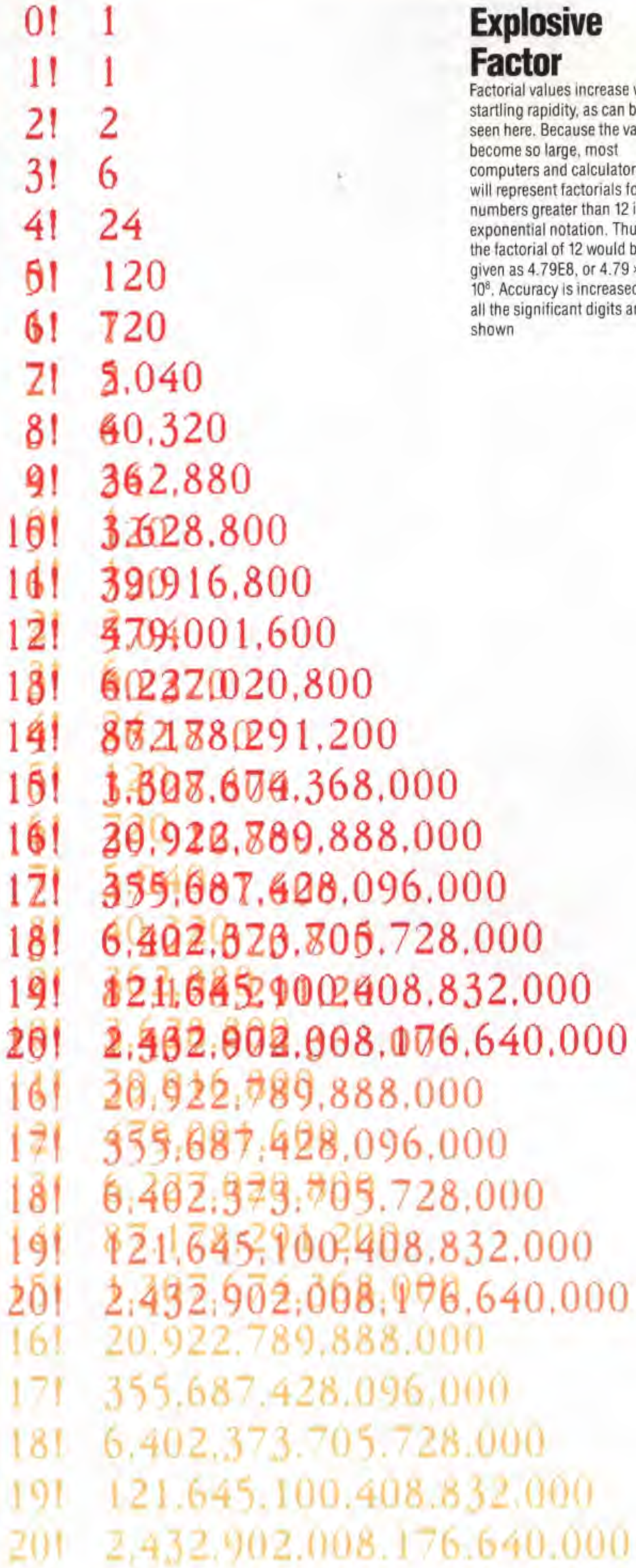

Explosive Factor

Factorial values increase with startling rapidity, as can be seen here. Because the values become so large, most computers and calculators will represent factorials for numbers greater than 12 in exponential notation. Thus, the factorial of 12 would be given as 4.79E8, or 4.79 × 10^8. Accuracy is increased if all the significant digits are shown

IAN McKINNELL

recursion. The procedure LONGMULT controls this general strategy:

```
TO LONGMULT :X :Y
   IF EMPTY? BUTLAST :Y THEN OUTPUT
   LONGMULT1 :X LAST :Y 0
   OUTPUT LONGADD (LONGMULT1 :X (LAST :Y)0)
   (LPUT "0 LONGMULT :X BUTLAST :Y)
END
```

The details of multiplying a line by a single digit are carried out by LONGMULT1:

```
TO LONGMULT1 :X :NO :CARRY
     TEST EMPTY? :X
     IFTRUE IF :CARRY = 0 THEN OUTPUT [] ELSE
     OUTPUT (LIST :CARRY)
     MAKE "PROD (LAST :X) * :NO + :CARRY
     OUTPUT LPUT REMAINDER :PROD 10
     LONGMULT1 BUTLAST :X :NO QUOTIENT
     :PROD 10
  END
```

We won't need procedures to perform division for calculating factorials, but you might care to extend the system to cover division for yourself.

We now have a set of primitives for carrying out arithmetic to any degree of precision. The only limitation on the size of numbers that can be handled is the total memory space available to the program.

MAKING MODIFICATIONS

We can now modify our original factorial program to use our new form of long multiplication.

```
TO FACT :X
   IF FIRST :X = 0 THEN OUTPUT [1]
   OUTPUT LONGMULT (FACT LONGSUB :X [1]) :X
END
```

To try it out type FACT [1 3]; you should get [6 2 2 7 0 2 0 8 0 0] as the result. There are problems, however. The calculation process is slow, and — on the Commodore 64 — the largest factorial we obtained before running out of memory was 34!, which has 39 digits (and took some time to be calculated).

The expression of large numbers as lists looks rather unusual, but we can modify the program to overcome this problem by translating back and forth between our usual notation and the list form. We employ two procedures — EXPLODE and IMPLODE — to do this.

EXPLODE 123 outputs [1 2 3] and IMPLODE [1 2 3] outputs 123

```
TO EXPLODE :X
   IF EMPTY? :X THEN OUTPUT[]
   OUTPUT (SENTENCE FIRST :X EXPLODE
   BUTFIRST :X)
END
```

```
TO IMPLODE :X
   IF EMPTY? :X THEN OUTPUT"
   OUTPUT (WORD FIRST :X IMPLODE
   BUTFIRST :X)
END
```