# PRIMITIVE PARTS

**An algorithm is a series of instructions that describe how some process may be performed. A knitting pattern is an algorithm; so is a recipe; and so is a computer program. We discuss how an understanding of the principles of algorithms can improve your programming.**

An algorithm describes a process either in terms of other processes that have already been defined, or in terms of processes that are so basic that they do not need to be defined. Thus, in a recipe, one instruction may be 'prepare a bechamel sauce', where a bechamel sauce recipe (algorithm) has been given elsewhere in the cookbook. Another instruction may be 'bring the mixture to the boil', where the operation of bringing something to the boil is assumed to be fully understood by the user. In programming terms, algorithms are constructed from instructions that either use algorithms (procedures, routines, functions) written elsewhere in the program, or ones built into the language (commands such as PRINT and DIM, or maths functions like LOG and TAN).

This article looks at how algorithms are constructed from other algorithms and *primitive* processes. The primitives at the disposal of a programmer are the commands and functions in the language. From these, algorithms are written that can do small things (move a sprite, say, or accept a number as an input). These algorithms are then used to build more general algorithms (updating the game display, or controlling a menu system), and these algorithms are in turn used as parts of larger ones again until the whole program, viewed as a single algorithm, is written in terms of lower-level algorithms. This concept is the basis of what is known as *structured* or *modular* programming and is a subject we will return to later in the course.

## DESIGNING ALGORITHMS

An algorithm has an input and an output. This is just to say that, as a process, the algorithm will work on some initial data to produce a result. This initial data is passed to the algorithm from outside in the guise of 'parameters', which remain constant for any particular use of the algorithm but may change between different uses. Passing parameters will be familiar to even a novice programmer since the simple program:

```
10 PRINT "Hello World!"
```

passes the parameter 'Hello World!' to the algorithm called by the PRINT command. Similar examples are FNA(P), TAN(P), LEFT$(P$,5) and POKE P,5, where P, P$ and 5 are all parameters. In the same way, an algorithm's results are passed back as parameters. If the programming language being used has local variables (e.g. PASCAL and C) then the parameters would normally be passed with a procedure call, as in:

```
procedure(parameter1,parameter2,etc.);
```

It is an essential first step in designing an algorithm to consider the contents of the input and output parameters, their types (integer, floating point, real, string, etc.) and their magnitudes and ranges.

When the input and output of the algorithm have been defined, the next step is to form ideas as to how one can be transformed into the other. Unfortunately, there is no 'cookbook' method for creating these transformations as they require creativity and ingenuity. However, there are several ways of helping the process along.

The most obvious and most often overlooked is to borrow the algorithm from somewhere else. At the simplest level, the in-built functions of a programming language provide many useful algorithms, such as string-handling, trigonometric functions, input/output, and (possibly) sorting and matrix manipulation. Apart from this, the algorithm needed may already exist in another of your programs. The code for this could be incorporated in the new program (creating your own library of algorithms is extremely useful). In addition, there are published collections of algorithms that are often available from public libraries. *The Art of Computer Programming, Vol I: Fundamental Algorithms* by D. E. Knuth, although heavy going, is highly recommended.

Programs published in the computer magazines are well worth scrutinising for routines that may be of use. Finally, there are algorithms that are used in other pertinent domains, and although they were never meant for computing they are extremely useful. The accountancy section of the local library will be full of books containing formulae for calculating balances and depreciation. A little research among these could make writing an accounts program a lot easier and the result is likely to be a lot more reliable. The same is true for other disciplines: engineering, electronics, maths, etc.

Whether adapting an existing algorithm, or creating one from scratch, there are certain criteria that must be applied to each instruction it contains. These are *definiteness* and *effectiveness*. Definiteness means that the instruction should