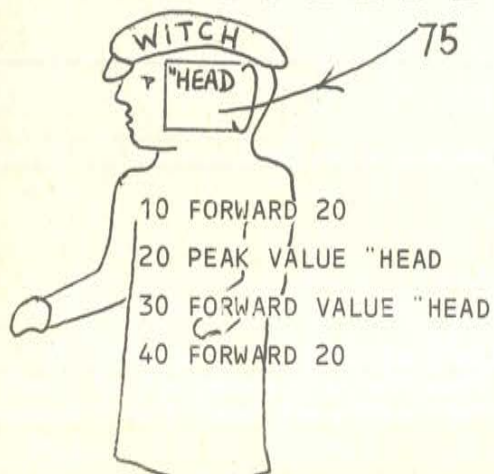


We run WITCH

W: WITCH 75

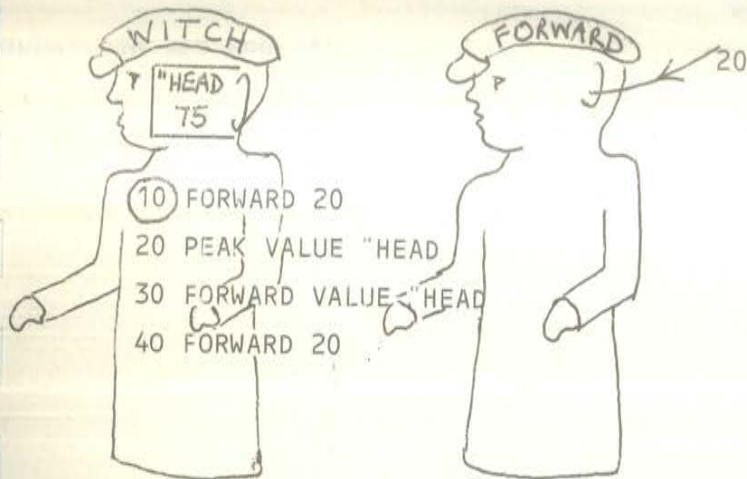
The worker WITCH is called

SNAPSHOT 1



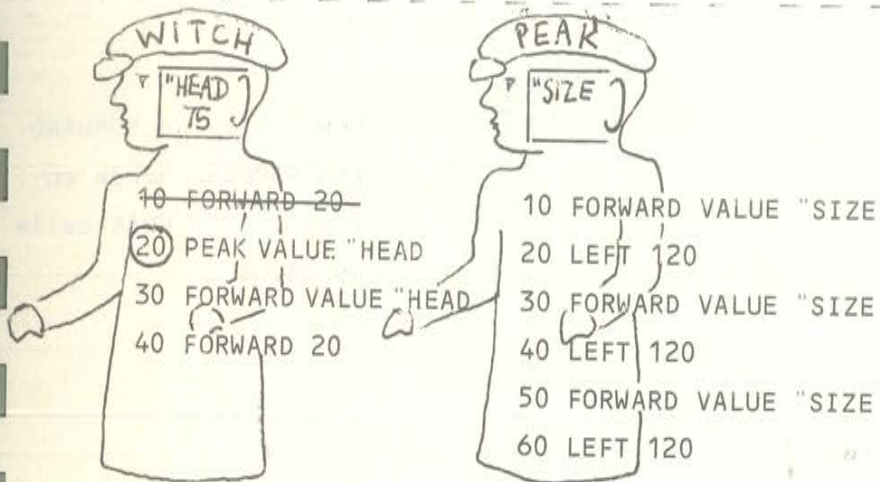
but WITCH needs one input which it gets from the line which called it. As far as WITCH is concerned the name of its input is "HEAD and its value is 75.

SNAPSHOT 2



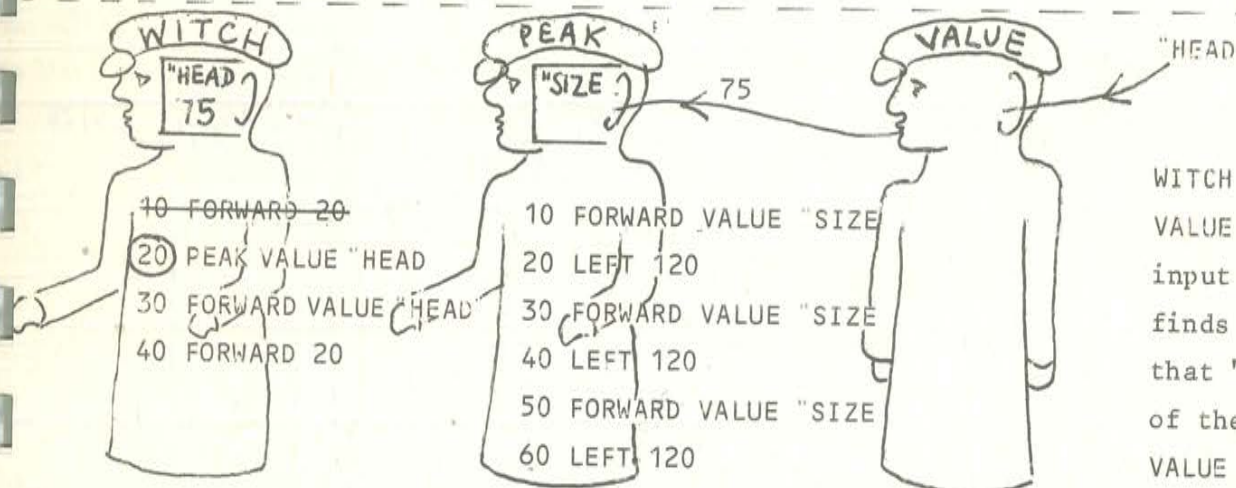
WITCH calls up FORWARD who gets his input 20 from the line in WITCH which called him

SNAPSHOT 3



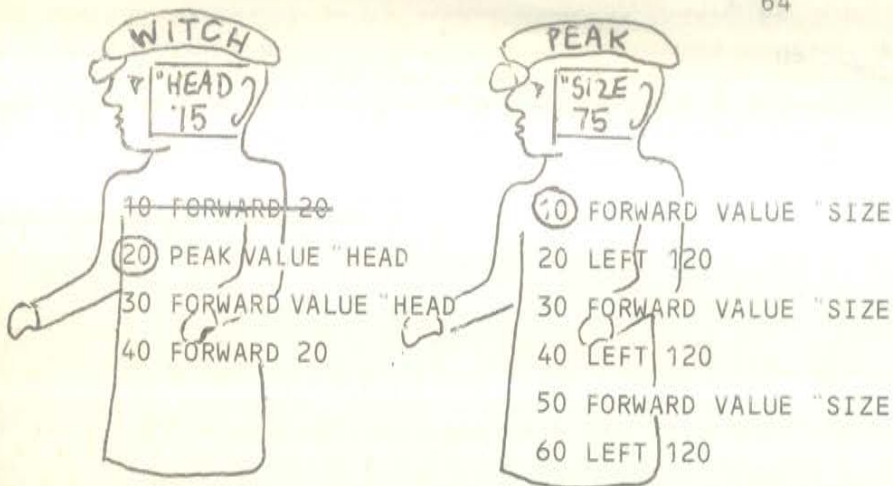
WITCH then reaches his line 20. He calls up PEAK. But PEAK needs one input.

SNAPSHOT 4



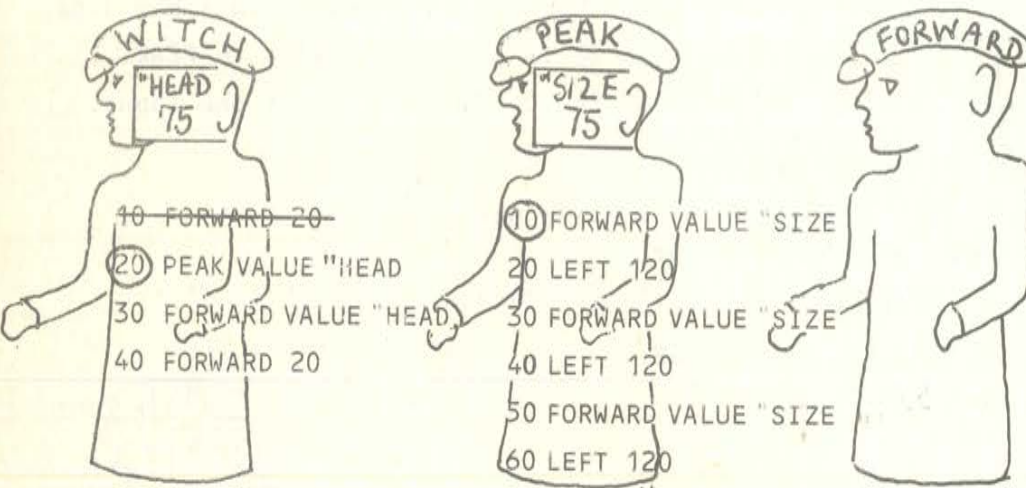
WITCH then calls up VALUE who get his input "HEAD". VALUE finds out from WITCH that "HEAD" is the name of the value 75. VALUE gives 75 as his result to PEAK.

SNAPSHOT 5



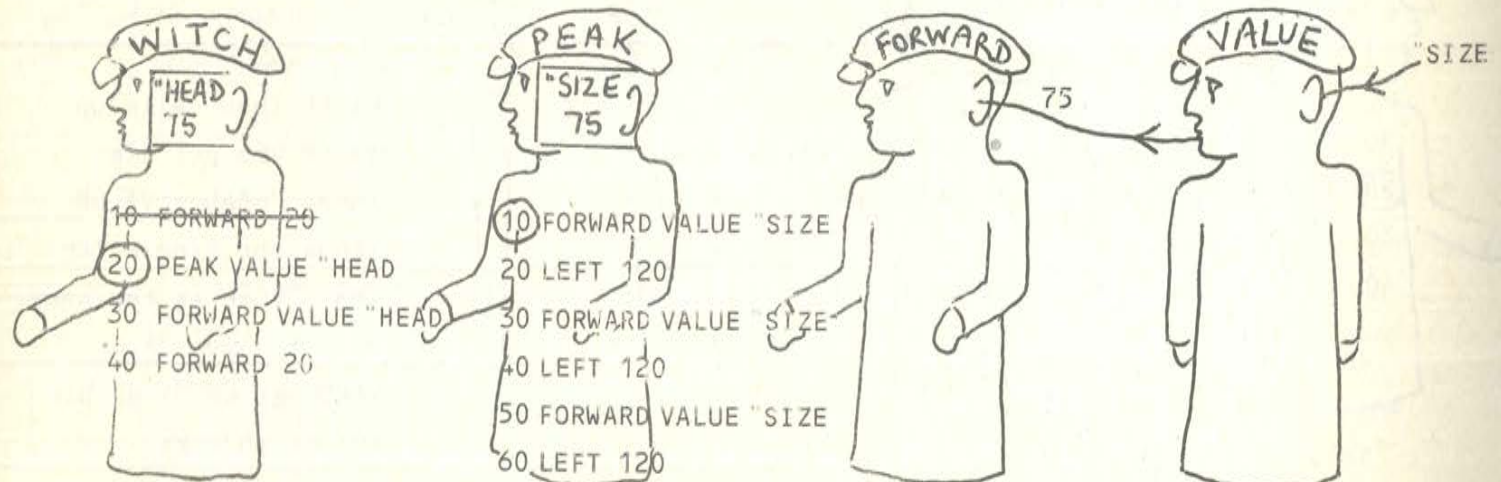
PEAK now has his input named "SIZE" which has the value 75. He is ready to execute his first line.

SNAPSHOT 6



PEAK calls on FORWARD. But FORWARD needs an input. So PEAK calls up VALUE.

SNAPSHOT 7



SNAPSHOT 8

So VALUE finds out from PEAK that "SIZE" is the name of the value 75. This is given as the input to FORWARD. The rest of the procedures are executed in turn until WITCH finishes.

EXERCISE 4: RECALL your HOUSE procedure.
Change HOUSE and its subprocedures so that
you can draw any size of HOUSE
e.g.

W: HOUSE 10



W: HOUSE 50



SUMMARY

A super-procedure which expects an input can tell any of its sub-procedures about the value of that input.

18. POLYGONS

The following procedure can be used to draw pentagons

```
W: DEFINE "BASIC
  D: 10 FORWARD 100
  D: 20 LEFT 72
  D: END
```

EXERCISE 1: Define BASIC and run it five times
e.g.

```
W: BASIC
W: BASIC
W: BASIC
W: BASIC
W: BASIC
```

or by running the control procedure REPEAT

```
W: REPEAT 5 BASIC
```

Running BASIC draws a pentagon because line 20 is a turn of 72 degrees.
If we used a different angle we would get a different polygon.

EXERCISE 2: Change line 20 of BASIC so that it can be used
to draw an octagon.

If we make the angle used in line 20 an input, BASIC would be used to
draw many different polygons.

EXERCISE 3: Change BASIC (using RETITLE) so that

```
W: SHOW "BASIC
types out
```

```
  DEFINE "BASIC "TURN
    10 FORWARD 100
    20 LEFT VALUE "TURN
  END
```

EXERCISE 4: Use BASIC to draw a lot of different polygons
e.g.

W: REPEAT 4 BASIC 90

W: REPEAT 24 BASIC 15

We can make a procedure to draw these polygons for us. It will need two inputs, the angle and the number of sides.

EXERCISE 5: Define the following procedure which has a very short name.

W: DEFINE "Z "ANGLE "NUMBER

D: 10 REPEAT VALUE "NUMBER BASIC VALUE "ANGLE

D: END

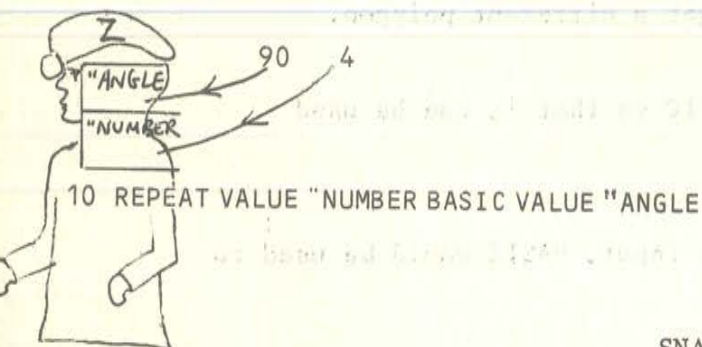
EXERCISE 6: Run Z to draw more polygons

e.g.

W: Z 4 90

W: Z 90 4

The following diagrams show snapshots from the movie of the worker Z supervising the worker BASIC

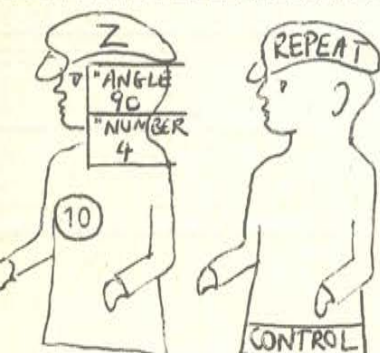


We run Z

W: Z 90 4

Z gets its two inputs. Its first input named "ANGLE has the value 90. Its second input named "NUMBER has the value 4

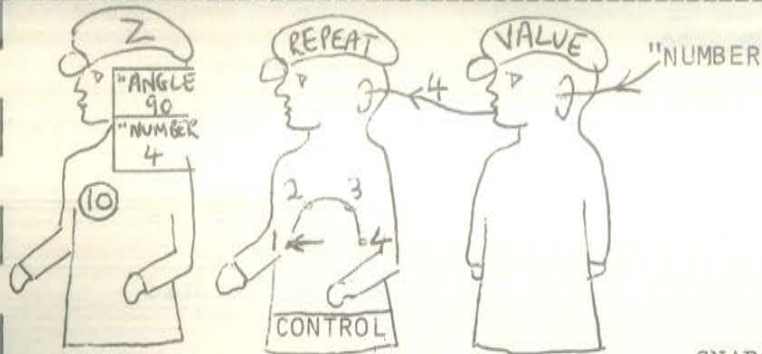
SNAPSHOT 1



Z starts to execute line 10 and calls REPEAT which needs two inputs. So Z calls more workers.

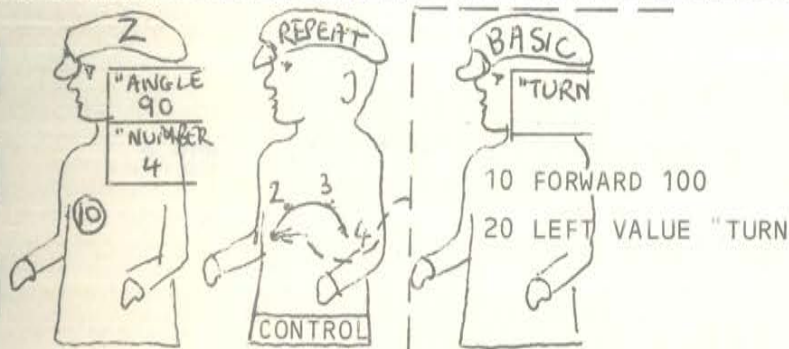
(We have not written out Z's line 10 to save space.)

SNAPSHOT 2



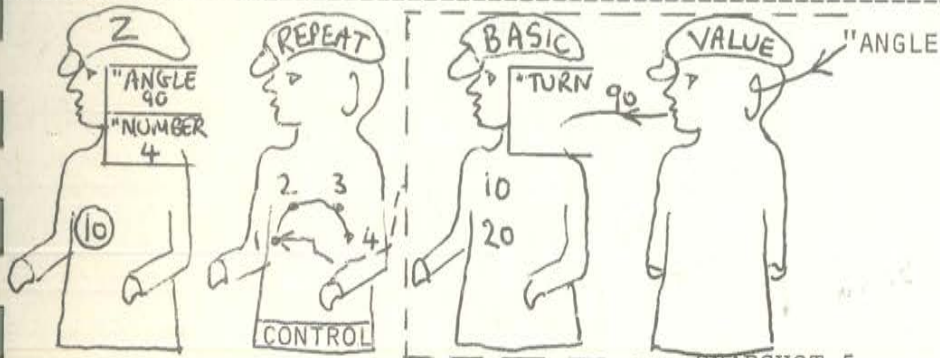
SNAPSHOT 3

VALUE is called. VALUE finds out from Z that the value of "NUMBER is 4. This is given to the control procedure REPEAT as its first input. REPEAT knows that he will be repeating four times, but he needs to know what to repeat.



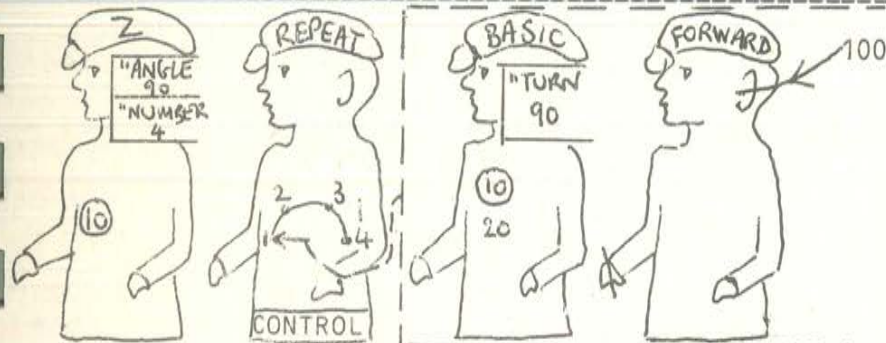
SNAPSHOT 4

The procedure to be repeated is BASIC but BASIC needs one input. So Z calls another worker.



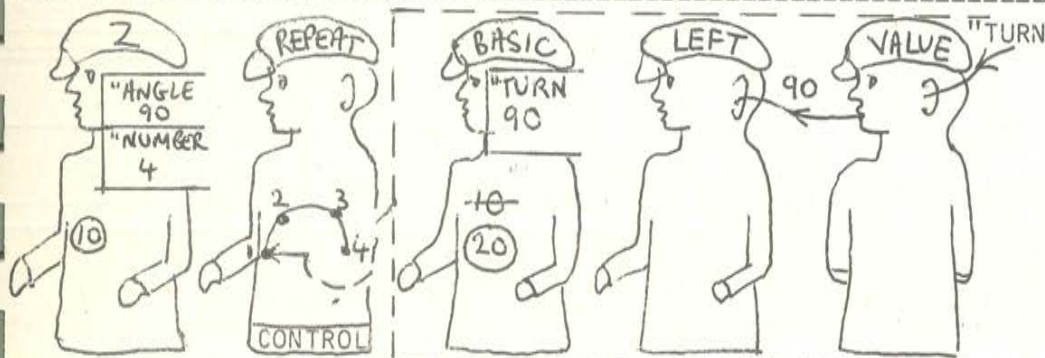
SNAPSHOT 5

VALUE finds out from Z that the value of "ANGLE is 90. This is given to BASIC. BASIC takes the input value 90. As far as he is concerned his input is named "TURN.



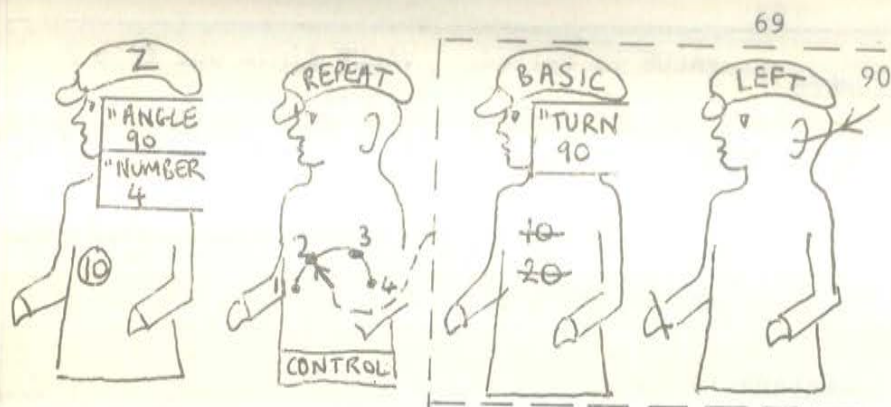
SNAPSHOT 6

BASIC can now start work. BASIC calls FORWARD who gets his input 100 from line 10 in BASIC. FORWARD draws a line.



SNAPSHOT 7

BASIC now executes line 20. He calls up two workers LEFT and VALUE. VALUE finds out from BASIC that the value of "TURN is



SNAPSHOT 8

LEFT can now rotate the turtle.

BASIC has been executed once.

The control procedure REPEAT repeats the process shown in SNAPSHOTS 6, 7, and 8 three more times.

SUMMARY

A procedure can pass the value of its input to a sub-procedure.

19. PROCEDURES WITH RESULTS

Try using a procedure which has an EFFECT as if it produced a RESULT, e.g. the procedure FORWARD.

EXERCISE 1: Try typing
W: PRINT FORWARD 100

So far all your own procedures have been defined for their EFFECTS, e.g. on the drawing devices or the teletype.

EXERCISE 2: Try typing
W: PRINT ADD 7 3

The procedure ADD passes its result to PRINT.

If we want to specify exactly what the RESULT of one of our own procedures is we use the LOGO procedure named

RESULT

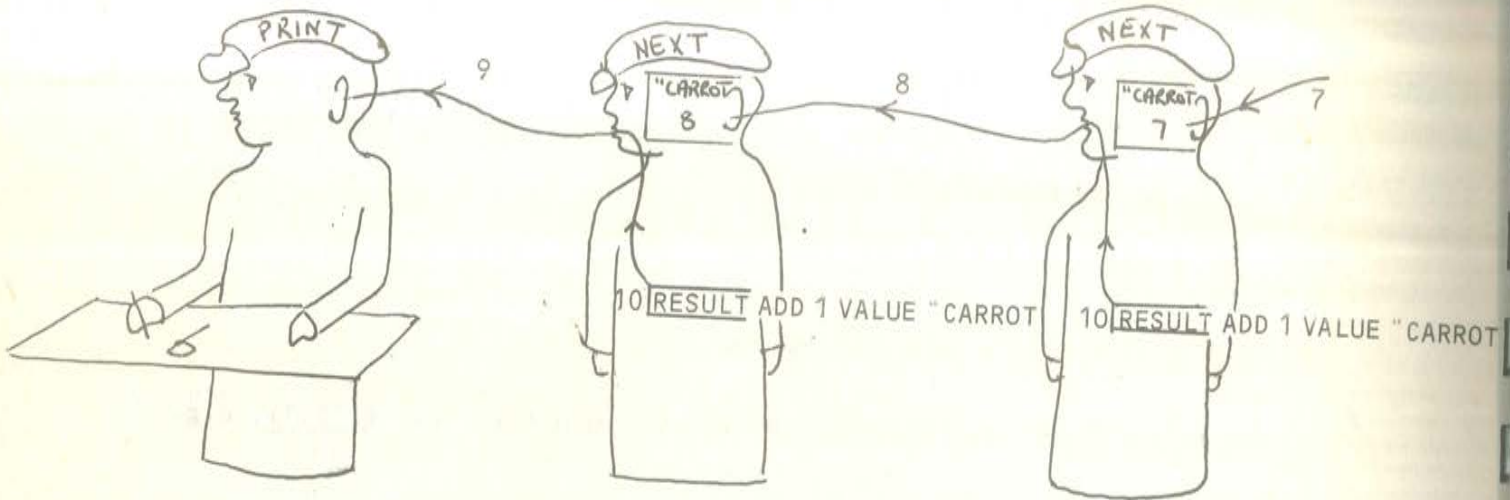
This procedure has one input. When RESULT is executed in one of your procedures, the value of its input becomes the RESULT of the whole procedure.

EXERCISE 3: Define the following procedure
W: DEFINE "NEXT "CARROT
D: 10 RESULT ADD 1 VALUE "CARROT
D: END

EXERCISE 4: Run NEXT e.g.
W: PRINT NEXT 7
W: PRINT NEXT NEXT 7

This diagram shows how the workers NEXT talk to PRINT.

W: PRINT NEXT NEXT 7



We have not shown the workers RESULT, ADD and VALUE who were also called.

EXERCISE 5: Define a procedure which doubles its input.
It should work as follows:
W: PRINT DOUBLE 50
100

EXERCISE 6: Define and run the following procedure:
W: DEFINE "SUMANDIFF "X "Y
D: 10 RESULT ADD VALUE "X VALUE "Y
D: 20 RESULT SUBTRACT VALUE "X VALUE "Y
D: 30 PRINT "FINISHED
D: END

Your procedures, like LOGO's, can only have one result. So LOGO stops executing a procedure after executing the first RESULT it finds.

RESULT is a CONTROL PROCEDURE. That is why lines 20 and 30 did not get executed.

EXERCISE 7: Define a procedure called SQUNUM which has one input. Its result should be the square of its input, e.g.

W: PRINT SQUNUM 9

81

EXERCISE 8: Try typing

W: PRINT SQUNUM SQUNUM SQUNUM SQUNUM 2

SUMMARY

Your own procedure can have a RESULT. The new procedure is

NAME OF PROCEDURE	INPUT	RESULT	EFFECT
RESULT	word, number or list	the same as its input	a control-procedure which stops your procedure and makes it give a result

20. RECURSION

A procedure can have a copy of itself as a sub-procedure.

```
W: DEFINE "LAUGH
    D: 10 PRINT "HAHA
    D: 20 PRINT "HOHO
    D: 30 LAUGH
    D: 40 PRINT "HEHE
    D: 50 PRINT [PLEASE STOP TICKLING ME]
    D: END
```

EXERCISE 1: Run this procedure.

To INTERRUPT LOGO executing a procedure, press the RED EMERGENCY STOP BUTTON. LOGO will type

INT:

Then to put LOGO back in the waiting state type Q followed by two presses on the green command button. LOGO will return to the waiting state

EXERCISE 2: Interrupt LAUGH and return LOGO to the waiting state.

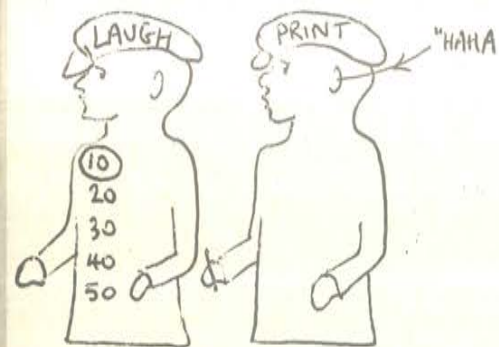
Lines 40 and 50 of LAUGH never get executed. Each procedure LAUGH, when it reaches line 30 calls for the execution of a sub-procedure LAUGH. The following snapshots of the execution of LAUGH illustrate this:-



We run LAUGH

W: LAUGH

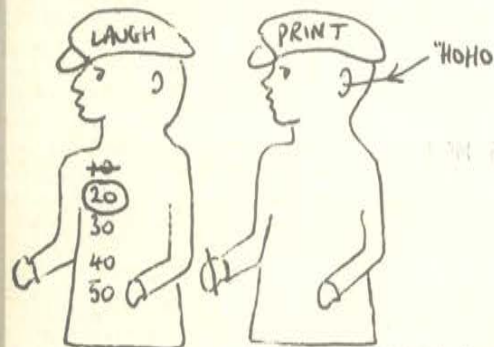
SNAPSHOT 1.



LAUGH calls PRINT

HAHA

SNAPSHOT 2

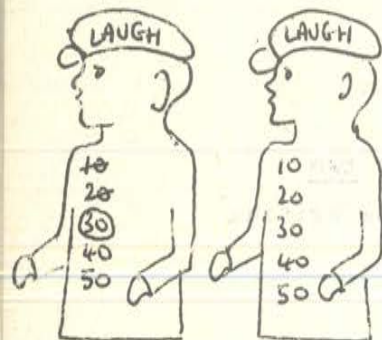


LAUGH calls another PRINT

HAHA

HOHO

SNAPSHOT 3

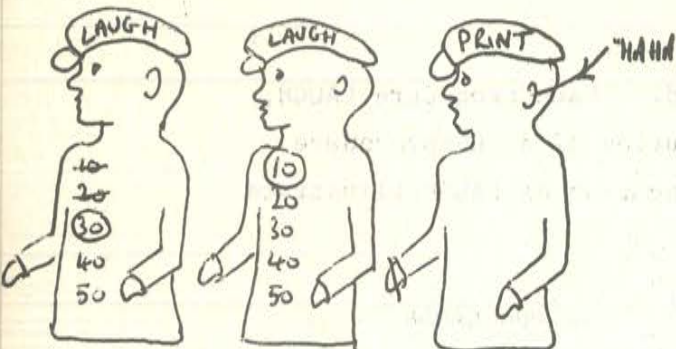


LAUGH calls another LAUGH

HAHA

HOHO

SNAPSHOT 4



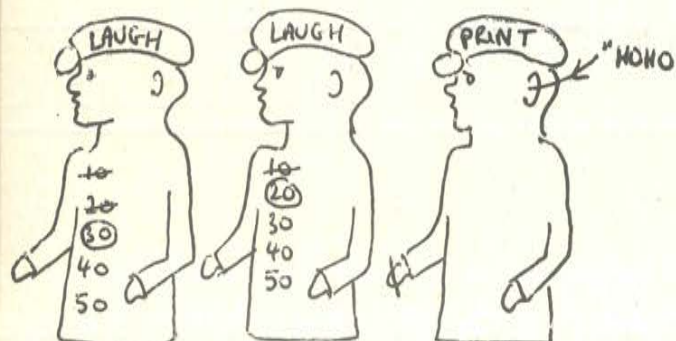
The new LAUGH calls PRINT

HAHA

HOHO

HAHA

SNAPSHOT 5

The new LAUGH calls another
PRINT

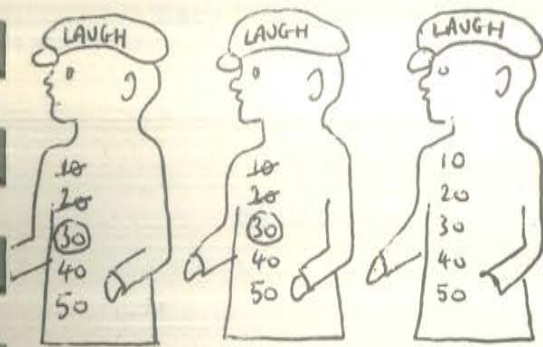
HAHA

HOHO

HAHA

HOHO

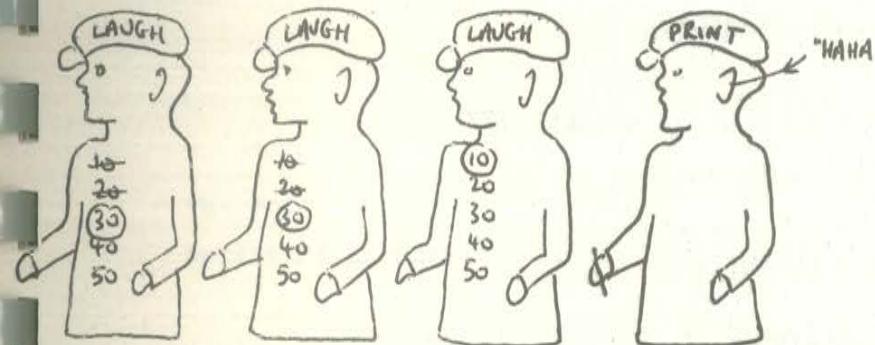
SNAPSHOT 6



The new LAUGH calls yet another
LAUGH

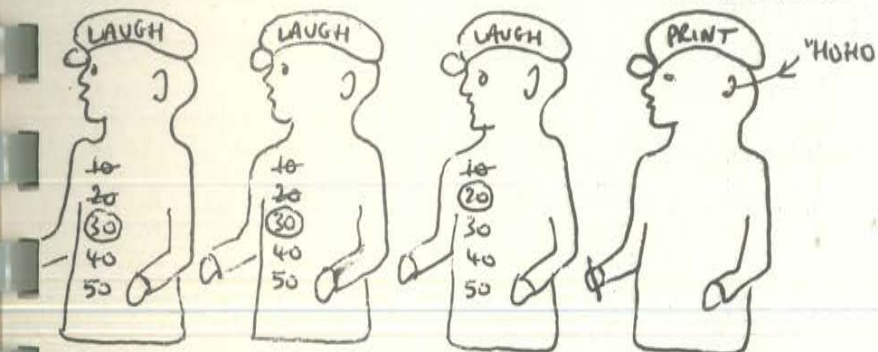
HAHA
HOHO
HAHA
HOHO

SNAPSHOT 7



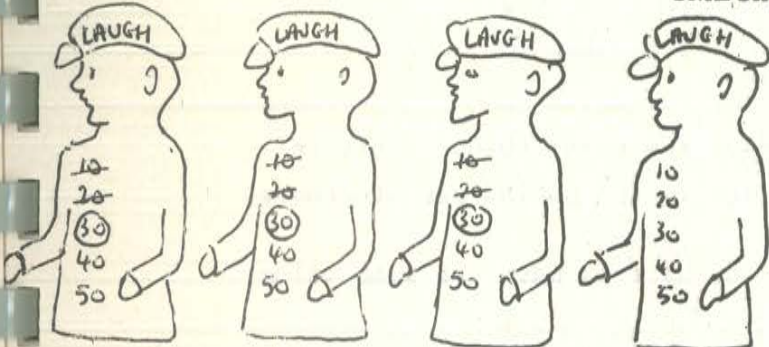
HAHA
HOHO
HAHA
HOHO
HAHA

SNAPSHOT 8



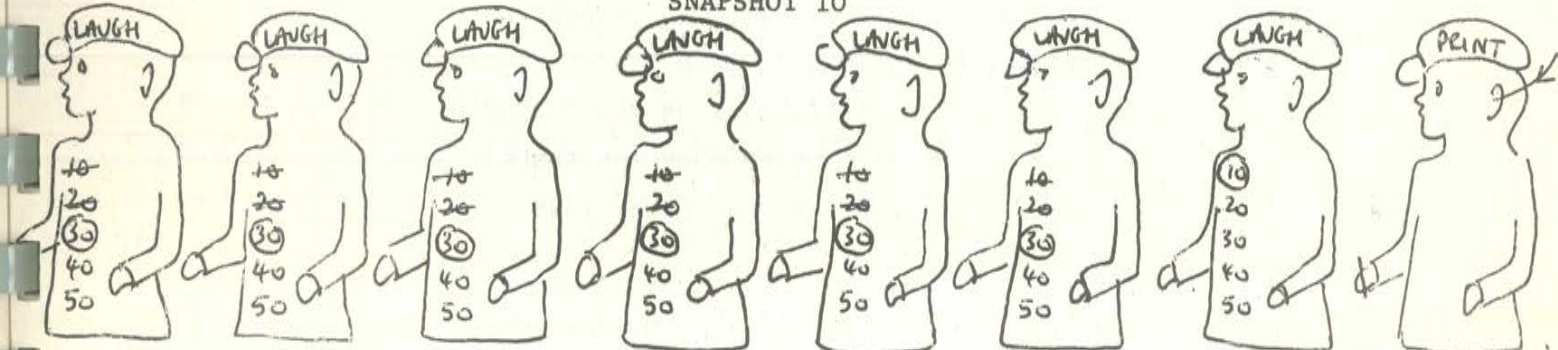
HAHA
HOHO
HAHA
HOHO
HAHA
HOHO

SNAPSHOT 9



HAHA
HOHO
HAHA
HOHO
HAHA
HOHO

SNAPSHOT 10



SNAPSHOT 21

Here is another recursive procedure

```
W: DEFINE "HEXAGON
  D:10 FORWARD 100
  D:20 LEFT 60
  D:30 HEXAGON
  D:END
```

EXERCISE 3: Try out procedure HEXAGON.

EXERCISE 4: Change the turn in line 20 to LEFT 177 and run the procedure again.

We can write recursive procedures with inputs:-

```
W: DEFINE "BORING "ADJECTIVE
  D:10 PRINT VALUE "ADJECTIVE-
  D:20 PRINT "WEATHER
  D:30 PRINT "TODAY
  D:40 BORING VALUE "ADJECTIVE
  D:END
```

EXERCISE 5: Try this procedure out

e.g.

W: BORING "LOVELY

W: BORING "COLD

The input for BORING is the word "LOVELY or the word "COLD. This is given as the value of the input to each succeeding BORING sub-procedure.

The sub-procedure in the recursion does not have to have the same value input as the super-procedure.

EXERCISE 6: Try defining the following procedure

```
W: DEFINE "INSOMNIA "YAWN
  D:10 REPEAT VALUE "YAWN PRINT "SHEEP
  D:20 PRINT [JUMPED OVER THE FENCE]
  D:30 INSOMNIA ADD 1 VALUE "YAWN
  D:END
```

Run this procedure with a number input

W: INSOMNIA 5

Each INSOMNIA procedure has an input named "YAWN. The first INSOMNIA has 5 as the value of "YAWN. The sub-procedure INSOMNIA it calls has 6 as the value of its "YAWN.

EXERCISE 7: Define the following procedure which takes two inputs

```
W: DEFINE "SWOP "A "B
    D:10 PRINT VALUE "A
    D:20 SWOP VALUE "B VALUE "A
    D:END
```

Run this procedure with any two inputs.

The following procedure, which needs two numbers as input, prints a whole series of numbers.

```
W: DEFINE "SPAGHETTI "NUMA "NUMB
    D:10 PRINT VALUE "NUMA
    D:20 SPAGHETTI (VALUE "NUMB)(ADD VALUE "NUMA VALUE "NUMB)
    D:END
```

EXERCISE 8: Define SPAGHETTI and try it out with various numbers. Try to guess what series of numbers will be printed.

```
W: SPAGHETTI 0 0
W: SPAGHETTI 0 1
W: SPAGHETTI 1 0
W: SPAGHETTI 1 1
W: SPAGHETTI 100 0
W: SPAGHETTI 0 100
```

SUMMARY

A procedure can have a copy of itself as a sub-procedure. This is called RECURSION. We shall use it again. There is a limit to the number of unfinished sub-procedures which LOGO can keep in its working memory.

21. SPIRALS

We drew polygons in note 18, by repeatedly going forward and turning. We can use recursion to draw shapes like SPIRALS which are like polygons except that they progressively change as they draw.

EXERCISE 1: Define the following procedure:

```
W: DEFINE "SPIRAL "ANGLE "SIDE "STEP
  D: 10 FORWARD VALUE "SIDE
  D: 20 RIGHT VALUE "ANGLE
  D: 30 SPIRAL (VALUE "ANGLE) (ADD VALUE "STEP VALUE "SIDE) VALUE "STEP
  D: END
```

Run the SPIRAL with different inputs e.g.

```
W: SPIRAL 90 0 10
W: SPIRAL 90 10 0
W: SPIRAL 60 50 10
W: SPIRAL 60 200 -10
```

EXERCISE 2: Define a version of SPIRAL in which the side stays the same but the angle changes each time a new worker SPIRAL is called.

SUMMARY

Recursion can be used to draw spirals, because each sub-procedure SPIRAL passes on a changed value of one of the inputs to the next SPIRAL sub-procedure.

22: TRUE OR FALSE

There are QUESTION PROCEDURES whose names end in Q. These procedures give either the word "TRUE or the word "FALSE as their result depending on the value of their input.

EXERCISE 1: Try typing

```
W: PRINT NUMBERQ 3
W: PRINT NUMBERQ "THREE
W: PRINT LISTQ [THE CAT SAT ON THE MAT]
W: PRINT WORDQ "THE
W: PRINT EQUALQ 48 48
W: PRINT EQUALQ "THE [THE]
W: PRINT EQUALQ 3 4
```

There is a question procedure named

NOT

This takes a word as its input. The word must be "TRUE or "FALSE

EXERCISE 2: Try typing

```
W: PRINT NOT "TRUE
W: PRINT NOT "FALSE
W: PRINT NOT EQUALQ 3 4
W: PRINT NOT WORDQ "THE
```

EXERCISE 3: Find out what the following QUESTION PROCEDURES do:

LESSQ GREATERQ ZEROQ

It is possible to have an empty list (a pallet without any boxes on it!).

EXERCISE 4: Try typing

```
W: PRINT [ ]
W: PRINT EMPTYQ [ ]
W: PRINT EMPTYQ [THE CAT SAT ON THE MAT]
```


It is also possible to have an empty word

EXERCISE 5: Try typing

W: PRINT "

W: PRINT EMPTYQ "

W: PRINT EMPTYQ "THE

SUMMARY

The new procedures are:-

<u>Name of procedure</u>	<u>input</u>	<u>result</u>
NUMBERQ	1 number word or list	"TRUE if input a number
LISTQ	1 " " " "	"TRUE if input a list
WORDQ	1 " " " "	"TRUE if input a word
EQUALQ	2 " " " "	"TRUE if inputs the same
LESSQ	2 numbers	"TRUE if first input less than second
GREATERQ	2 numbers	"TRUE if first input greater than second
EMPTYQ	1 number word or list	"TRUE if input empty
ZEROQ	1 " " " "	"TRUE if input \emptyset
NOT	1 word "TRUE or "FALSE	1 word "FALSE or "TRUE

23. CONTROL PROCEDURES

There are some special LOGO procedures which can control how a command is executed. REPEAT and RESULT are both CONTROL PROCEDURES. There is another CONTROL PROCEDURE named

IF

This procedure needs one input which must be either the word "TRUE or the word "FALSE. This input will usually be the result of running a QUESTION PROCEDURE. If the input is "TRUE then the command following a MARKER the English word THEN is executed.

EXERCISE 1: Try
W:IF NUMBERQ 9 THEN PRINT [ITS A NUMBER]

EXERCISE 2: Try
W:IF EQUALQ 3 4 THEN PRINT "SNAP

There may be another command we want to execute if the input to IF is the word "FALSE. The MARKER ELSE is used to mark the beginning of such a command.

When we use IF we may want to type a command that is too long to fit on one line. In cases like this we must tell LOGO that the command is not finished when we get to the end of the line. We do this by pressing the + button, before we press the green command button. When a command is being continued onto another line, LOGO types the prompt C: for continue.

EXERCISE 3: Try typing
W:IF NUMBERQ "CAT THEN PRINT [ITS A NUMBER] +
C:ELSE PRINT [ITS NOT A NUMBER]

EXERCISE 4: Try typing
W:DEFINE "LIAR "A "B
D:10 IF EQUALQ VALUE "A VALUE "B THEN +
C:PRINT [THEY ARE DIFFERENT] ELSE +
C:PRINT [THEY ARE THE SAME]
D:END

Try out LIAR with various inputs.

EXERCISE 5:

Try typing

```
W: DEFINE "DOUBLEQ "A "B
D: 10 IF EQUALQ VALUE "A (MULTIPLY VALUE "B 2) +
C: THEN RESULT "TRUE ELSE RESULT "FALSE
D: END
```

Try out this question procedure

e.g.

```
W: PRINT DOUBLEQ 10 5
W: PRINT DOUBLEQ 19 37
W: PRINT DOUBLEQ 5 10
```

EXERCISE 6:

Try typing

```
W: DEFINE "PONTOON "SCORE
D: 10 IF GREATERQ VALUE "SCORE 21 THEN PRINT "BUST
D: 20 IF EQUALQ VALUE "SCORE 21 THEN PRINT "PONTOON
D: 30 IF GREATERQ VALUE "SCORE 17 THEN PRINT "STICK
D: 40 PRINT "TWIST
D: END
```

Try this procedure out

e.g.

```
W: PONTOON 25
W: PONTOON 21
W: PONTOON 18
```

The difficulty with PONTOON is that it does not stop after printing out its first word. The control procedure RESULT is designed for exactly this sort of problem. As well as returning its input as the result of the procedure it is in, it also stops any further execution inside that procedure.

EXERCISE 7:

Using RESULT define a procedure NEWPONT which returns as its result one of "BUST, "PONTOON, "STICK or "TWIST.

Try it out

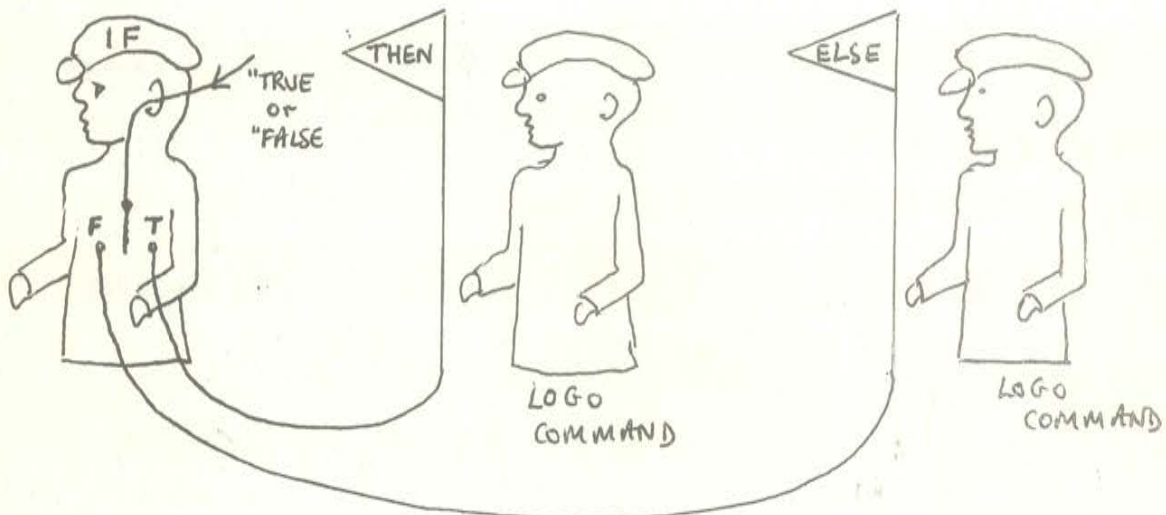
e.g.

```
W: PRINT NEWPONT 18
```


SUMMARY

We have introduced the new CONTROL PROCEDURE IF and its markers THEN and ELSE.

WE have introduced the + button for continuing long commands on to extra lines.



24. QUIZZES

You can use LOGO to write Quiz procedures. These can ask your friends questions and respond to their answers. There is a LOGO procedure named

REPLY

When this procedure is executed a prompt

REPLY:

is typed. LOGO then waits until the green command button is typed. Whatever was typed after the prompt and before the green command button is made into a list. This list is the result of executing the procedure REPLY

EXERCISE 1: Try typing

```
W: DEFINE "ECHO
D:10 PRINT [I WILL ECHO WHATEVER YOU TYPE]
D:20 PRINT REPLY
D:30 PRINT [DID YOU HEAR THE ECHO]
D:END
```

Run this procedure and type in something to be echoed when you get the prompt.

EXERCISE 2: Try typing

```
W: DEFINE "FOOTBALL
D:10 PRINT [WAT IS THE GREATEST +
C: FOOTBALL TEAM IN THE WORLD]
D:20 IF EQUALQ REPLY [PENICUIK WANDERERS] +
C: THEN PRINT [RIGHT ON] ELSE PRINT [WRONG AGAIN]
D:END
```

Try out this procedure.

EXERCISE 3: Type in a quiz procedure of your own.

EXERCISE 4: Define a superprocedure which has two quiz procedures as its subprocedures.

The following procedure TEACHER is very useful as a subprocedure in quiz procedures. It can be used for any question and answer.

```
W: DEFINE "TEACHER "QUESTION "ANSWER
D:10 PRINT VALUE "QUESTION
D:20 IF EQUALQ REPLY VALUE "ANSWER +
C: THEN PRINT "RIGHT ELSE PRINT "WRONG
D:END
```

EXERCISE 5: Define TEACHER and then use it in a superprocedure like the following

```
D: DEFINE "LESSON
W:10 TEACHER [WHAT IS 3 TIMES 43] [12]
W:20 TEACHER [WHAT IS THE CAPITAL OF SCOTLAND] [CARDIFF]
W:30 TEACHER [WHY DID THE CHICKEN CROSS THE ROAD] [TO +
C:GET TO THE OTHER SIDE]
W:END
```

SUMMARY

The new procedure is

NAME OF PROCEDURE	INPUT	EFFECT	RESULT
REPLY	None	Types a prompt	A list of what was typed in.

25. STOPPING PROCEDURES

In note 20 we defined a recursive procedure INSOMNIA which looked like this

```
W: DEFINE "INSOMNIA "YAWN
  D:10 REPEAT VALUE "YAWN PRINT "SHEEP
  D:20 PRINT [JUMPED OVER THE FENCE]
  D:30 INSOMNIA ADD 1 VALUE "YAWN
  D:END
```

EXERCISE 1: RECALL or DEFINE procedure INSOMNIA.

The only way you can stop INSOMNIA is by interrupting it by pressing the red emergency stop button. There is a way of defining INSOMNIA such that it stops when you want it to. There is a CONTROL PROCEDURE called

STOP

which needs no input and produces no result. Its effect is to stop the execution of the procedure it is in.

EXERCISE 2: Type in the following change to INSOMNIA.

```
W: CHANGE "INSOMNIA
  D:5 IF GREATERQ VALUE "YAWN 5 THEN STOP
  D:END
```

EXERCISE 3: Run INSOMNIA with different inputs

e.g.

W: INSOMNIA 4

W: INSOMNIA 16

W: INSOMNIA 5

EXERCISE 4: Change INSOMNIA by adding the following lines:

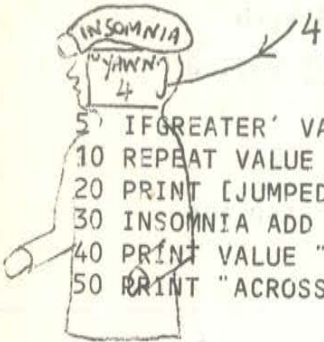
```
W: CHANGE "INSOMNIA
    D:40 PRINT VALUE "YAWN
    D:50 PRINT "ACROSS
    D:END
```

Run INSOMNIA with various inputs as before.

A procedure stopped by STOP produces no special result. Here are some snapshots of what happens when you type

W: INSOMNIA 4

We have not drawn in all the workers who take part in this process.

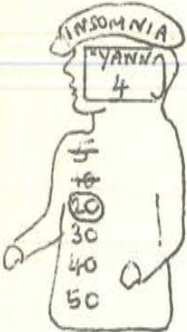


```
5 IF GREATER VALUE "YAWN 5 THEN STOP
10 REPEAT VALUE "YAWN PRINT "SHEEP
20 PRINT [JUMPED OVER THE FENCE]
30 INSOMNIA ADD 1 VALUE "YAWN
40 PRINT VALUE "YAWN
50 PRINT "ACROSS
```

INSOMNIA is called with 4 as the value of its input

Line 5 checks whether the value of "YAWN is greater than 5.

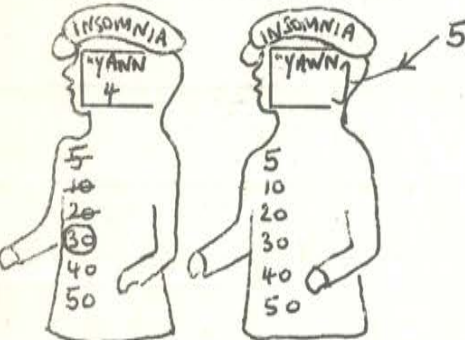
SNAPSHOT 1



Lines 10 and 20 have the effect of printing

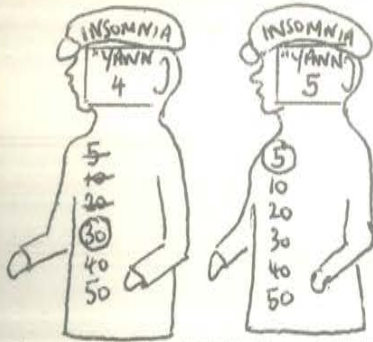
```
SHEEP
SHEEP
SHEEP
SHEEP
[JUMPED OVER THE FENCE]
```

SNAPSHOT 2



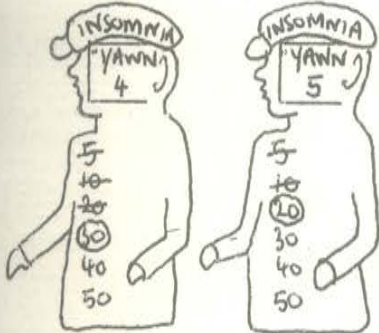
INSOMNIA calls a new INSOMNIA with 5 as the value of its input.

SNAPSHOT 3



SNAPSHOT 4

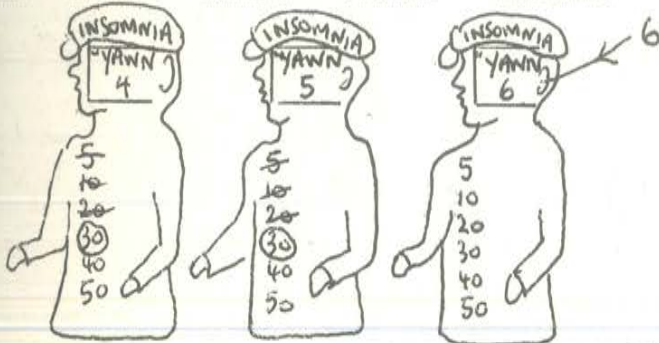
Line 5 checks whether the value of "YAWN" is greater than 5. It is not so



SNAPSHOT 5

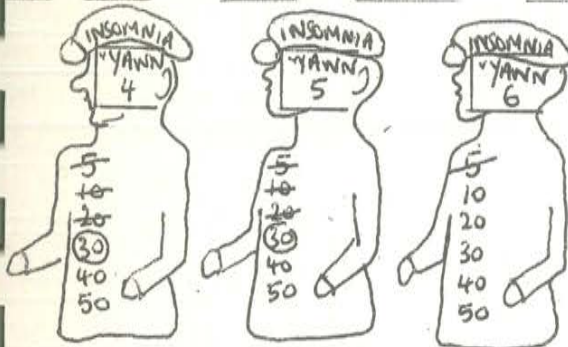
Lines 10 and 20 print

SHEEP
SHEEP
SHEEP
SHEEP
[JUMPED OVER THE FENCE]



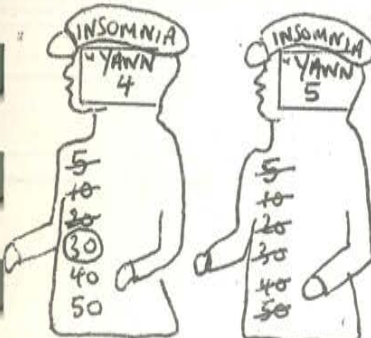
SNAPSHOT 6

INSOMNIA calls yet another INSOMNIA with 6 as the value of its input.



SNAPSHOT 7

Line 5 finds that the value of "YAWN" is bigger than 5; so this INSOMNIA stops, and tells the procedure which called it that it has finished.



SNAPSHOT 8

Now the second INSOMNIA can continue on its line 40 printing

5
ACROSS

Now it has finished



So the first INSOMNIA can
continue on its line 40 to
print

4
ACROSS

SNAPSHOT 9

Now it has finished as well.

SUMMARY

The new control procedure is

NAME OF PROCEDURE	INPUT	EFFECT
STOP	none	stops procedure it is in.

26. TRACING PROCEDURES

There is a procedure named

TRACE

which can be used to help you debug and understand your procedures. It needs one input which should be the quoted name of a procedure to be traced, or a list of names of procedures to be traced.

EXERCISE 1: Recall and show the INSOMNIA procedure from note 25.
Run TRACE as follows
W: TRACE "INSOMNIA

The message means that INSOMNIA has been MARKED so that, in the future, every time it is executed a message will be typed by LOGO. This message will tell you that

- (a) the procedure has been called
- (b) what the values of its inputs are

When a MARKED procedure finishes another message is typed. This tells you

- (a) what the result of the procedure is, if any
- (b) that the procedure has finished.

EXERCISE 2: Run INSOMNIA with various inputs.

You can TRACE LOGO's procedures as well as your own.

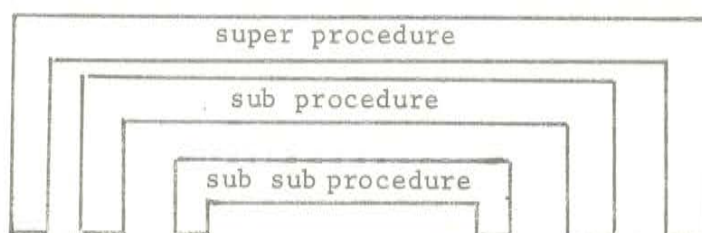
EXERCISE 3: MARK LOGO's procedure ADD for tracing
W: TRACE "ADD
Now run INSOMNIA again.
Also try
W: PRINT ADD ADD ADD 1 2 4 8

In chapter 25 a series of snapshots of INSOMNIA 4 being executed was drawn. Here we show the messages produced by TRACING INSOMNIA 4.

Comments		LOGO Messages
		W: INSOMNIA 4
INSOMNIA called		-->INSOMNIA
name and value of input		YAWN = 4,
10 effects of PRINT		SHEEP SHEEP SHEEP SHEEP [JUMPED OVER THE FENCE]
20		
30		
INSOMNIA called		-->INSOMNIA
name and value of input		YAWN = 5,
10 effects of PRINT		SHEEP SHEEP SHEEP SHEEP SHEEP [JUMPED OVER THE FENCE]
20		
30		
INSOMNIA called		-->INSOMNIA
name and value of input		YAWN = 6,
5 no result		NO RESULT
INSOMNIA ended		<INSOMNIA
40 effects of PRINT		5 ACROSS
50		
no result produced		NO RESULT
INSOMNIA ended		<INSOMNIA
40 effects of PRINT		4 ACROSS
50		
no result produced		NO RESULT
INSOMNIA ended		<INSOMNIA
		W:

Each call to a sub-procedure is shown by the > moving two spaces to the right.

EXERCISE 4: Turn the diagram on the previous page on its side. The sub-procedures look like NESTED tables:



LOGO can also be commanded to remove the MARK from a procedure by running the procedure

UNTRACE

which needs a single input, either a quoted name or a list of names.

EXERCISE 5: Try
W: UNTRACE [ADD INSOMNIA]
and run INSOMNIA

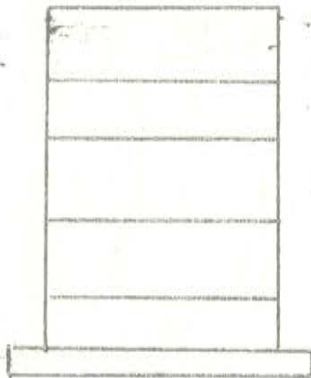
SUMMARY

The new procedures are:

NAME OF PROCEDURE	INPUT	EFFECT
TRACE	list of procedure names or single quoted procedure name	marks named procedures so that message given when they are executed
UNTRACE	list of procedure names or single quoted procedure name	removes mark from named procedures.

27. HOW LISTS WORK

A list is like a stack of boxes on a pallet



There is a LOGO procedure named

FIRST

which gives as its RESULT the top box, or first element of the list which is its input.

EXERCISE 1: Try typing

W: PRINT FIRST [THE CAT SAT ON THE MAT]

W: PRINT FIRST [SAT CAT ON THE MAT]

W: PRINT FIRST [CAT]

You can PUT a box onto the top of the stack by running the LOGO procedure

PUT

which gives the whole of the new stack or list as its result. PUT needs two inputs, the first input is the new box, the second input is the stack it is to be put on.

EXERCISE 2:

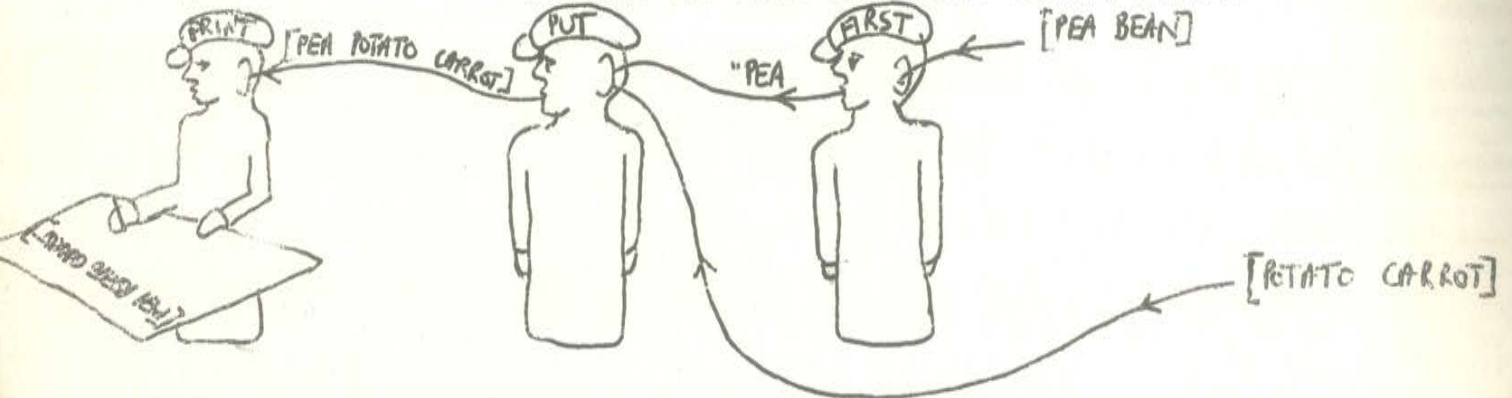
Try typing

W: PRINT PUT "CAT [DOG RABBIT]

W: PRINT PUT 10 [9 8 7 6 5 4 3 2 1]

W: PRINT PUT [9 8 7 6 5 4 3 2 1] 10

W: PRINT PUT FIRST [PEA BEAN] [POTATO CARROT]



You can also knock off and throw away the top box by running the procedure

REST

which gives as its RESULT the rest of the stack.

EXERCISE 3:

Try typing

W: PRINT REST [THE CAT SAT ON THE MAT]

W: PRINT REST [10 9 8 7 6 5 4 3 2 1]

W: PRINT FIRST REST [THE CAT SAT ON THE MAT]

EXERCISE 4:

Make a procedure named SECOND which takes as its input a list. The result of SECOND should be the second element of the list.

EXERCISE 5:

Make a procedure named THIRD whose result is the third element of a list.

EXERCISE 6:

Try typing

W: PRINT THIRD [CAT DOG]

FIRST, REST and PUT all work with numbers and words too

EXERCISE 7: Try typing
 W: PRINT FIRST 783
 W: PRINT REST "ELEPHANT
 W: PRINT PUT 5 94321

SUMMARY

The new procedures are:-

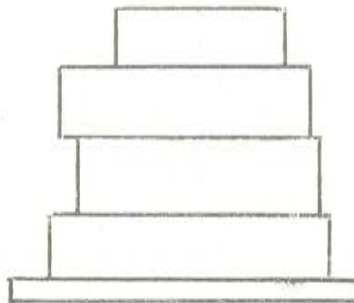
NAME OF PROCEDURE	INPUT	EFFECT
FIRST	List	The first element of the list
	Word	The first letter of the word
	Number	The first digit in the number
REST	List, word or number	The rest of the list word or number
PUT	Two inputs numbers, words or lists	Puts first input in front of second. Except lists cannot be put in front of words or numbers. (Can't put pallets on boxes.)

28. FINDING THINGS IN LISTS

We use lists for storing several names or numbers. For example this is a list of names

[DONALD MARY HAMISH FERGUS FIONA ANDREW]

People can easily tell that the name FERGUS is somewhere in this list by looking at it. But in LOGO a list is like a stack of boxes on a pallet.



Imagine there was a name on a piece of paper in each box and we wanted to find out whether one of the names was FERGUS.

There would be three jobs to do:-

- Job 1. : If there are no boxes left on the pallet then say FERGUS cannot be found and stop searching.
- Job 2. : If the name FERGUS is in the top box then say FERGUS has been found and stop searching.
- Job 3. : Throw away the top box and search the rest of the boxes on the pallet.

This is another example of RECURSION.

EXERCISE 1: Define this procedure which carries out the three jobs

```

W: DEFINE "SEARCH BOXES
Job 1      D: 10 IF EMPTYQ VALUE "BOXES THEN PRINT [NOT FOUND]
           D: 20 IF EMPTYQ VALUE "BOXES THEN STOP
Job 2      D: 30 IF EQUALQ "FERGUS (FIRST VALUE "BOXES) THEN PRINT [FOUND]
           D: 40 IF EQUALQ "FERGUS (FIRST VALUE "BOXES) THEN STOP
Job 3      D: 50 SEARCH REST VALUE "BOXES
           D: END

```

EXERCISE 2: Try SEARCH with different inputs

e.g.

```

W: SEARCH [DONALD MARY HAMISH FERGUS FIONA ANDREW]
W: SEARCH [DONALD MARY HAMISH FIONA ANDREW]
W: SEARCH [1 2 3 FERGUS 4 5]
W: SEARCH [1 2 FERGUS 3 4 5]

```

EXERCISE 3: Mark SEARCH for tracing and run it again

SEARCH is a recursive procedure. Each SEARCH worker looks in the top box of the stack he is given and then calls on another SEARCH worker to look in the rest of the boxes

EXERCISE 4: Define a procedure like SEARCH which has an extra input. The value of this input will be the element to be looked for in the list. This procedure could look for anything, not just "FERGUS.

If this procedure were called FIND it would be run like this:-

```

W: FIND "HAMISH [DONALD MARY HAMISH FERGUS]
[FOUND]
W:

```

FIRST and REST also work with words and numbers as well as lists. So we could find a digit in a number or a letter in a word.

EXERCISE 5: Try
 W: FIND "B "ABCDEFGH
 W: FIND 6 12345

EXERCISE 6: Mark FIND for tracing and run it again.

FIND and SEARCH give no result. They only have the effect of printing
[FOUND] or [NOT FOUND].

SUMMARY

Recursive procedures can be used to find out what elements there are in
a list.

29. COMING BACK OUT OF RECURSION

In note 28 procedures FIND and SEARCH were defined which gave no result. They only had the effect of printing [FOUND] or [NOT FOUND]. A more useful version of FIND would give the result "TRUE or "FALSE. It would then be a question procedure and could be used with the control procedure IF.

EXERCISE 1: Define this new version of FIND named MEMBERQ which does the same three jobs as FIND.

```
W: DEFINE "MEMBERQ "THING "BOXES
    D: 10 IF EMPTYQ VALUE "BOXES THEN RESULT "FALSE
    D: 20 IF EQUALQ (VALUE "THING) (FIRST VALUE "BOXES) THEN RESULT "TRUE
    D: 30 RESULT MEMBERQ VALUE "THING REST VALUE "BOXES
    D: END
```

Try out MEMBERQ e.g.

```
W: PRINT MEMBERQ "CAT [DOG CAT RABBIT]
```

```
W: PRINT MEMBERQ "CAT [DOG RABBIT ZEBRA LION]
```

MEMBERQ can combine lines 10 and 20 of FIND because RESULT gives a result and stops the procedure. Lines 30 and 40 of FIND can be combined for the same reason.

In line 30, MEMBERQ is called as a sub-procedure. The RESULT on this line ensures that the result of the MEMBERQ sub-procedure becomes the result of the MEMBERQ super-procedure.

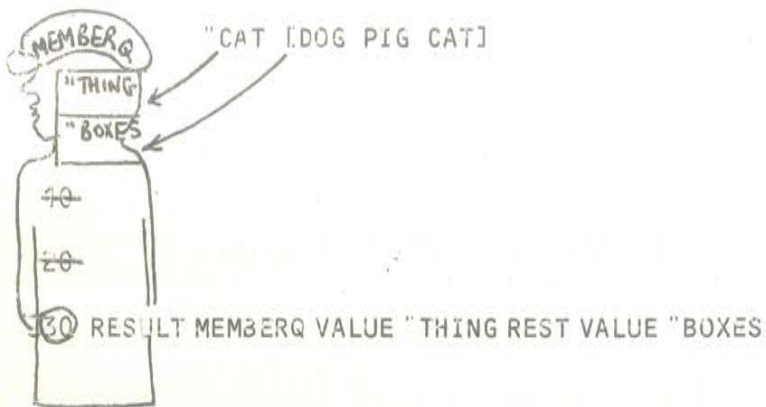
EXERCISE 2: Trace MEMBERQ and run it again
Try MEMBERQ on a word or number e.g.

```
W: PRINT MEMBERQ "T "ROOM
```

Compare the trace of

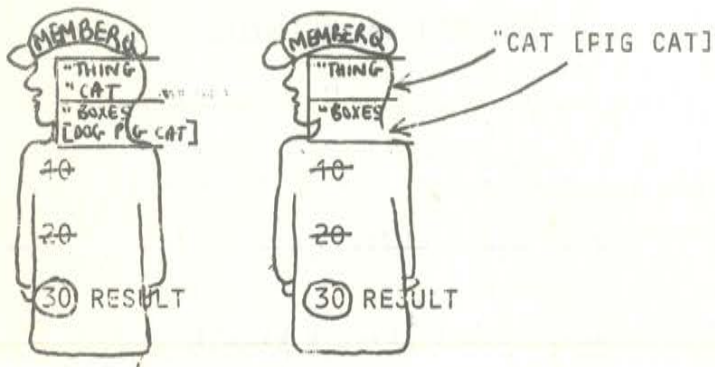
```
W: PRINT MEMBERQ "CAT [DOG PIG CAT]
```

with the next diagram.



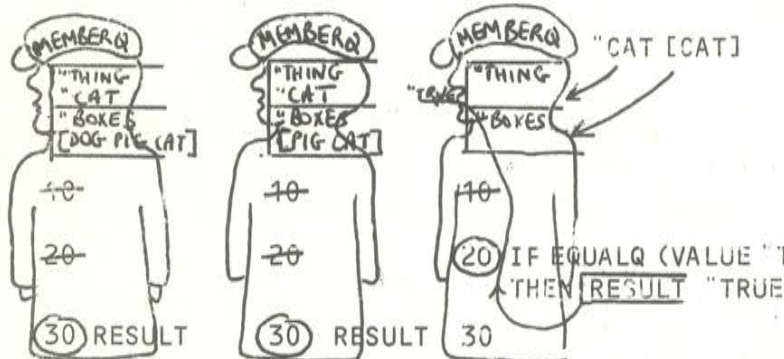
line 10: [DOG PIG CAT] is not empty
 line 20: "CAT is not equal to "DOG
 line 30: the input to RESULT is
 needed so call a
 MEMBERQ sub-procedure
 and give it "CAT and
 [PIG CAT] as inputs.

SNAPSHOT 1



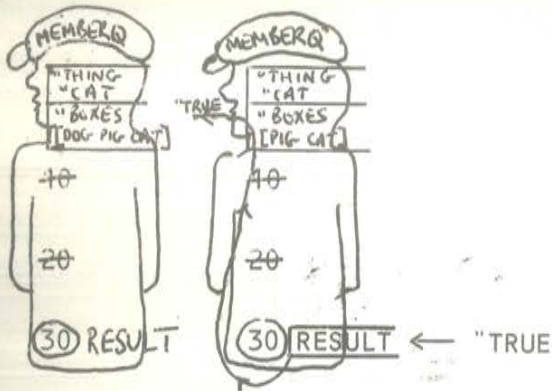
line 10: [PIG CAT] is not empty
 line 20: "CAT is not equal to "PIG
 line 30: the input to RESULT is
 needed so call a
 MEMBERQ sub-procedure
 and give it "CAT and
 [CAT] as inputs

SNAPSHOT 2



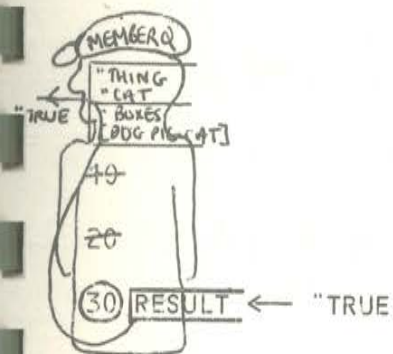
line 10: [CAT] is not empty
 line 20: "CAT is equal to "CAT
 so give result "TRUE
 which stops the procedure.

SNAPSHOT 3



line 30: RESULT now has its input and can stop the procedure with the result "TRUE.

SNAPSHOT 4



line 30: RESULT now has its input and can stop the procedures with the result "TRUE.

SNAPSHOT 5

EXERCISE 3: Using MEMBERQ as a sub-procedure define a procedure which expects a single letter as the value of its input. The procedure should print either "VOWEL or "CONSONANT depending on the input letter value.

You can compare the way MEMBERQ comes out of recursion with the following procedure named CHOP.

```
W: DEFINE "CHOP "BOXES
  D: 10 PRINT VALUE "BOXES
  D: 20 IF EMPTYQ VALUE "BOXES THEN STOP
  D: 30 CHOP REST VALUE "BOXES
  D: END
```

EXERCISE 4: Define CHOP and run it with different inputs e.g.

```
W: CHOP [COMING OUT OF RECURSION]
W: CHOP "ABCDEFG
```

EXERCISE 5: Change CHOP by inserting a line 40
D: 40 PRINT VALUE "BOXES

Try to predict how this version of CHOP
will work.

Try out CHOP.

CHOP works in much the same way as INSOMNIA which is described in note 26
and note 25.

SUMMARY

MEMBERQ is an example of a procedure which is recursive and gives a
result. MEMBERQ can search down a list and send a result back depending
on what it finds.

30. WORKING ON LISTS

You can write a procedure which counts how many elements there are in a list. This procedure is similar to MEMBERQ but has an overall result which is the number of elements in the list.

The stages in counting elements in a list, or boxes on a pallet, are as follows:-

- Job 1: If the pallet is empty the total is \emptyset .
 Job 2: Throw away the top box and add 1 to the total from counting the rest of the boxes.

EXERCISE 1: Define the following counting procedure

W: DEFINE "TALLY "BOXES

Job 1: D: 10 IF EMPTYQ VALUE "BOXES THEN RESULT \emptyset

Job 2: D: 20 RESULT ADD 1 (TALLY REST VALUE "BOXES)

D: END

Try out TALLY with some lists. Also try it with a word and a number.

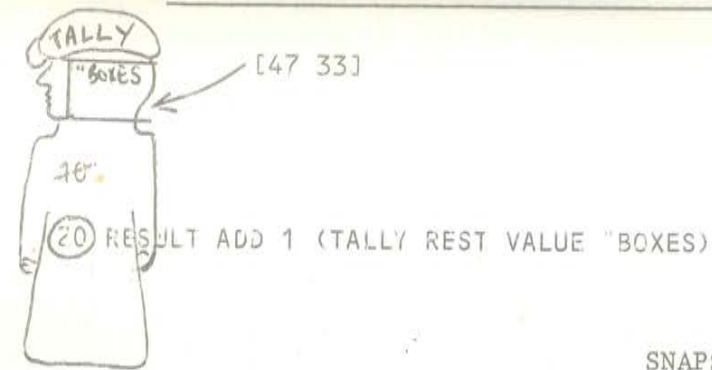
EXERCISE 2: Trace TALLY and see how it runs.

The snapshots on the next page show the command

W: PRINT TALLY [47 33]

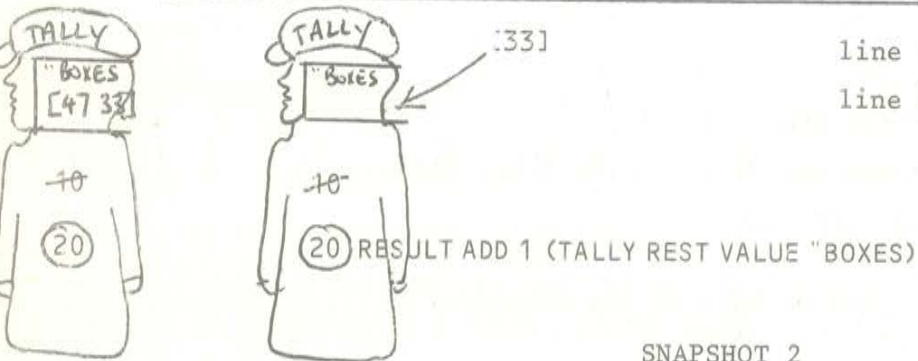
2

being executed. Many of the procedures are not shown in the snapshots which concentrate on TALLY.



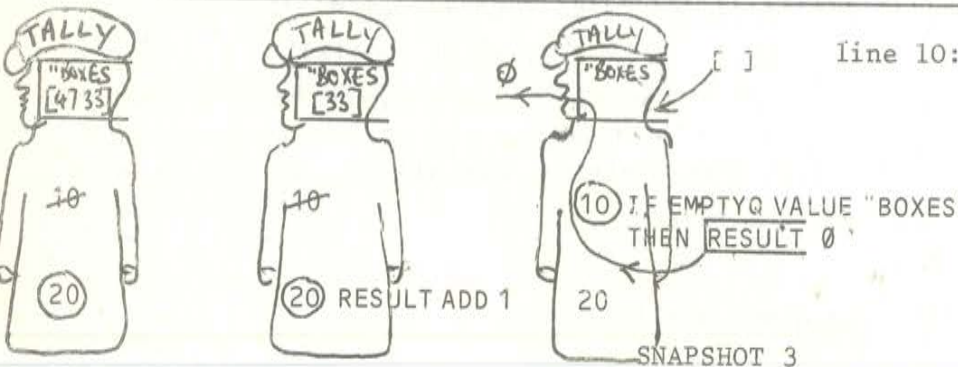
SNAPSHOT 1

line 10: [47 33] is not empty
 line 20: In order to give the result of adding 1 to the total the rest of the boxes must be tallied. Call a TALLY sub-procedure.



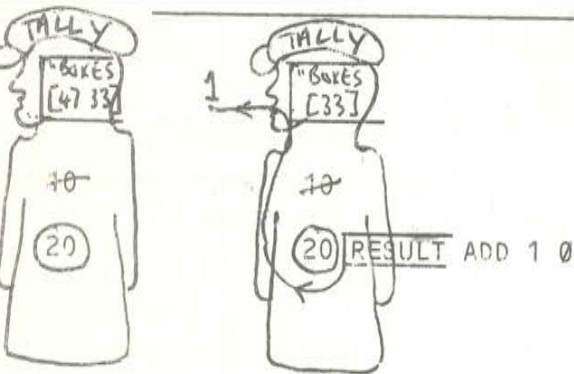
SNAPSHOT 2

line 10: [33] is not empty
 line 20: In order to give the result of adding 1 to the total the rest of the boxes must be tallied. Call a TALLY sub-sub-procedure.



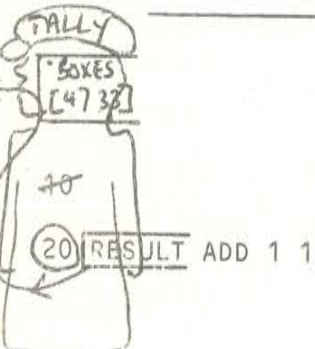
SNAPSHOT 3

line 10: [] is empty so give the result 0



SNAPSHOT 4

Now line 20 can be completed
 ADD 1 0 comes to 1,
 so give the result 1.



SNAPSHOT 5

Now line 20 can be completed
 ADD 1 1 comes to 2,
 so give the result 2.

You can write a very similar procedure to TALLY which adds up a list of numbers (like on a shopping bill). This procedure could be named TOTAL. Instead of adding 1 each time as TALLY does, it would add the first number in the list to the total.

EXERCISE 3: Define a procedure TOTAL which adds up a list of numbers. Try it out.

EXERCISE 4: Define a super-procedure AVERAGE which uses TALLY and TOTAL as sub-procedures to calculate the average of a list of numbers.

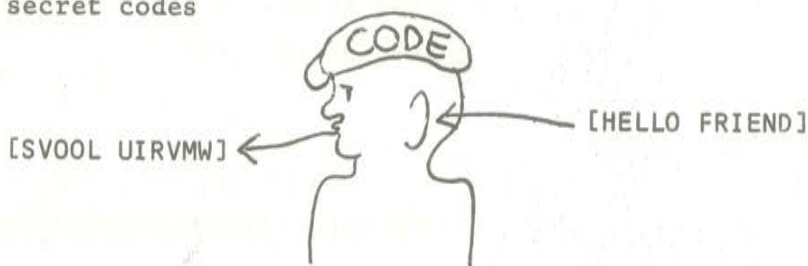
EXERCISE 5: Define a procedure named MANY which works like TOTAL but which multiplies together all the numbers in a list.
For example: How many seconds in a day?
W: PRINT MANY [60 60 24]
The procedure would work out $60 \times 60 \times 24$.

SUMMARY

As well as finding out what elements are in a list, you can also work with the elements of a list using recursion.

31. CONSTRUCTING LISTS

You can define a procedure which constructs a list and gives this list as its result. This note uses this kind of procedure to show you how to make secret codes



The input to `CODE` is a list of words. `CODE` will construct a new list of coded words using procedure `PUT` to make the list.

One of the sub-jobs is to code each separate letter. Here is an example of a procedure for coding letters.

W: DEFINE "FLIP "LETTER

D: 10 IF EQUALQ VALUE "LETTER "A THEN RESULT "Z

D: 20 IF EQUALQ VALUE "LETTER "B THEN RESULT "Y

D: 30 IF EQUALQ VALUE "LETTER "C THEN RESULT "X

:

D: 260 IF EQUALQ VALUE "LETTER "Z THEN RESULT "A

D: END

(We have left out some lines)

EXERCISE 1: Define this procedure or your own procedure which encodes individual letters.

Try this procedure out

e.g.

W: PRINT FLIP "J

W: PRINT FLIP 7

Complete words can be coded by procedure SCRAMBLE. It will use FLIP as a sub-procedure. There are two jobs to do.

Job 1: If there are no more letters in the word to be coded then the result is the empty word

Job 2: Otherwise PUT the flipped first letter of the word at the beginning of the scrambled rest of the word.

```
W: DEFINE "SCRAMBLE "WORD
  D: 10 IF EMPTYQ VALUE "WORD THEN RESULT "
  D: 20 RESULT PUT (FLIP FIRST VALUE "WORD) (SCRAMBLE REST VALUE "WORD)
  D: END
```

EXERCISE 2: Define SCRAMBLE and run it with various words
e.g.

```
W: PRINT SCRAMBLE "LOGO
```

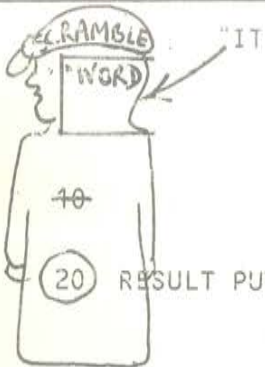
The following snapshots show some of the procedures called in executing

```
W: PRINT SCRAMBLE "IT
```

EXERCISE 3: Type

```
W: TRACE [SCRAMBLE FLIP PUT]
```

Compare the trace with the diagrams.

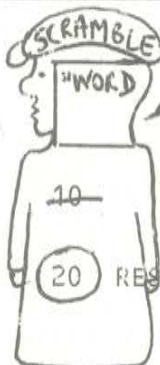
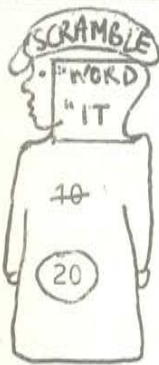


RESULT PUT "R (SCRAMBLE "T)

SNAPSHOT 1

line 10: "IT is not empty

line 20: My result will be a new word made with PUT. Call FLIP to change "I to "R. Call SCRAMBLE as a sub-procedure to deal with the rest of the word

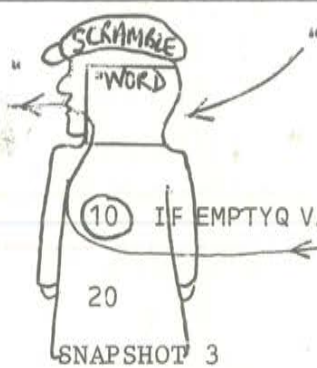
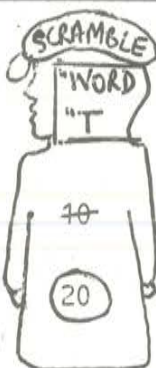
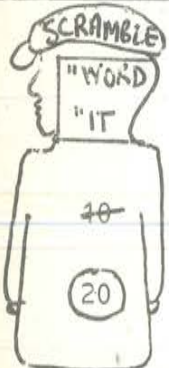


RESULT PUT "G (SCRAMBLE")

SNAPSHOT 2

line 10: "T is not empty

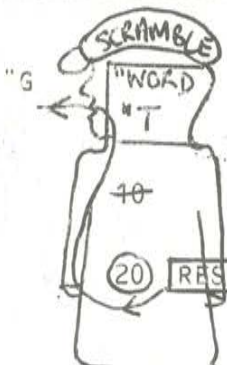
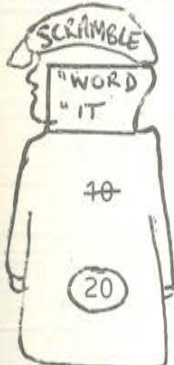
line 20: My result will be a new word made with PUT. Call FLIP to change "T to "G. Call SCRAMBLE as a sub-sub-procedure to deal with the rest of the word.



IF EMPTYQ VALUE "WORD THEN RESULT"

SNAPSHOT 3

line 10: " is empty so give the result "



RESULT PUT "G "

SNAPSHOT 4

Now PUT can make a new word out of "G and ". This word is "G. I can now give "G as my result.



RESULT PUT "R "G

SNAPSHOT 5

Now PUT can make a new word out of "R and "G. This word is "RG. I can now give "RG as my result.

'The Story of the three SCRAMBLES'

In the example, three procedures named SCRAMBLE were called. Let us call them super-SCRAMBLE, sub-SCRAMBLE and sub-sub-SCRAMBLE



I can deal with my top box but I have no where to put it until my sub-SCRAMBLE finishes.



I can deal with my top box but I have no where to put in until my sub-sub-SCRAMBLE finishes.



I can deal with my stack because it is empty. Here's an empty stack for you sub-SCRAMBLE.



Thank you sub-sub-SCRAMBLE. I can now put my new box on that empty stack. Here's a new stack for you super-SCRAMBLE.



Thank you sub-SCRAMBLE. I can now put my new box on that stack. Here's a new stack for you PRINT.

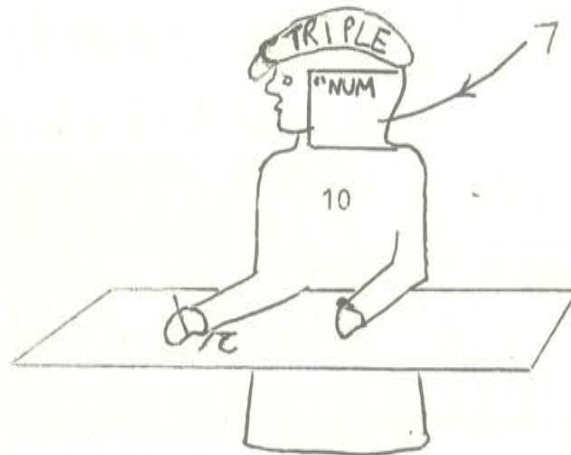
EXERCISE 3: Define a procedure CODE which uses SCRAMBLE as a sub-procedure. CODE should encode a whole list of words
e.g.
W: PRINT CODE [THE CAT SAT ON THE MAT]

This procedure will do two jobs very much like CODE.

- Job 1: If there are no more words in the list then the result is the empty list.
- Job 2: Otherwise PUT the scrambled first word of the list at the beginning of the CODED rest of the list.

SUMMARY

A procedure can construct a list using PUT and recursion. Each CODE called can only put its new list together when it has the new element and the new list to put it in.

32. VARIABLES

EXERCISE 1: Define the following procedure
 W: DEFINE "TRIPLE "NUM
 D: 10 PRINT MULTIPLY VALUE "NUM 3
 D: END

Run the procedure

W: TRIPLE 7

EXERCISE 2: Try typing the following
 W: PRINT VALUE "NUM

When you ran the procedure TRIPLE it had an input called "NUM which had the value 7. This value is PRIVATE to that worker TRIPLE. The procedure VALUE can only get the 7 out of "NUM when VALUE is called inside the worker TRIPLE.

EXERCISE 3: Define the following procedure

W: DEFINE "QUAD "NUM

D: 10 PRINT MULTIPLY VALUE "DIGIT 4

D: END

Run the procedure

e.g.

W: QUAD 12

QUAD did not work because when LOGO tried to execute VALUE "DIGIT no input of that name could be found.

An input is a type of VARIABLE. A VARIABLE has a name and a value.

A variable is like a box with a name on it. Inside the box can be a word, number or list.

There is a way in LOGO of making PUBLIC boxes or variables that any worker can look into.

There is a LOGO procedure named

MAKE

which creates LOGO boxes. It needs two inputs. The first input is a LOGO word to be the name of the new box. The second input is to be the value or contents of the new box

EXERCISE 4: Create some new boxes

W: MAKE "NUMBER 10101

W: MAKE "SEASONS [SUMMER WINTER SPRING AUTUMN]

W: MAKE "JAM "STRAWBERRY

W: MAKE "DIGIT 11

"NUMBER
10101

"SEASONS
[SUMMER WINTER SPRING AUTUMN]

"JAM
"STRAWBERRY

"DIGIT
11

We use the procedure VALUE to find out what is in a box.

EXERCISE 5: Look inside your new boxes

e.g.

W: PRINT VALUE "NUMBER

EXERCISE 6: Run your procedure QUAD again

e.g.

W: QUAD 10000

When inside the procedure QUAD, LOGO tried to execute VALUE "DIGIT, it first looked for an input (a PRIVATE box) named "DIGIT. When it could not find one it then looked for a PUBLIC variable named "DIGIT. It found the one you made in Exercise 4.

As well as making new boxes, the procedure MAKE can also be used for changing the contents of a box which already exists.

EXERCISE 7: Try the following changes to the value of "DIGIT.

W: MAKE "DIGIT 2

W: MAKE "DIGIT ADD 5 6

W: PRINT VALUE "DIGIT

SUMMARY

There are two sorts of variables PRIVATE (inputs) and PUBLIC ones. PRIVATE variables only have values when the worker they belong to is being executed.

PUBLIC variables are stored in the working memory separately from procedures. Once a PUBLIC variable has been made the box stays in working memory until you type GOODBYE.

The new LOGO procedure is

NAME OF PROCEDURE	INPUT	EFFECT
MAKE	A word and one of a word a number a list	Creates a box in working memory whose name is the value of the first input and whose content is the value of the second input.

33. USING PUBLIC BOXES

Here is a procedure for a guessing game you can try on your friends.

EXERCISE 1: Define this procedure

```
W: DEFINE "GAME
  D: 10 PRINT [TRY TO GUESS THE NUMBER]
  D: 20 IF EQUALQ (FIRST REPLY) (VALUE "SECRET) THEN +
C: PRINT [WELL DONE] ELSE GAME
D: END
```

EXERCISE 2: Put a value in "SECRET and then ask a friend to run GAME.

You chose the number to go in "SECRET. But LOGO can choose numbers for itself. There is a procedure named

RANDOM

which needs one number as input. RANDOM gives a result which is a randomly chosen number between the input and zero.

EXERCISE 3: Try out RANDOM a few times
e.g.

```
W: PRINT RANDOM 10
W: PRINT RANDOM 10
W: PRINT RANDOM 10
W: PRINT RANDOM 237
```

Now you can play GAME yourself by letting LOGO choose the number to go in "SECRET.

EXERCISE 4: Put a random number in "SECRET

W: MAKE "SECRET RANDOM 10

Now run GAME and find out yourself what number LOGO chose.

You can move the turtle in a random way as well.

EXERCISE 5: Try defining and running this procedure

W: DEFINE "DRUNK

D: 10 FORWARD RANDOM 50

D: 20 LEFT RANDOM 360

D: 30 DRUNK

D: END

Here is a guessing game which gives hints as it runs.

W: DEFINE "GUESSER

D: 10 PRINT [CAN YOU GUESS THE NUMBER]

D: 20 MAKE "ANSWER FIRST REPLY

D: 30 IF GREATERQ VALUE "ANSWER VALUE "SECRET +

C: THEN PRINT [TOO BIG]

D: 40 IF LESSQ VALUE "ANSWER VALUE "SECRET +

C: THEN PRINT [TOO SMALL]

D: 50 IF EQUALQ VALUE "ANSWER VALUE "SECRET

C: THEN PRINT [WELL DONE] ELSE GUESSER

D: END

EXERCISE 6: Define and run a procedure like GUESSER.

Don't forget to put a value in the public box "SECRET which each GUESSER will need to look in.

In line 20 we stored the guess in a public box named "ANSWER. This was because we needed to use this guess in both lines 30 and 40. If we had used REPLY twice, LOGO would have waited for two different guesses, one on line 30 and one on line 40.

SUMMARY

MAKE can be used for storing values which can only be worked out once, but which we need to be used several times, or looked at by several procedures.

The new procedure is:-

NAME OF PROCEDURE	INPUT	RESULT
RANDOM	number	a randomly chosen number between 0 and the input.

APPENDIX A. PRINTING

The procedure PRINT types its input at the teletype and then moves the teletype carriage ready for a new line.

If you want to type several things on the same line you can run a procedure named

TYPESET

This procedure needs one input which can be a number, word, or list. This input is used to build the line to be typed.

For example if you type

W: TYPESET [THIS IS PAGE]

W: TYPESET 118

W: TYPESET [OF THE PRIMER]

Thus a line consisting of [THIS IS PAGE] 118 [OF THE PRIMER] will be built. This line will not be typed at the teletype until you build a carriage movement into the line.

There is a new procedure named

CARRIAGE

which needs no input. Once this procedure is run any line which has been built is typed at the teletype, and the carriage moved to the next line.

There is also a procedure named

SPACE

which will build a single space character into a line. It needs no input. There is a procedure named

TAB

which will build a tab of six spaces into a line. It needs no input.

There is a procedure named

SAY

which works in a way very similar to PRINT. It needs one input which can be a number, word or list. The input is typed at the teletype and

the carriage moved. However if the input is a list the list brackets [] are not typed

E.g.

W: SAY [GOODBYE FRED]

GOODBYE FRED

W:

There is also a new procedure named

TYPE

which works in just the same way as TYPESET except that a space is also inserted before the thing to be typed

E.g.

W: TYPE "FRED

W: TYPE "SMITH

W: CARRIAGE

will have the effect of typing

FRED SMITH

SUMMARY

The new procedures are

NAME OF PROCEDURE	INPUT	EFFECTS
TYPESET	number, word or list	builds input into line
CARRIAGE	no input	causes line to be typed
SPACE	no input	builds space into line
TAB	no input	builds six spaces into line
SAY	number, word or list	as for PRINT but lists typed without brackets
TYPE	number, word or list	as for TYPESET but a space also typed in front of input.

APPENDIX B. MORE ABOUT DRAWING CIRCLES

The procedure ARC draws part of a circle curving to the left. There is a procedure which draws circles curving to the right. It is named

ARCRIGHT

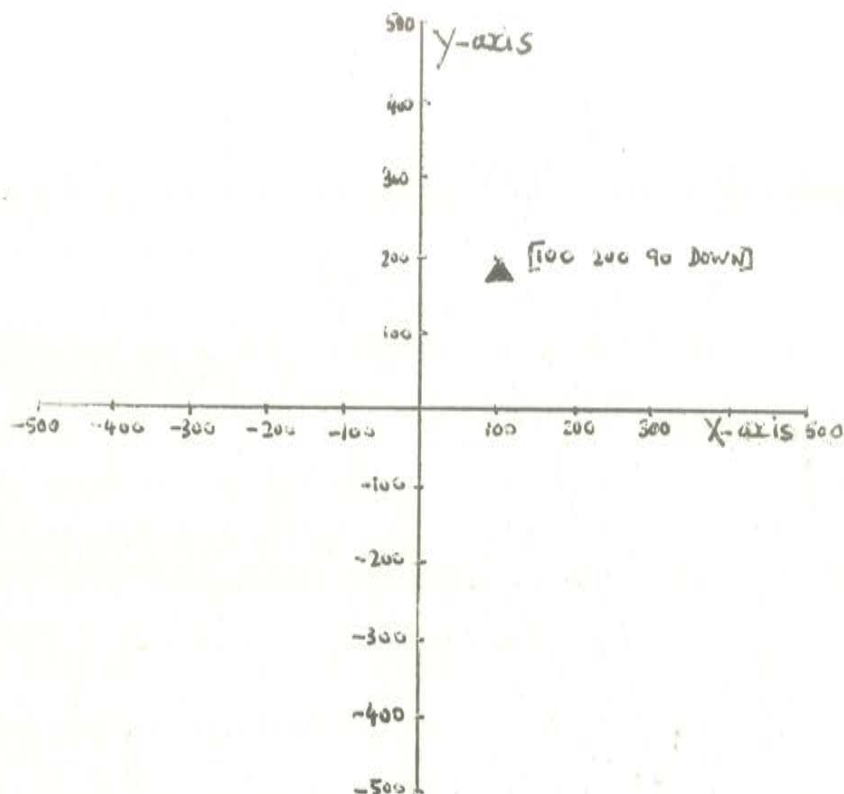
Like ARC it has two inputs. The first input is the radius of the circle. The second input is the amount the turtle turns in moving round the arc (so 360 degrees gives a whole circle).

SUMMARY

The new procedure is

NAME OF PROCEDURE	INPUT	EFFECT
ARCRIGHT	2 numbers	an arc is drawn.

APPENDIX C. THE TURTLE STATE



The turtle state is a list of four elements:

[the x coordinate, the y coordinate, the heading, the penstate]

There is a procedure named

STATE

whose result is the state of the turtle. It needs no inputs. You already know how to change the turtle state

E.g.

```
W: PRINT STATE
[100 200 90 DOWN]
W: LIFT
W: PRINT STATE
[100 200 90 UP]
W: FORWARD 30
W: PRINT STATE
[100 230 90 UP]
W: LEFT 15
W: PRINT STATE
[100 230 105 UP]
```


We can JUMP the turtle to change its state.

There is a procedure named

POSITION

which needs one input. This is a list of the new desired turtle state. The turtle will jump to the new position without leaving a line

W: POSITION [400 200 180 DOWN]

It is also possible to just jump the turtle along the x axis or the y axis. There is a procedure named

SETX

which needs one number input. The turtle will jump along the x axis so that this input becomes its new x coordinate.

W: SETX -100

W: PRINT STATE

[-100 200 180 DOWN]

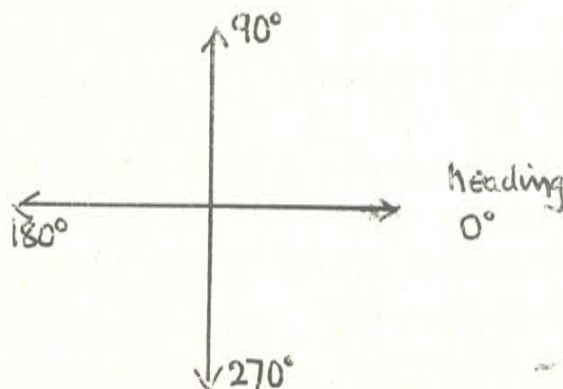
There is a similar procedure named

SETY

You can also set the turtle heading to a particular value by running a procedure named

SETHEADING

This procedure needs one number input. In LOGO the turtle always starts at zero degrees which is along the x axis.



Rotating to the left increases the heading

You can find out individual elements in the turtle state.

There are procedures named

XCOR
YCOR
HEADING
PEN

which need no input.

E.g.

W: PRINT HEADING
105
W: PRINT PEN
UP

SUMMARY

The turtle state can be changed without leaving a line. You can also find out what the turtle state is.

The new procedures are

NAME OF PROCEDURE	INPUTS	RESULT	EFFECT
POSITION	list of new turtle state	none	jumps to new state
SETX	number - new x coordinate	none	jumps along x axis
SETY	number - new y coordinate	none	jumps along y axis
SETHEADING	number - new heading	none	jumps to new heading
STATE	none	list of turtle state	none
XCOR	none	current x coordinate	none
YCOR	none	current y coordinate	none
HEADING	none	current heading	none
PEN	none	current pen state	none

APPENDIX D. THE LOGO CLOCK

LOGO has a clock which ticks in seconds. The clock starts at zero at the beginning of the session.

There is a procedure named

TIME

which needs no inputs. Its result is the time since the start of the session.

You can reset the clock to zero at any moment by running a procedure named

RESET

which needs no input.

One of the ways you can use the clock is to time how long it takes someone to reply in one of your quiz procedures.

For example

```

W: DEFINE "TIMER
  D: 10 PRINT [HOW MANY PEOPLE LIVE IN EDINBURGH]
  D: 20 RESET
  D: 30 IF EQUALQ REPLY [493281] THEN
    PRINT [EXCELLENT] ELSE PRINT [WRONG AGAIN]
  D: 40 PRINT [YOU TOOK]
  D: 50 PRINT TIME
  D: 60 PRINT "SECONDS
  D: END

```

SUMMARY

The new procedures are

NAME OF PROCEDURE	INPUT	EFFECT
TIME	none	gives time in seconds since start of session
RESET	none	resets LOGO's clock to zero

APPENDIX E. PAPER TAPE

There is another device you can use. It is a paper tape punch. To connect yourself to the paper tape punch run the procedure named

TAPE

This procedure needs no inputs.

You can have blank tape run out of the punch by running the procedure named

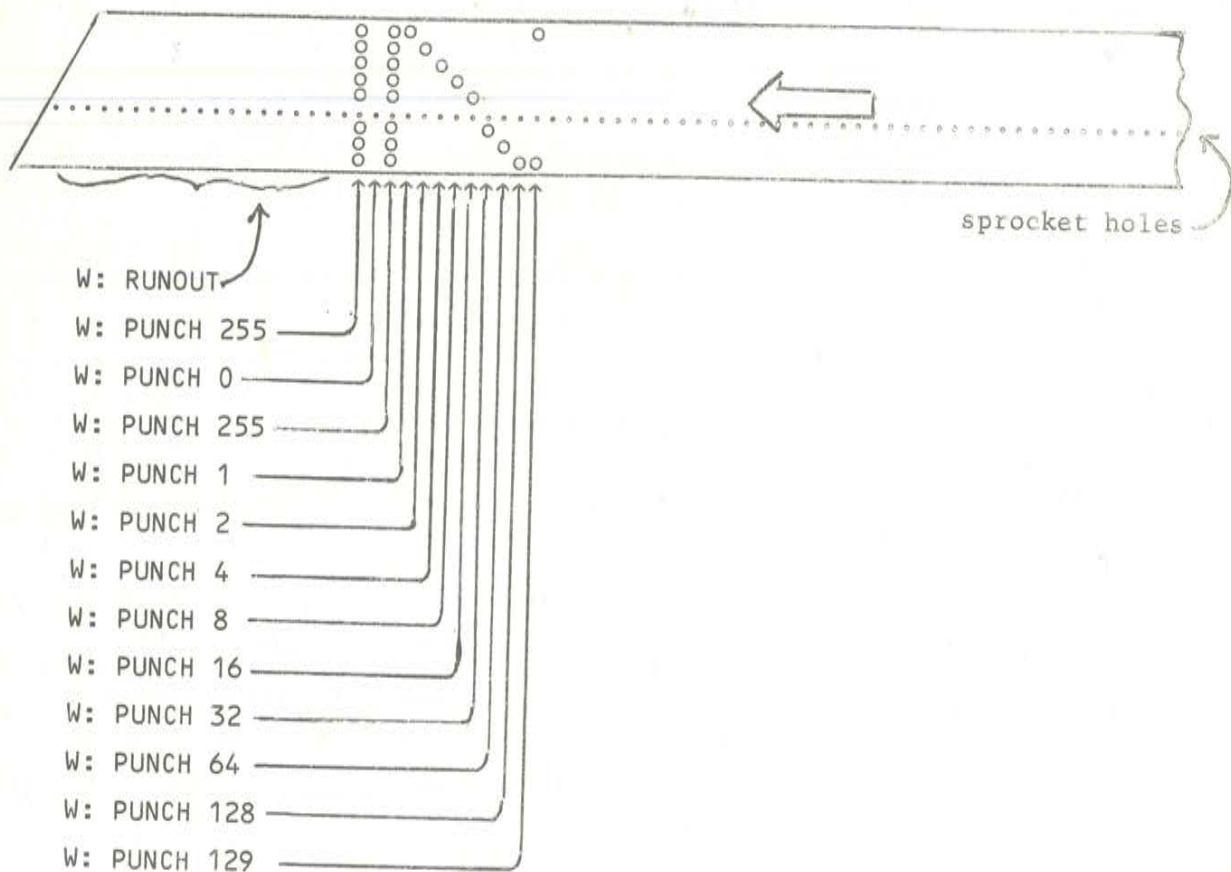
RUNOUT

This procedure needs no input.

You can punch holes in the paper tape by running the procedure named

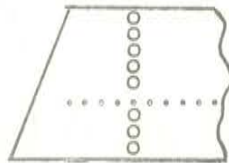
PUNCH

This procedure has one input which must be a number between 0 and 255.



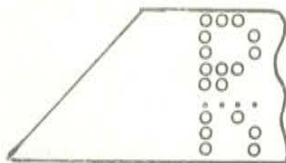
There are eight places across the tape where holes may be punched. This is like a binary number of eight digits. A hole corresponds to the digit 1. A blank corresponds to the digit 0.

Thus 255 (decimal) corresponds to 11111111 (binary) which corresponds to

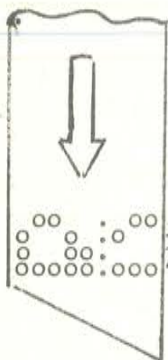


on the tape.

To punch a particular pattern of holes you must first translate the pattern in to a binary number. Then you must translate the binary number into a decimal number as input for procedure PUNCH



The letter R can be produced by running procedure PUNCH four times.



binary	decimal	LOGO
11111111 →	128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 → 255	W: PUNCH 255
10011000 →	128 + 0 + 0 + 16 + 8 + 0 + 0 + 0 → 152	W: PUNCH 152
10010100 →	128 + 0 + 0 + 16 + 0 + 4 + 0 + 0 → 148	W: PUNCH 148
01100011 →	0 + 64 + 32 + 0 + 0 + 0 + 2 + 1 → 99	W: PUNCH 99

SUMMARY

The new procedures are

NAME OF PROCEDURE	INPUTS	EFFECT
TAPE	none	connects to tape punch
RUNOUT	none	produces some blank tape
PUNCH	number between 0 and 255 inclusive	punches pattern of holes.

APPENDIX F. AN ABBREVIATION FOR VALUE

To save space in lines of LOGO you can abbreviate as follows:-

(a) W: PRINT VALUE "FRED

can be abbreviated to

W: PRINT :FRED

(b) or the SPIRAL procedure from Chapter 21.

```
W: DEFINE "SPIRAL "ANGLE "SIDE "STEP
  D: 10 FORWARD VALUE "SIDE
  D: 20 RIGHT VALUE "ANGLE
  D: 30 SPIRAL (VALUE "ANGLE)(ADD VALUE "STEP +
  C: VALUE "SIDE) VALUE "STEP
  D: END
```

could be written as

```
W: DEFINE "SPIRAL "ANGLE "SIDE "STEP
  D: 10 FORWARD :SIDE
  D: 20 RIGHT :ANGLE
  D: 30 SPIRAL :ANGLE (ADD :STEP :SIDE) :STEP
  D: END
```

Note that there is no space typed between the : and the name.

SUMMARY

VALUE " may be abbreviated to :

APPENDIX G. MORE ABOUT DEFINING PROCEDURES

If you use a particular procedure often you may wish to give it a shorter name. To do this you run the procedure named

ABBREVIATE

This needs two inputs. The first is the name of the procedure to be abbreviated. The second is the new abbreviated name e.g.

W: ABBREVIATE "ELEPHANT "LUMP

Now the procedure ELEPHANT can be run either by using the name ELEPHANT or the name LUMP. If you want to use this abbreviated name at another session you will have to REMEMBER the abbreviated procedure, e.g.

W: REMEMBER "LUMP

If you want to make more space between the lines of a procedure you would like to change you can use the procedure named

RENUMBER

This needs one input, the name of the procedure to be renumbered, e.g.

W: RENUMBER "LUMP

This would have the effect of renumbering the lines of LUMP in tens (i.e. 10, 20, 30, 40, ...) but keeping the order the same.

SUMMARY

The new procedures are

NAME OF PROCEDURE	INPUTS	EFFECT
ABBREVIATE	two quoted words	the second word becomes an alternative name for the procedure named with the first word.
RENUMBER	one quoted word	the lines of the procedure named are renumbered in tens.

APPENDIX H. GLUEING THINGS TOGETHER

Two words or numbers can be glued together to make larger words or numbers. The procedure that does this is named

WORD

and needs two inputs, e.g.

```
W: PRINT WORD "CAT "DOG
CATDOG
W: PRINT WORD "123 "345
123345
W: PRINT WORD "CAT "123
CAT123
```

Two lists can be made into one list by using the procedure named

JOIN

This works as follows

```
W: PRINT JOIN [HOW ARE][YOU TODAY]
[HOW ARE YOU TODAY]
```

A list can be made by using the procedure named

LIST

This takes two inputs, either of which may be a word list or number, e.g.

```
W: PRINT LIST "CAT "DOG
[CAT DOG]
W: PRINT LIST [THE CAT SAT ON THE MAT] 7
[ [THE CAT SAT ON THE MAT] 7]
```

Note that lists can have lists as elements.

SUMMARY

The new procedures are

NAME OF PROCEDURE	INPUTS	RESULT
WORD	two words or numbers	inputs are glued together into a larger word or number
JOIN	two lists	the lists are joined to make a single list
LIST	two words, numbers or lists.	a new list is made using the inputs as elements.

APPENDIX I. BOTH and EITHER

The results of two question procedures can be combined. This is useful if you want the control procedure IF to work on the results of two question procedures, e.g. in English:

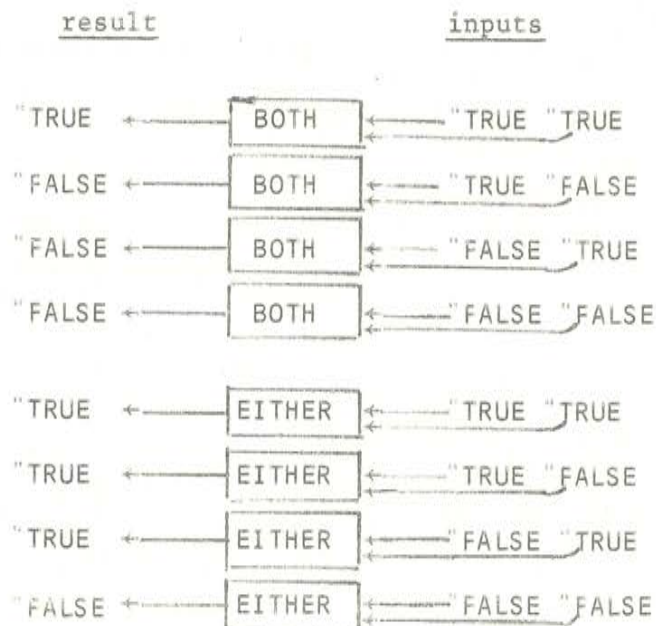
if both a sunny day and school holiday then go swimming.

if either hungry or thirsty then look in fridge.

The inputs for BOTH and EITHER must be the words "TRUE or "FALSE.

Try typing

W: PRINT BOTH "TRUE "TRUE
 W: PRINT BOTH "TRUE "FALSE
 W: PRINT BOTH "FALSE "TRUE
 W: PRINT BOTH "FALSE "FALSE
 W: PRINT EITHER "TRUE "TRUE
 W: PRINT EITHER "TRUE "FALSE
 W: PRINT EITHER "FALSE "TRUE
 W: PRINT EITHER "FALSE "FALSE



The words "TRUE and "FALSE would normally be the result of running the question procedures.

SUMMARY

The new procedures are:

NAME OF PROCEDURE	INPUTS	RESULT
BOTH	two words "TRUE or "FALSE	the word "TRUE or "FALSE
EITHER	two words "TRUE or "FALSE	the word "TRUE or "FALSE

APPENDIX J. THE END OF THE LIST

There are a number of procedures for working on the last element of a list (or bottom box of a stack). One of these is like FIRST and is named

LAST

E.g.

```
W: PRINT LAST [THE CAT SAT ON THE MAT]
MAT
```

Another is like REST and is named

BUTLAST

The result of this procedure is all-but-the-last.

E.g.

```
W: PRINT BUTLAST [THE CAT SAT ON THE MAT]
[THE CAT SAT ON THE]
```

Another is like PUT and is named

LASTPUT

E.g.

```
W: PRINT LASTPUT "DOG [THE CAT SAT ON THE]
[THE CAT SAT ON THE DOG]
```

LAST, BUTLAST and LASTPUT all work on words and numbers as well, in the same way that FIRST, REST and PUT do.

SUMMARY

NAME OF PROCEDURE	INPUTS	RESULT
LAST	a number or word or list	last digit last character last element
BUTLAST	a number or word or list	all but the last digit all but the last character all but the last element
LASTPUT	two inputs: numbers, words or lists	Puts first input behind second. Except lists cannot be put behind words or numbers.

APPENDIX K. WHILE

There is a control-procedure named

WHILE

which is a little like REPEAT. Instead of a number as first input WHILE needs "TRUE or "FALSE. This "TRUE or "FALSE would normally be the result of running a question procedure.

The second input to WHILE is a command, e.g.:-

W: WHILE "TRUE PRINT [AGAIN]

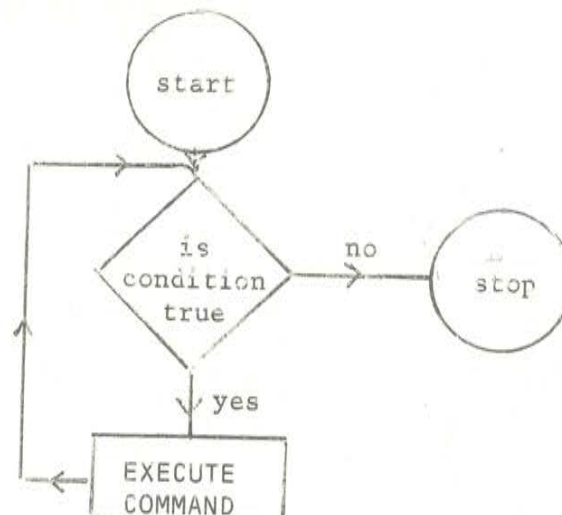
WHILE is also a little like IF, e.g.

W: IF "TRUE THEN PRINT [AGAIN]

WHILE is useful because it allows a command to be executed repeatedly while a condition is true.

While has two jobs to do:

- job 1: is the condition true?
- job 2: if the answer was true then execute the command and do job 1 again.
if the answer was false then WHILE has finished.



Here is an example using WHILE:-

```
W: WHILE GOODMOODQ PRINT "SMILE
```

The procedure GOODMOODQ must give either "TRUE or "FALSE as its result

For example:

```
DEFINE "GOODMOODQ
10 IF GREATERQ (RANDOM 10) 2 THEN RESULT "TRUE
    ELSE RESULT "FALSE
END
```

SUMMARY

The new control procedure is

NAME OF PROCEDURE	INPUTS	EFFECT
WHILE	true or false and command	executes command again and again while first input stays true.

APPENDIX L. AND

There is a LOGO control procedure named

AND

You can use it to make a line of LOGO which contains more than one command. For example, you can type

```
W: FORWARD 100 AND PRINT MULTIPLY 7 9
```

or

```
W: PRINT "ONE AND PRINT "TWO AND PRINT "THREE
```

after LOGO has executed the command FORWARD 100 the control procedure AND makes LOGO look for another command to execute. So LOGO then executes the command PRINT MULTIPLY 7 9 and prints 63.

AND can be useful when you want to do more than one thing depending on an IF. Suppose we want to write a procedure which can take any input but which will move the turtle if the input is a number and print OK. Without using AND we would write the procedure like this

```
W: DEFINE "CHOOSY "THING
  D: 1 IF NUMBERQ VALUE "THING THEN PRINT "OK
  D: 2 IF NUMBERQ VALUE "THING THEN FORWARD VALUE "THING
  D: END
```

If we use AND the procedure would look like this

```
W: DEFINE "CHOOSY "THING
  D: 1 IF NUMBERQ VALUE "THING THEN PRINT "OK +
  C: AND FORWARD VALUE "THING
  D: END
```

SUMMARY

The new control procedure AND can be placed between two commands.

APPENDIX M. WHICH LINE IS EXECUTED NEXT ?

Normally the commands in a procedure are executed in the order of the line numbers. This order can be changed by using the control procedure named

GO

in your definition. GO needs one input which must be a number. When a GO is found in a procedure the next line to be executed in that procedure is the one whose line number is the input to GO. Here is an example of a procedure which uses GO

```
W: DEFINE "HEXAGON
  D: 10 FORWARD 100
  D: 20 RIGHT GO
  D: 30 GO 10
  D: END
```

This procedure will draw a hexagon and keep on drawing over that hexagon.

GO is often used in THEN or ELSE

For example

```
W: DEFINE "QUESTION
  D: 10 PRINT [WHAT IS THE LENGTH OF THE THAMES]
  D: 20 IF EQUALQ REPLY [279] THEN PRINT [WELL DONE] ELSE GO 10
  D: END
```

SUMMARY

The new procedure is

NAME OF PROCEDURE	INPUT	EFFECT
GO	a number	changes which line is executed next.

APPENDIX N. RUNNING PROCEDURES

Normally a procedure is run by typing its unquoted name. But we can also run a procedure by using LOGO's procedure named

RUN

The first input for RUN must be the quoted name of a procedure. Any inputs for the procedure to be run must follow that quoted procedure name

e.g.

W: RUN "FORWARD 100

or

W: RUN "PRINT "FRED

RUN can be used inside a procedure. Here is an example

W: DEFINE "MUTTER "PROC

D: 10 PRINT RUN VALUE "PROC [THE CAT SAT ON THE MAT]

D: 20 PRINT [I HAVE JUST EXECUTED YOUR PROCEDURE NAMED]

D: 30 PRINT VALUE "PROC

D: END

W: MUTTER "FIRST

THE

[I HAVE JUST EXECUTED YOUR PROCEDURE NAMED]

FIRST

W: MUTTER "REST

[CAT SAT ON THE MAT]

[I HAVE JUST EXECUTED YOUR PROCEDURE NAMED]

REST

SUMMARY

The new procedure is

NAME OF PROCEDURE	INPUTS	RESULT or EFFECT
RUN	a quoted procedure name and its own additional inputs.	the result or effect of running the procedure whose quoted name is RUN's first input.

APPENDIX O. MECCANO

LOGO may be used to control two electric motors which can drive machines made with the Meccano set. To connect LOGO to the motors you should run the procedure named

MECCANO

This procedure needs no input. You should also turn the knob on the control box for the turtle and motors to the position marked MECCANO. Otherwise your commands to the meccano motors will be directed to the turtle's motors.

The procedure for rotating the shaft of a motor is named

ROTATE

This procedure needs a single, positive or negative, number as input. If the input is positive the motor will rotate clockwise. Normally the motor labelled "A" will be driven by procedure ROTATE. You can drive the other motor, labelled "B", by running a procedure named

MOTORB

This procedure needs no input. It acts like a switch. Having run it, all further ROTATE commands will drive motor B unless you switch back to motor A by running a procedure named

MOTORA

As well as driving one motor, it is also possible to drive the pair of motors at once. This can be done at any time, whichever way the switch is set, by running a procedure named

PAIR

This procedure, like ROTATE, needs a single positive or negative number as input and causes both shafts to rotate together.

SUMMARY

NAME OF PROCEDURE	INPUT	EFFECT
MECCANO	none	connects LOGO to the meccano motors
ROTATE	one number	rotates one motor shaft
PAIR	one number	rotates both motor shafts
MOTORA	none	selects motor A for procedure ROTATE
MOTORB	none	selects motor B for procedure ROTATE

APPENDIX P. HOW PROCEDURES ARE GIVEN THE VALUES OF THEIR INPUTS

This note shows how LOGO matches values to input names. It is a supplement to note 18. The matching depends on the order of the input names in the procedure title. LOGO does not look inside a procedure when it is matched to its inputs.

EXERCISE 1. Define the procedure

```
W: DEFINE "POLYBIT "LAMP "RHUBARB
      D: 10 FORWARD VALUE "RHUBARB
      D: 20 RIGHT VALUE "LAMP
      D: 30 WHERE
      D: END
```

Type in

W: POLYBIT

LOGO complained because there were no values to be matched to the names "LAMP and "RHUBARB

TITLE LINE	DEFINE	"POLYBIT	"LAMP	"RHUBARB
PROCEDURE CALL	POLYBIT			

EXERCISE 2. Type in

W: POLYBIT 400

This provided a value to match the first input "LAMP, but LOGO still complained because there was no value to match "RHUBARB

TITLE LINE	DEFINE	"POLYBIT	"LAMP	"RHUBARB
PROCEDURE CALL	POLYBIT	400		

EXERCISE 3. Type in

W: POLYBIT 400 10

This provides values to match both input names. The value 400 is matched with the first input "LAMP, and the value 10 is matched with the second input "RHUBARB

TITLE LINE	DEFINE	"POLYBIT	"LAMP	"RHUBARB
PROCEDURE CALL	POLYBIT	400	10	

In line 10 of POLYBIT, FORWARD uses the value of "RHUBARB and in line 20, RIGHT uses the value of "LAMP.

This matching of values to input names works in exactly the same way when a procedure is called inside another procedure. For example, consider the procedure

```
W: DEFINE "SHAPE "GORILLA "VIVA "HANDBAG
  D: 10 REPEAT VALUE "VIVA POLYBIT VALUE "HANDBAG VALUE "GORILLA
  D: END
```

Suppose we type in W: SHAPE 100 3 72

Then the first match that happens is:-

TITLE LINE	DEFINE	"SHAPE	"GORILLA	"VIVA	"HANDBAG
PROCEDURE CALL		SHAPE	100	3	72

Line 10 of SHAPE uses the values of the inputs as follows:

10	REPEAT	VALUE "VIVA	POLYBIT	VALUE "HANDBAG	VALUE "GORILLA
10	REPEAT	3	POLYBIT	72	100

Then LOGO matches values for the input names of POLYBIT

TITLE LINE	DEFINE	"POLYBIT	"LAMP	"RHUBARB
PROCEDURE CALL		POLYBIT	72	100

EXERCISE 4. Find out what is drawn by

W: SHAPE 100 3 72

EXERCISE 5. In each of the six examples below FIRST fill in the boxes to predict what you think will get drawn by using the three values 12, 20 and 30 in different orders as inputs for SHAPE.

A. When you type W: SHAPE 20 30 12 the drawing will have

- (a) How many sides? ☐
- (b) What turn between each side? ☐
- (c) What length for each side? ☐
- (d) The turtle finishing exactly where it started YES/NO
- (e) A gap between where the ink starts and where it finishes ... YES/NO

NOW check your answers by running SHAPE.

B. When you type W: SHAPE 30 20 12 the drawing will have

- (a) How many sides? ☐
- (b) What turn between each side? ☐
- (c) What length for each side? ☐
- (d) The turtle finishing exactly where it started YES/NO
- (e) A gap between where the ink starts and where it finishes YES/NO

NOW check your answers by running SHAPE.

C. When you type W: SHAPE 12 30 20 the drawing will have

- (a) How many sides? ☐
- (b) What turn between each side? ☐
- (c) What length for each side? ☐
- (d) The turtle finishing exactly where it started YES/NO
- (e) A gap between where the ink starts and where it finishes YES/NO

NOW check your answers by running SHAPE.

D. When you type W: SHAPE 30 12 20 the drawing will have

- (a) How many sides? ☐
- (b) What turn between each side? ☐
- (c) What length for each side? ☐
- (d) The turtle finishing exactly where it started YES/NO
- (e) A gap between where the ink starts and where it finishes ... YES/NO

NOW check your answers by running SHAPE.

E. When you type W: SHAPE 20 12 30 the drawing will have

- (a) How many sides? ☐
- (b) What turn between each side? ☐
- (c) What length for each side? ☐
- (d) The turtle finishing exactly where it started YES/NO
- (e) A gap between where the ink starts and where it finishes ... YES/NO

NOW check your answers by running SHAPE.

F. When you type W: 12 20 30 the drawing will have

- (a) How many sides? ☐
- (b) What turn between each side? ☐
- (c) What length for each side? ☐
- (d) The turtle finishing exactly where it started YES/NO
- (e) A gap between where the ink starts and where it finishes ... YES/NO

NOW check your answers by running SHAPE.

PROBLEM: Are there three numbers which used in any order as inputs for
SHAPE will always leave the turtle where it started?

(There is a small prize for the first correct solution.)

SUMMARY

When a procedure is called the order of values is matched to the order of
input names in the title line.

INDEX OF PROCEDURE NAMES

NAME OF PROCEDURE	TYPE	CHAPTER	PAGE	INPUTS	RESULT
ABBREVIATE	defining	G	128	2	no
ADD	arithmetic	9	34	2	yes
AND	control	L	137	0	no
ARC	drawing	9	34	2	no
ARCRIGHT	drawing	B	120	2	no
BACKWARD	drawing	2	8	1	no
BORROW	memory	16	59	1	no
BOTH	question	I	131	2	yes
BUTLAST	lists	J	133	1	yes
CARRIAGE	printing	A	118	0	no
CENTRE	drawing	2	10	0	no
CHANGE	defining	6	26	1	no
CLEAR	drawing	2	11	0	no
DEFINE	defining	4	18	variable	no
DEFINED	memory	16	58	0	yes
DELETE	defining	6	27	1	no
DISPLAY	drawing	2	8	0	no
DIVIDE	arithmetic	9	35	2	yes
DROP	drawing	2	10	0	no
EITHER	question	I	131	2	yes
EMPTYQ	question	22	80	1	yes
END	defining	4	18	0	no
EQUALQ	question	22	79	2	yes
FIRST	lists	27	93	1	yes
FORGET	memory	7	31	1	no
FORWARD	drawing	2	8	1	no
FREE	drawing	2	11	0	no
GO	control	M	138	1	no
GOODBYE	control	2	12	0	no
GREATERQ	question	22	79	2	yes

NAME OF PROCEDURE	TYPE	CHAPTER	PAGE	INPUTS	RESULT
HEADING	drawing	C	123	0	yes
HOOT	drawing	2	11	0	no
IF	control	23	81	1	no
JOIN	lists	H	129	2	yes
LAST	lists	J	133	1	yes
LASTPUT	lists	J	133	2	yes
LEFT	drawing	2	8	1	no
LESSQ	question	22	79	2	yes
LIFT	drawing	2	10	0	no
LIST	lists	H	129	2	yes
LISTQ	question	22	79	1	yes
MAKE	variables	32	112	2	no
MULTIPLY	arithmetic	9	35	2	yes
NOT	question	22	79	1	yes
NUMBERQ	question	22	79	1	yes
PEN	drawing	C	123	0	yes
PLOTTERA	drawing	2	8	0	no
PLOTTERB	drawing	2	8	0	no
POSITION	drawing	C	122	1	no
PRINT	printing	3	14	1	no
PUNCH	tape	E	125	1	no
PUT	lists	27	93	2	yes
RANDOM	arithmetic	33	115	1	yes
RECALL	memory	7	30	1	no
REMAINDER	arithmetic	9	36	2	yes
REMEMBER	memory	7	29	1	no
REMEMBERED	memory	16	58	0	yes
RENUMBER	defining	G	128	1	no
REPEAT	control	12	48	2	no
REPLY	control	24	84	1	yes
RESET	clock	D	124	0	no
REST	lists	27	94	1	yes
RESULT	control	19	70	1	yes

NAME OF PROCEDURE	TYPE	CHAPTER	PAGE	INPUTS	RESULT
RETITLE	defining	15	57	variable	no
RETURN	memory	16	60	0	no
RIGHT	drawing	2	8	1	no
RUN	control	N	139	variable	variable
RUNOUT	paper-tape	E	125	0	no
SAY	printing	A	118	1	no
SETHEADING	drawing	C	122	1	no
SETX	drawing	C	122	1	no
SETY	drawing	C	122	1	no
SHOW	defining	4	22	1	no
SPACE	printing	A	118	0	no
STATE	drawing	C	121	0	yes
STOP	control	25	86	0	no
SUBTRACT	arithmetic	9	35	2	yes
TAB	printing	A	118	0	no
TAPE	paper-tape	E	125	0	no
TIME	clock	D	124	0	yes
TRACE	debugging	26	90	1	no
TURTLE	drawing	2	8	0	no
TYPE	printing	A	119	1	no
TYPESET	printing	A	118	1	no
UNDEFINE	defining	7	31	1	no
UNTRACE	debugging	26	92	1	no
VALUE	variables	13	54	1	yes
WHERE	drawing	2	11	0	no
WHILE	control	K	135	2	no
WORD	words	H	129	2	yes
WORDQ	question	22	79	1	yes
XCOR	drawing	C	123	0	yes
YCOR	drawing	C	123	0	yes
ZEROQ	question	22	79	1	yes

DIFFERENT TYPES OF PROCEDURE

TYPE	PROCEDURE NAME	PAGE	TYPE	PROCEDURE NAME	PAGE
ARITHMETIC	ADD	34	DRAWING	SETX	122
	DIVIDE	35		SETY	122
	MULTIPLY	35		STATE	121
	RANDOM	115		TURTLE	8
	REMAINDER	36		WHERE	11
	SUBTRACT	35		XCOR	123
CLOCK				YCOR	123
	RESET	124	LISTS	BUTLAST	133
	TIME	124		FIRST	93
CONTROL	AND	137		JOIN	129
	GO	138		LAST	133
	GOODBYE	12		LASTPUT	133
	IF	81		LIST	129
	REPEAT	48		PUT	93
	REPLY	84		REST	94
	RESULT	70	MEMORY	BORROW	59
	RUN	139		DEFINED	58
	STOP	86		FORGET	31
	WHILE	135		RECALL	30
DEBUGGING				REMEMBER	29
	TRACE	90		REMEMBERED	58
	UNTRACE	92		RETURN	60
DEFINING	ABBREVIATE	128	PAPER-TAPE	PUNCH	125
	CHANGE	26		RUNOUT	125
	DEFINE	18		TAPE	125
	DELETE	27	PRINTING	CARRIAGE	118
	END	18		PRINT	14
	RENUMBER	128		SAY	118
	RETITLE	57		SPACE	118
	SHOW	22		TAB	118
	UNDEFINE	31		TYPE	119
DRAWING				TYPESET	118
	ARC	34	QUESTION	BOTH	131
	ARCRIGHT	120		EITHER	131
	BACKWARD	8		EMPTYQ	80
	CENTRE	10		EQUALQ	79
	CLEAR	11		GREATERQ	79
	DISPLAY	8		LESSQ	79
	DROP	10		LISTQ	79
	FORWARD	8		NOT	79
	FREE	11		NUMBERQ	79
	HEADING	123		WORDQ	79
	HOOT	11		ZEROQ	79
	LEFT	8	VARIABLES	MAKE	112
	LIFT	10		VALUE	54
	PEN	123	WORDS	WORD	129
	PLOTTERA	8			
	PLOTTERB	8			
	POSITION	122			
	RIGHT	8			
	SETHEADING	122			

MARKERS AND PROMPTS

TYPE	SYMBOL	PAGE
MARKERS	"	14
	[]	16
	()	40
	THEN ELSE	81
	+	81
	:	127
	↑	14
	←	13
	-	78
PROMPTS	W:	7
	D:	17
	REPLY:	84
	C:	81
	INT:	73
CHARACTER SET	A - Z	
	0 - 9	
	(and also MARKERS)	

ERRATA

- Page 4 Exercise 7: replace 'store button' by 'define button'
- Page 38 Snapshot 3: replace 'SUM' by 'ADD'
- Page 42 Snapshot 2: replace 3 instances of 'RIGHT 60' by 'RIGHT 90'
- Page 85 Exercise 5: replace 'D:' by 'W:'
and replace 4 instances of 'W:' by 'D:'
- Page 95 Summary: replace 'EFFECT' by 'RESULT'
- Page 97 Exercise 1: replace 'BOXES' by '"BOXES'
- Page 110 Exercise 3: should be Exercise 4
replace section 'This procedure
will rest of the list' by
'This procedure will do two jobs very
much like SCRAMBLE.'
- Job 1: if there are no more words in
the list then the result is
the empty list.
- Job 2: otherwise PUT the scrambled
first word of the list at the
beginning of the coded rest
of the list.'
-
- Page 116 Exercise 5: line 50 of procedure
insert '+' after SECRET

