FIGURE IT OUT

In this instalment of our LOGO course, we look at the facilities the language offers for working with numbers. LOGO would probably not be the first choice of language for applications that require a lot of calculation, but it does offer an impressive array of numerical primitives.

Almost all LOGO implementations support both integer and real (decimal) arithmetic, using the infix operators + - * / . These operators are called 'infix' because they are written between the numbers they work on — for example, 3+4. Some LOGOs also include 'prefix' arithmetic, in which our example would be written as SUM 3 4. One advantage of this notation is that it is consistent with the way in which other LOGO operations and commands are written.

MIT LOGO supports infix arithmetic only, but it is simple to program prefix forms if they are required. Define SUM and PRODUCT and try them:

```
TO SUM :A :B
  OUTPUT :A + :B
END

TO PRODUCT :A :B
  OUTPUT :A * :B
END
```

The 'precedence' of operations (the order in which they are carried out) follows the usual mathematical rules. Anything within brackets is done first, followed by multiplications and divisions, and finally additions and subtractions:

```
PRINT (3 + 4) * 5
PRINT 3 + 4 * 5
```

Now try the prefix forms:

```
PRINT PRODUCT 5 SUM 3 4
PRINT SUM 3 PRODUCT 4 5
```

This demonstrates another advantage of the prefix forms — there is no need for rules of precedence and the line is evaluated in the same way as any other line of LOGO commands.

The usual division operation (/) gives the result as a real number. Two other operations, QUOTIENT and REMAINDER, are often useful for working with integers.

```
QUOTIENT 47 5 is 9
REMAINDER 47 5 is 2
```

A standard method for converting a number in base 10 to binary is to keep dividing the number by two until the result is zero. The binary number is found by writing the remainders found at each stage in reverse order. For example, to convert 12 to binary:

```
12/2 = 6; remainder = 0
6/2  = 3; remainder = 0
3/2  = 1; remainder = 1
1/2  = 0; remainder = 1
```

So, reading the remainders upwards, we find that decimal 12 is 1100 in binary.

Using QUOTIENT and REMAINDER we can implement this technique easily in LOGO. By putting the print statement *after* the recursive call we get the remainders printed in the correct (reverse) order.

```
TO BIN :X
  IF :X = 0 THEN STOP
  BIN QUOTIENT :X 2
  PRINT1 REMAINDER :X 2
END
```

Two operations exist for rounding numbers — INTEGER and ROUND. INTEGER outputs the whole number part of a number, simply ignoring any figure after the decimal point, and ROUND rounds a number up or down to the nearest whole number.

The following procedures calculate the compound interest on an investment at a given rate of interest. In PRETTY.PRINT, INTEGER is used to get the pounds, and ROUND is used to round the pennies to the nearest whole number.

```
TO COMPOUND :PRINCIPAL :RATE :YEARS
  IF :YEARS = 0 THEN PRETTY.PRINT
  :PRINCIPAL STOP
  COMPOUND :PRINCIPAL * ( 1 + :RATE / 100 )
    :RATE :YEARS — 1
END

TO PRETTY.PRINT :MONEY
  MAKE "POUNDS INTEGER :MONEY
  MAKE "PENCE ROUND ( :MONEY —
    :POUNDS ) * 100
  ( PRINT :POUNDS "POUNDS :PENCE
    "PENCE)
END
```

TESTING TIME

We have already used =, <, and > as logical tests in a number of procedures. The logical operations ALLOF, ANYOF and NOT can be used to combine other tests. ALLOF is true if both its inputs are true, ANYOF is true if either of its inputs is true, and NOT is true if its input is false. So we get:

```
IF ANYOF :X > 0 :X = 0 THEN PRINT "POSITIVE
IF NOT :X < 0 THEN PRINT "POSITIVE
IF ALLOF :X > 0 :X < 100 THEN PRINT
  [BETWEEN 0 AND 100]
```

LISSAJOUS FIGURES