Keeping Control

All versions of Basic feature 'control structures' that govern the flow of a program. Some machines, however, offer a wide range of alternatives, with subtle differences

The first 10 parts of the Basic Programming course have covered almost all of the more important aspects of the BASIC language. In this issue we will present an overview of the topics we have covered so far, deal with a few interesting asides and give some pointers to where we shall go next.

First the overview: a high-level language such as BASIC provides the user with a set of instructions that are translated internally into a form the computer can understand. Any computer program can be written using just two simple patterns, called 'constructs'. These are 'sequence' constructs and 'control structures' of which only two are essential in BASIC: IF . . . THEN . . . ELSE and WHILE . . . DO. Most other computer languages provide considerably more.

The sequence construct allows the task to be broken down into a set of sub-tasks that perform the main task when executed in sequence. The size of the sub-tasks depends on the language; in BASIC the sub-tasks are represented by the statements written on each line, and the sequence is represented by the line numbers. Thus, if the task is to multiply the value assigned to a variable by 10, the sequence we could use might be:

```
110 INPUT N
120 LET N = N * 10
130 PRINT N
```

In addition to sequence constructs, we also need control structures. These are constructs that alter the order of execution of statements in a program.

The simplest control structure provided by BASIC is GOTO. This is an unconditional jump (or branch) that re-directs the execution of the program to a specified line number without a test or condition having to be satisfied. GOSUB is also an unconditional branch, but the program will always RETURN to the point immediately after the GOSUB and its use in structured programming is perfectly acceptable.

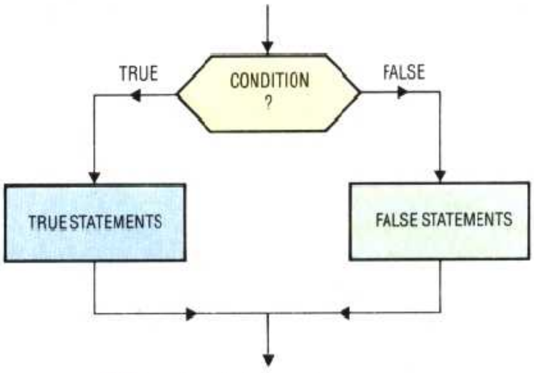
The IF . . . THEN . . . ELSE control structure is available in BASIC. It takes the form of the IF . . . THEN statement and has the following syntax ('syntax' is the computer jargon for 'form'):

> IF (specified condition) is true THEN execute specified statement (ELSE) execute the next statement

Note that in standard BASIC, the ELSE part of IF . . . THEN . . . ELSE is implied. In some BASIC dialects and in certain other languages, PASCAL for example, ELSE forms part of the statement.

IF . . . THEN . . . ELSE (IF . . . THEN in BASIC) performs one of two sub-tasks depending on whether a certain condition is true or not. Consider the following program, which is designed to find the square roots of numbers input from the keyboard unless a 'flag' value of −9999 is input (in order to terminate the program):

```
10 PRINT "INPUT A NUMBER"
20 INPUT N
30 IF N = -9999 THEN GOTO 70
40 LET S = SQR(N)
50 PRINT "THE SQUARE ROOT OF ";N;" IS ";S
60 GOTO 10
70 END
```



The IF . . . THEN . . . ELSE Control Structure
If the condition is True, the True statements will be executed. If the condition is False, the False statements will be executed

What line 30 is really saying here is 'IF it is true that N = −9999 THEN go to the end of the program ELSE (if it is not true that N = −9999) execute the next line of the program to find the square root'.

The other essential control structure (WHILE . . . DO) is not directly available in BASIC, but it can easily be simulated. WHILE . . . DO is a type of 'do-loop' and it means 'repeat a statement or set of statements WHILE a specified condition is true' or 'WHILE a condition is true DO something'.

WHILE . . . DO always tests the condition before the statements are executed, so if the test fails first time through, the statements (called the body of the loop) are not executed. As an example, consider a games program that prompts the player to 'PRESS SPACE-BAR WHEN READY'. This part of the program could be written (in 'pseudo-language' or simplified English) as:

```
WHILE space-bar is not pressed
DO scan keyboard
start game
```

In BASIC this could be written: