Top Gear

By paying careful attention to variables and program structure, you can speed up the operation of almost any Basic program

BASIC is, despite what its critics say, a versatile language and a powerful educational aid. You can write any program in BASIC, provided your machine has enough memory and the execution time is not important. However, because BASIC is usually interpreted rather than compiled (see page 184), it can be painfully slow in executing programs — especially those that require the same instruction to be translated and executed repeatedly.

Sorting, for example, is a highly repetitive process: the procedure is carried out within a loop, and there are smaller loops nested inside the main loop (see page 286). If 100 items are to be sorted, the program may make between 2,500 and 5,000 iterations of the loop. A BASIC sort will always be slow, but the way the code is written can make a significant difference to the speed of execution. If an instruction is to be repeated 5,000 times, and if coding it properly can save one hundredth of a second of execution time for each repetition, then there will be a total saving of 50 seconds — a considerable improvement for the user.

To observe the difference that good and bad coding can make, you will need a timing mechanism and a 'testbed' program. If you own a Commodore computer, you can use the system clock, with the associated variables TI$ and TI, as part of the testbed program. If your computer doesn't have an accessible clock, you'll have to use a stopwatch to time the code in execution. It is also a good idea to make your program 'beep' at you when it starts and finishes, so that you'll know when it's operating.

The testbed program looks like this:

```
1000 L=500
2000 PRINT "***GO***":REM "BEEP"
       instructions here
2100 TI$="000000"
2200 FOR K=1 TO L
.....
2900 NEXT K:T9=TI
2950 REM "BEEP" instructions here
3000 PRINT "******STOP******"
3100 PRINT "That took    "; (T9/60); "    seconds"
```

Lines 2100 and 3100 are for Commodore users. For other machines, delete or replace them with appropriate code. The space between lines 2200 and 2900 is where we will put the code to be timed. Notice that the timings will refer to L repetitions where L is the limit of the loop. Testing only one execution of a piece of code would be very inaccurate because the system clock measures only in 60ths of a second, and there is a timing overhead imposed by the code of the testbed program as well.

Here are some general rules for writing efficient BASIC, roughly in order of importance:

1. Avoid all arithmetic in loops.

Exponentiation (x^3, meaning 'x raised to the power of 3', for example), and mathematical functions (cos(y), meaning 'the cosine of the angle y', for example) are particularly slow. Multiplication and division are slower processes than addition and subtraction, but even the quickest of these operations (addition) is relatively slow.

In the testbed program insert these lines:

```
900 Z=1.1
2300 X=Z↑3
```

and run it. On our test machine 500 repetitions took 27.95 seconds. Now replace line 2300 with:

```
2300 X=Z*Z*Z
```

and run it. This took 3.55 seconds — a dramatic difference!

Further investigation will reveal the level of exponentiation at which it becomes worthwhile replacing repeated multiplication by the exponentiation function. On our computer this was at the 18th power (when X=Z↑18). Remember, however, that to calculate $Z^{2.3}$, for example, repeated multiplication would be useless, whereas the exponentiation function (↑) works for all real numbers, including negative ones.

Use the testbed program to see how long the other arithmetic processes take, and compare alternatives. Is it quicker to divide a number by 2, or multiply it by 0.5, for example?

2. Use variables rather than numerical constants.

Every time a numerical constant (7,280 for example) occurs in a BASIC instruction, time is spent translating the number into usable form. Try this line:

```
2300 X=X+7280
```

On our machine that took 4.63 seconds to execute 500 repetitions, whereas: