!

LOGO is an ideal language for exploring mathematics. You begin by developing a few procedures for basic arithmetic tasks, and then use these primitives to perform quite complex calculations. We demonstrate how the language is used to calculate factorials, and show how a few results are transformed into 'factorial trees'.

In how many ways can you arrange four people in four chairs around a table? The first person can be seated in any one of the four seats, but once he has sat down only three choices remain for the second, then there are two choices for the third, and for the last there is only one place left. So, the total number of different arrangements is $4 \times 3 \times 2 \times 1$. This is usually written as 4! and read as '4 factorial'. Factorials are often found in mathematics problems concerning arrangements, combinations and probabilities.

It is simple to write a recursive definition to calculate factorials. First of all, we must note that the factorial of 0 is defined as 1. The factorial of any non-zero positive number — x say — is the factorial of x−1 multiplied by x. Translating this into a program we get:

```
TO FACTORIAL :X
  IF :X = 0 THEN OUTPUT 1
  OUTPUT (FACTORIAL :X — 1) * :X
END
```

To try it out, type PRINT FACTORIAL 6 — the result should be 720.

This procedure works fine up to 12, but beyond this the numbers become too large to be held as integers by the computer. On the Commodore 64, for example, PRINT FACTORIAL 13 gave 6.22702E9 — that is, 6.22702 times 10^9. This is hardly satisfactory, as the last four digits have been lost. There are many reasons (including simple curiosity) why we might want to know what these remaining digits are. The first thing we need to do, therefore, is extend the arithmetic capabilities of LOGO so that it can calculate to greater than seven figure accuracy.

To simplify matters, we will only consider positive integers. We'll represent the integers as lists — so we will represent 1,234,567 as [1 2 3 4 5 6 7]. The following two procedures will do addition on such numbers. Try them out with PRINT LONGADD [1 2 3] [5 6 9] — the result should be [6 9 2]:

```
TO LONGADD :X :Y
  OUTPUT LONGADD1 :X :Y 0
END

TO LONGADD1 :X :Y :CARRY
  IF (ALLOF (EMPTY? :X) (EMPTY? :Y) (:CARRY =
  0)) THEN OUTPUT []
  TEST EMPTY? :Y
  IFTRUE IF :CARRY = 0 THEN OUTPUT :X ELSE
  OUTPUT LONGADD1 :X [1] 0
  TEST EMPTY? :X
  IFTRUE IF :CARRY = 0 THEN OUTPUT :Y ELSE
  OUTPUT LONGADD1 [1] :Y 0
  MAKE "SUM (LAST :X) + (LAST :Y) + :CARRY
  OUTPUT LPUT REMAINDER :SUM 10 LONGADD1
  BUTLAST :X BUTLAST :Y QUOTIENT :SUM 10
END
```

These procedures work in much the same way as we would do additions on paper, adding from the left and incorporating any number carried from the previous column.

Subtraction is a similar process. However, we have included a routine to delete leading zeros from an answer, so that we don't end up with results such as [0 0 0 7 8].

```
TO LONGSUB :X :Y
  OUTPUT STRIPZEROS LONGSUB1 :X :Y 0
END

TO LONGSUB1 :X :Y :BORROW
  IF (ALLOF (EMPTY? :X) (EMPTY? :Y) (:BORROW =
  0)) THEN OUTPUT [0]
  TEST EMPTY? :Y
  IFTRUE IF :BORROW = 0 THEN OUTPUT :X ELSE
  OUTPUT LONGSUB1 :X [1] 0
  IF EMPTY? :X THEN PRINT [SORRY, I CAN'T
  HANDLE A NEGATIVE RESULT] TOPLEVEL
  MAKE "DIFF (LAST :X) — (LAST :Y) — :BORROW
  IF :DIFF <0 THEN OUTPUT LPUT (10 + :DIFF)
  LONGSUB1 BUTLAST :X BUTLAST :Y 1
  OUTPUT LPUT :DIFF LONGSUB1 BUTLAST :X
  BUTLAST :Y 0
END

TO STRIPZEROS :X
  IF EMPTY? :X THEN OUTPUT [0]
  IF NOT ((FIRST :X) = 0) THEN OUTPUT :X
  OUTPUT STRIPZEROS BUTFIRST :X
END
```

Long multiplication is slightly more complicated. We'll implement it using the technique normally taught in schools. For example, supposing we want to multiply 123 by 338. The problem is split up into three parts: first we multiply 123 by 8; then we multiply 123 by 330; and, finally, we add the two results together. This method depends on the fact that the second stage can be broken down into two sub-stages: firstly, 123 is multiplied by 33; and then a zero is placed at the end of the result. To multiply a number by 33 clearly involves the use of