Let's look at a version of POLYSPI with a stop rule and consider exactly what happens when it is run:

```
TO POLYSPI : LENGTH
   IF :LENGTH > 15 THEN STOP
   FD :LENGTH
   RT 90
   POLYSPI ( :LENGTH + 5 )
END
```

This is what happens when we run POLYSPI 10. The POLYSPI procedure is called and a local variable is defined with its value set at 10. Since this value is not greater than 15, LOGO proceeds to carry out the movement FD 10 RT 90, and then makes a new call to POLYSPI, but this time with an input value of 15. This causes a copy of the procedure to be called again. Because LENGTH is not greater than 15, the turtle is made to move FD 15 RT 90, and another call to POLYSPI is made. But this time, the local variable has been increased to 20, so the procedure stops and returns control to the procedure that called it (POLYSPI 15). This procedure in turn has come to its final line, and returns control to its calling procedure. This also stops, at which point the program has come to its end.

We have shown how recursion in LOGO involves procedures calling copies of themselves. It is important to keep in mind that the recursive calls are copies that exist alongside the original procedure, working as if they were completely different from it. When finished, such a procedure always returns control to the procedure that called it. To illustrate more clearly the process of returning from procedure calls, we can rearrange POLYSPI in this way:

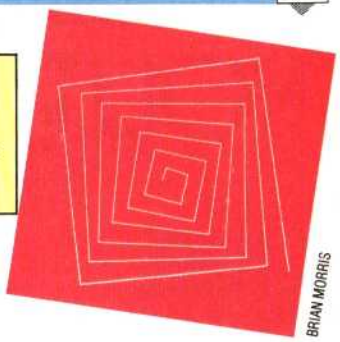
```
TO POLYSPI :LENGTH
   IF :LENGTH > 15 THEN STOP
   POLYSPI ( :LENGTH + 15 )
   FD :LENGTH
   RT 90
END
```

If you run this you will see that the program does its drawing 'backwards': the lines are drawn spiralling inwards rather than outwards. (This will be shown more clearly if you use a larger value in the condition statement — for example, using 50 instead of 15.) What is significant here is that LOGO draws each line as control is returned from the procedure calls. In our previous example, a line was drawn and control was then passed to another procedure. But here, all the procedures are called before any drawing begins, and the last created value of LENGTH is the one used first.

Finally, we should note that recursion is a technique that uses up a lot of memory. Procedures in which the recursive call is in the last line are the most efficiently implemented, however, as they don't take up any extra memory no matter how many times they're called. If a procedure can be written so it is 'end recursive' then this is usually worth doing.

Procedure Problem 3

Write a recursive procedure to draw a tower of squares one on top of the other, halving the length of the side each time.

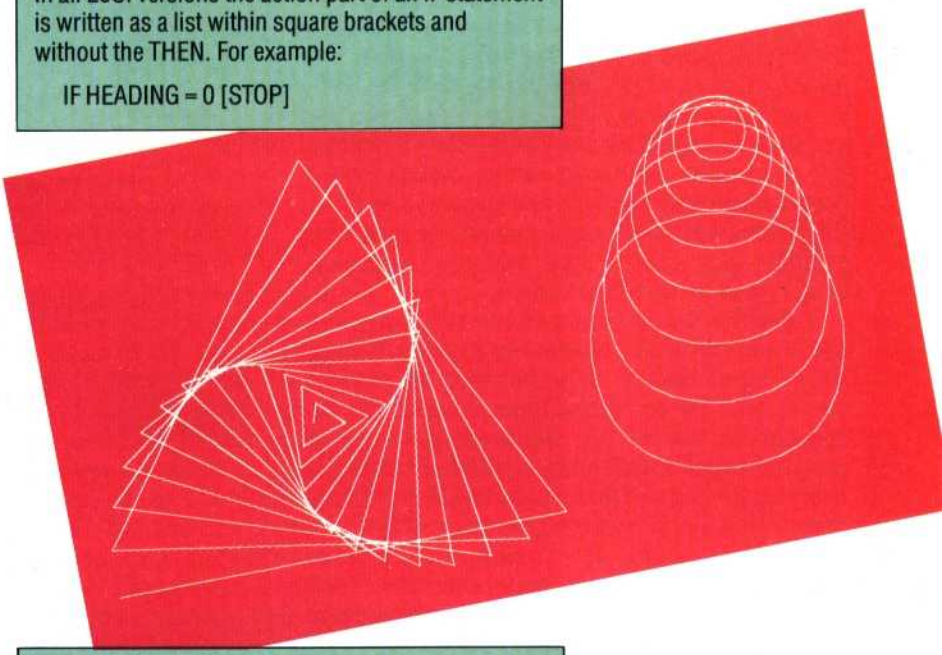


BRIAN MORRIS

Logo Flavours

In all LCSI versions the action part of an IF statement is written as a list within square brackets and without the THEN. For example:

```
IF HEADING = 0 [STOP]
```



Exercise Answers

A procedure to draw a circle, given the radius as input, with the present position as a point on the circumference:

```
TO CIRCLE :RADIUS
   REPEAT 36 [FD ( 2 * :PI * :RADIUS / 36 )
   RT  10]
END
MAKE "PI (3.14159)
```

If we adapt this procedure so that the centre of the circle is at the present position we get:

```
TO C.CIRCLE :RADIUS
   PU LT 90 FD :RADIUS RT 90 PD
   CIRCLE :RADIUS
   PU LT 90 BK :RADIUS RT 90 PD
END
```

TARGET uses C.CIRCLE to draw 5 concentric circles:

```
TO TARGET
   C.CIRCLE 10 C.CIRCLE 20 C.CIRCLE 30
   C.CIRCLE 40 C.CIRCLE 50
END
```