16 Conditional and loop instructions

Programs and parts of programs can be made to execute over and over again either continuously, or a specified number of times. The instructions you put in your program to make this happen are called LOOP instructions.

The FOR . . . NEXT loop

The most common type of loop is **FOR. . . NEXT**, which uses a variable to count the number of repetitions required.

```
10 FOR N = 1 TO 6
20 PRINT N
30 NEXT N
>RUN
         1
         2
         3
         4
         5
         6
```

In this program, N is printed at each pass through the loop. N is called the control variable.

You can start the control variable at any number you choose, and you may alter the amount by which it changes on each pass, the step size.

```
10 FOR N = 7 TO 11 STEP 1.6
20 FOR J = 20 TO 10 STEP -5
30 PRINT N,J
40 NEXT J
50 NEXT N
>RUN
         7        20
         7        15
         7        10
       8.6        20
       8.6        15
```

```
         8.6          10
        10.2          20
        10.2          15
        10.2          10
```

This program shows that you can use decimal step sizes, or negative step sizes. You may start the control variable at any value; and you may use **FOR. . . NEXT** loops within each other. This is called nesting, and you can nest as many loops as you wish.

Type

```
LISTO 2
```

followed by [RETURN] , and then LIST the program again.

```
10 FOR N = 7 TO 11 STEP 1.6
20   FOR J = 20 TO 10 STEP -5
30     PRINT N,J
40     NEXT J
50   NEXT N
```

Each loop is shown indented from the previous one.

LISTO is a list option instruction and can take any number from 0 to 7.

LISTO0 is the normal setting – it lists exactly what the computer has stored in memory.

LISTO1 lists the program with an extra space after each line number.

LISTO2 lists the program with indents on **FOR . . . NEXT** loops.

LISTO4 lists the program with indents on **REPEAT . . . UNTIL** loops.

These effects may be obtained in any combination by adding the numbers together, so **LISTO3** would give extra spaces after line numbers and indented **F0R . . . NEXT** loops.

Here are some further points on the use of **FOR . . . NEXT** loops.

(i) You do not need to specify the control variable to which **NEXT** refers. The following program will work exactly the same as the one above.

```
10 FOR N = 7 TO 11 STEP 1.6
20 FOR J = 20 TO 10 STEP -5
30 PRINT N,J
40 NEXT
50 NEXT
```

The computer assumes that **NEXT** applies to the loop it is in at the present moment.

If you do put the variable names after **NEXT**, but get them mixed up, then this is what happens.

```
10 FOR N = 7 TO 11 STEP 1.6
20 FOR J = 20 TO 10 STEP -5
30 PRINT N,J
40 NEXT N
50 NEXT J
>RUN
         7         20
        8.6        20
       10.2        20


No FOR at Line 58
```

The computer starts to execute the N loop before the J loop, and when it reaches line 58 it cannot find the **FOR** to go with **NEXT J**. Loops must be nested one within another; they must not cross.

(ii) The number of **FOR**s, and the number of **NEXT**s must be the same. The following program does not give an error, but it is left hanging in midair.

```
10 FOR N = 7 TO 11 STEP 1.6
20 FOR J = 20 TO 10 STEP -5
30 PRINT N,J
40 NEXT
         7         20
         7         15
         7         10
```

If you do this, you will run into trouble.

(iii) You must never jump out of a **FOR...NEXT** loop using **GOTO**. As in (ii) above this will often not result in an error, but the program will be impossible to follow.

(iv) The stop condition for a loop is that, for a positive step size, the control variable is greater than the terminating value; for a negative step size, the control variable is less than the terminating value. However, all loops will be executed at least once.

```
10 FOR NUMBER = 6 TO 0
20 PRINT NUMBER
30 NEXT
>RUN
         6
```

When the loop has been completed, the control variable moves on an extra step, so the above program will end up with **NUMBER** equal to 7. Here is a program to show this:

```
10 FOR Size = 100 TO 103 STEP 1.5
20 PRINT "INSIDE LOOP, Size = ";Size
30 NEXT
40 PRINT "OUTSIDE LOOP, Size = ";Size
INSIDE LOOP, Size = 100
INSIDE LOOP, Size = 101.5
INSIDE LOOP, Size = 103
OUTSIDE LOOP, Size = 104.5
```

(v) **FOR...NEXT** loops are used when you wish to go around a loop a fixed number of times. There may be as many lines as you like between the **FOR** and its corresponding **NEXT**, and control variables need not be directly assigned with numbers. They can be assigned with any arithmetic expression, containing variables or other functions.

```
10 MODE 5
20 FOR angle = 0 TO 2*PI STEP .1
30 PLOT 69,649 + 440*SIN(angle), 512 + 400*
COS(angle)
40NEXT
```

The REPEAT . . . UNTIL loop

Another very useful lwp is **REPEAT . . . UNTIL**, which waits until a condition is fulfifled before coming out of the loop.

```
10 INPUT'"This program turns decimals into
fractions"'"Give me a decimal: "decimal
20 numerator% = 1: denominator% = 1
30 PRINT "Program running"
40 REPEAT
50 fraction = numerator%/denominator%
60 IF fraction>decimal THEN denominator% =
denominator%+1
70 IF fraction<decimal THEN numerator% = nu
merator%+1
80 UNTIL fraction = decimal
90 PRINT;decimal;" is equal to ";numerato
r%;"/"denominator%'"Program end"
```

This program asks you to input a decimal. It then prints out the fractional equivalent. (Don't input too complicated a decimal or the program will run for hours.) Lines 50 and 60 are repeated until the condition at line 80 is fulfilled. In this example, the condition is that fraction = decimal.

Line 80 is called a conditional statement, and the result of a conditional statement may either be **TRUE** or **FALSE**. In the example shown above, the statement becomes **TRUE** when fraction is equal to decimal, so the program loop is repeated only whilst the conditional statement is **FALSE**. Of course, the computer doesn't understand **TRUE** and **FALSE**, so it assigns numeric values to these conditions:

0 for **FALSE**, –1 for **TRUE**.

There are a number of logical operators which can be used in conditional statements:

A = B	True when A is equal to B
A < B	True when A is less than B
A > B	True when A is greater than B
A <= B	True when A is less than or equal to B
A >= B	True when A is greater than or equal to B

A <> B	True when A is not equal to B
NOT A	True when A is false
TRUE	True always
FALSE	False always
A AND B	True if both A and B are true
A OR B	True if either A or B is true, or if both are true
A EOR B	True if either A or B is tme, but false if both are true

There is more about logic operations in the next section on the **IF** statement.

REPEAT. . . UNTIL is easily followed and understood by other people who read your programs, and should be used in preference to **GOTO**.

IF . . . THEN . . . ELSE

The **IF** statement enables the computer to make a choice about something.

```
10 REPEAT
20 A$ = GET$
30 IF A$="Y" THEN PRINT "YES"; ELSE PRINT
 A$;
40 UNTIL FALSE
```

This program turns the Y key into a YES-button. Line 30 contains a conditional statement. If the condition is tme then the computer obeys whatever comes after **THEN**. If the statement is false then the computer carries out whatever comes after **ELSE**.

IF statements can carry out more than one instruction, if these instructions are placed on the same line and are separated by colons:

```
10 REPEAT
20 A$ = GET$
30 IF A$="Y" THEN FOR I = 1 TO 6 : PRINT
"YES";: NEXT ELSE PRINT A$;
40 UNTIL FALSE
```

and now you will get six YESs when you type Y.

These multi-statement lines are not restricted to the **IF** statement. Any line in a program may carry out more than one instruction if each is separated by a colon. However, it is generally better to use a procedure rather than fill an **IF** statement full of colons. Procedures we explained in chapter 17.

IF statements may ask for a complex condition, using the logical operators **AND**, **OR**, and **EOR**.

```
10 REPEAT
20 A$ = GET$
30 B$ = GET$
40 IF A$ = "Y" AND B$ = "Y" THEN PRINT "YES
"; ELSE PRINT A$;B$
50 UNTIL FALSE
```

This program will only print **YES** if you type two Ys in succession.

The **ELSE** part of the statement is optional, and may be omitted.

Alternatively you can extend the **IF** statement by chaining it:

```
10 REPEAT
20 A$ = GET$
30 IF A$ = "Y" THEN PRINT "YES" ELSE IF A$
= "N" THEN PRINT "NO" ELSE PRINT "MAYBE"
40 UNTIL FALSE
```

This program demonstrates the use of one IF statement after another . . .

Using the **IF** statement you can now find out some more about how the computer deals with **TRUE** and **FALSE**.

```
10 X=8>6
20 Y=6>8
30 PRINT X,Y
>RUN
        -1         0
```

Because 8 is greater than 6, 8 > 6 is **TRUE**, so X is –1. 6 > 8 is **FALSE** so Y is 0.

```
10 REPEAT
20 INPUT X
```

```
30 IF X THEN PRINT;X; " IS TRUE" ELSE PRINT
;X; " IS FALSE"
40 UNTIL 0
```

This program allows you to enter numbers, and to see whether the computer treats them as **TRUE** or **FALSE**. You will see that only 0 is treated as **FALSE**, all other values being **TRUE**.

The above program can be rewritten:

```
10 REPEAT
20 INPUT X
30 IF X THEN PRINT ;X;" IS TRUE" ELSE PRINT
;X;" IS FALSE"
40 UNTIL FALSE
```

which has exactly the same effect.

Another important use of **IF** is with strings and string variables. For example, you might find this in the middle of a program

```
100 REM The answer should be 560
110 INPUT "SO WHAT'S THE ANSWER THEN?"'X
120 IF X = 560 THEN A$ = "YES" ELSE A$ = "N
O"
130 IF A$ = "YES" THEN PRINT A$;" WELL DONE
" ELSE PRINT A$;" TRY AGAIN"
```

This will test to see what string A$ contains, and will print one of two messages accordingly.

Less than and greater than can also be used:

```
10 A$ = "HELLO"
20 IF A$ < "GOODBYE" THEN...
```

This will be true, because the IF statement compares the ASCII values of the first character in each string. If the first two characters are the same, then the next two characters are compared, and so on. So ‘MELON’ is less than ‘MELTED’. This is very useful for putting strings into alphabetical order, but it does not work if the strings are a mixture of upper and lower case letters.

The following operators may be used with strings:

=	the same as
<>	not equal to
<	less than
>	greater than
<=	less than, or equal to
>=	greater than, or equal to