Alternative Translation

Computers 'think' in machine code; programmers prefer to write in a high level language such as Basic. Compilers and interpreters offer different methods of translation between them

When computers were first developed they didn't have keyboards. Program instructions had to be entered one step at a time by setting each of eight switches to 'up' or 'down', to represent a single operation. These patterns of 'up' and 'down' were examples of machine code.

It was logical to replace the switches by a typewriter keyboard, and replace the patterns of switch settings by real English words. The result was the 'high-level' language such as BASIC, replacing the low-level machine codes.

As processors, however, computers did not change, but continued to work on the original patterns of switches (and still do), so programmers had to develop programs written in the original low-level notation to translate these high-level programs into patterns that the processors could work on. These low-level programs came to be called interpreters or compilers, according to their method of translation.

In computing (as elsewhere), any gain in power or speed has to be paid for — in money, time or freedom of action. So it is with interpreters and compilers. Together they provide all the program translation facilities that a programmer needs. Interpreters are strong in some areas and compilers in others, but each pays for its advantages with compensating disadvantages.

Interpreters, usually built into the home computer, are the cheap way of translating high-level language programs into something a computer can understand. They don't use up much memory — leaving more space for your programs.

Micros costing less than about £400 almost invariably feature a BASIC interpreter: you type in a BASIC program, type RUN, and either the program works, or it stops with an error message from the system — something like:

```
SYNTAX ERROR ON LINE 123
```

So you type LIST, find the error, correct it, type RUN, and it either works or stops again, and so on. Note that some of the more sophisticated BASIC interpreters actually check for syntax errors as each line is entered.

You may have done this sort of thing hundreds of times without having given a thought to the interpreter. Its chief virtue is precisely that it is an invisible device that allows you to work on your program without ever bothering about where it is in memory or how to execute it — the program is at your fingertips, and you can RUN it, LIST it, or EDIT it immediately.

The interpreter is easy to use, but not very sophisticated: every time you type RUN, the interpreter has to find your BASIC program in memory and translate and execute it line by line. If your program contains this loop:

```
400 LET N=0
500 PRINT N
600 LET N=N+1
700 IF N<100 THEN GOTO 500
```

the interpreter has to translate and execute lines 500 to 700 a hundred times, as if it had never encountered them before.

Compilers are different. They're expensive, difficult to write, and occupy and use a lot of memory. They are almost always disk-based software, so the user needs an expensive system.

What they offer is flexibility, power and speed; faced with the four lines of BASIC above, a compiler would translate them all once, then execute that code a hundred times.

This allows quite a saving in time – but at a price. Suppose you have a BASIC compiler and you want to enter and run a BASIC program.

First you load and run the File Creation Program (called the Editor), which allows you to type in the program and save it to disk as a 'source file'.

Files must be named so that you can find them once you've created them (just like files in a real filing cabinet), so the Editor asks you to name the source file. File names often consist of two parts: the first is a label, any name you choose – say MYPROG – and the second part is usually a three-letter code indicating the nature of the file contents; this code is the 'extension'. A BASIC file might have the code BAS as its extension. Your source file is now on disk under the name MYPROG.BAS. Now, typing:

```
COMPILE MYPROG.BAS
```

will cause the computer to LOAD and RUN the BASIC compiler on a BASIC source file called MYPROG.BAS.

You wait a few seconds, depending on the length of your program while the compiler translates your program into an 'object file', which it saves on disk under the name MYPROG.OBJ – the OBJ extension indicating that this is the object file, a machine code translation of a source file.