Ch3 : Loops

A very frequent programming requirement is to perform a set of instructions several times. Rather than writing the set of instructions several times (which is impractical for all but a small number of repetitions), they are controlled by a special instruction which causes them to be repeated as many times as desired.

Such program constructs are called loops, and each repetition of a set of statements is often called an iteration.

The For loop

Suppose a program is required to read 10 numbers, add each one to a running total and then display the final total. The program in Example 3.1 accomplishes this task using a loop.

```
1    Sub Main()

2       'This program accepts 10 numbers and adds them up.
3       'It then displays the result

4        'Variables
5        Dim Number As Integer
6        Dim Total As Integer
7        Dim Count As Integer

8        Console.Clear()
9        Console.WriteLine("Enter ten Numbers")
10       Total = 0
11       For Count = 1 To 10
12            Number = Console.ReadLine()
13            Total = Total + Number
14       Next Count

15       Console.WriteLine("The Total is  :{0}", Total)

16       Console.ReadKey()
17  End Sub
```

Code Analysis

1	Start of subroutine Main ( )
2 & 3	Brief comment describing the program
4	Variables section
5,6,7	Three Integer variables (memory locations) **Number, Total & Count**
8	Console.Clear() this statement clears the console screen
9	Informs the user to enter 10 numbers.
10	The variable **Total** is set to 0
11	For loop starts. The For loop has a counter (in this case **count**) a start value (in this case 1) and an end value (in this case 10). The statements inside the For loop will continue giving a new value to **count** after each iteration.
12	Readline ( ) allowed data to be read from the user. The input is placed into the variable (memory location) named **Number**.
13	**Total** becomes equal to itself plus the value in memory location called **Number** i.e. **Number** is added to the **Total**
14	End of the loop construct. Count is given a new value and the loop will repeat. The loop will continue to repeat until count reaches its final value 10.

15	The Total is displayed on the screen.
16	Readkey ( ) serves only to keep the screen visible until the user presses a key.
17	Ends the subroutine. And the program.

A FOR loop is used to repeat the two instructions which repeatedly read a number and add it to a running total.

The FOR loop requires that a control variable, called Count in this example, is defined as type integer. The control variable is automatically given the first value specified (1 in this example) and, each time the statements within the loop are repeated, it is increased by 1 until it finally reaches the second value specified (10 in this example).

This same program could be used to add any number of numbers by changing the end value in the FOR statement.

This next example uses a for loop to print a table of conversion Inches to Centimetres

```
1   Sub Main()

2         'This program prints a conversion table Inches to Centemetres.

3         Const CONVERSIONFACTOR = 2.54
4         Const MAXINCHES = 12
5         'Variables
6         Dim Inches As Integer
7         Dim Centimetres As Decimal

8         Console.Clear()
9         Console.WriteLine("{0,20} {1,20}", "Inches", "Centimetres")
10        Console.WriteLine ("{0,20} {0,20}", "---------")
11        For Inches = 1 To MAXINCHES
12             Centimetres = Inches * CONVERSIONFACTOR
13             Console.WriteLine("{0,20} {1,20}", _
                        Inches.ToString(), Centimetres.ToString("0.00"))
14        Next
15        Console.ReadKey()

16     End Sub
```

Inches	Centimetres
1	2.54
2	5.08
3	7.62
4	10.16
5	12.70
6	15.24
7	17.78
8	20.32
9	22.86
10	25.40
11	27.94
12	30.48

Code Analysis

1	Start of subroutine Main ( )
2	Brief comment describing the program
3	Const. Set **CONVERSIONFACTOR** = `2.54` Used in conversion calculation
4	Const. Set **MAXINCHES** = `12` Used to set limit of FOR loop (i.e. loop will run 12 times so table will show 12 rows
5	**Variables section**
6	DIM (Declare in Memory) a location named Inches. It will be allowed to store Integer values
7	DIM (Declare in Memory) a location named Centimetres. It will be allowed to store decimal values
8	`Console.Clear()` this statement clears the console screen
9	Write the letters in quotes " " are written to the console screen. Note the brackets {0,20} 0 to use the first argument from the list and 20 specifies "field width" –effectively right justification within 20 charactes. {1,20} as above but uses the second argument in the list.
10	This line displays the underline for each column heading.
11	**Start of the FOR loop** The For loop has a counter (in this case **Inches)** a start value (in this case 1) and an end value (in this case **MAXINCHES**). The statements inside the For loop will continue giving a new value to **Inches** after each iteration.
12	**Centimetres** is calculated by multiplying Inches by `CONVERSIONFACTOR.` The result is placed into location named **Centimetres**
13	The values for **Inches** and **Centimetres** are converted to sting and printed on screen using the same field width as the headings above (to ensure that they line up). An appropriate format for Centimetres is specified "0.00" to ensure each value uses two decimal places.
14	End of the loop construct. Inches is given a new value and the loop will repeat. The loop will continue to repeat until **Inches** reaches its final value **MAXINCHES** (set at 12).
15	Readkey ( ) serves only to keep the screen visible until the user presses a key.
16	Ends the subroutine. And the program.

Notice that the end value in the FOR statement is a constant called MAXINCHES; this could also have been defined as a variable used in a ReadLine() instruction. A slight variation in the format of a FOR statement allows the count variable to go up in increments other than 1. You may specify a Step clause as shown below:

```
For count = 1 to 100 step 2
        …..
Next
```

Would increment the count by two at the end of each loop.

Likewise to count down from a high value to a low value, you could write

```
For count = 12 to 1 step –1
        …..
Next
```

which would cause the variable count to start at 12 and go down to 1 in steps of 1.

The While loop

The FOR statement is a very useful means of implementing a loop, but certain programming problems require a different approach to repeating a set of instructions. For example, consider the following outline program description:

> *Read a set of decimal numbers representing the cost of a number of items. Accumulate the total cost of the items until a value of zero is entered, then display the number of items purchased and their total cost.*

Here it is **not known** how many times the loop is to be repeated: the user decides when to terminate the loop by entering *a rogue value,* (zero in this case). The rogue value is used in another type of loop instruction, the while instruction. Example 3.3 shows how a while loop can be used in conjunction with a rogue value.

The rogue value is defined as a constant. Because the user may want to terminate the program immediately, without entering any values, the program asks for a purchase amount before entering the loop starting (on line 14).

The **while** instruction requires that a true/false expression is included after the word `while'. Thus the expression, **Amount > ROGUEVALUE,** will be true if **Amount** entered is greater than zero, and it will be false if **Amount** is not greater than zero, that is if it is equal to, or less than zero.

When the expression is true, the statements between the **While** and **End While**, that is lines 16 to 19, will be executed; as soon as the expression becomes false, the loop terminates and the program goes on to line 21.

Notice that the last instruction in the lines to be repeated is the ReadLine( ) instruction to read another value: this means that because the next instruction to be executed is the **while** instruction (line 15), the value typed in by the user is immediately compared with the rogue value. This ensures that the rogue value is not processed as an actual data item.

```
1    Sub Main()
2    'This program demonstrates a while loop.
    ' A set of values are entered (ends with rogue value 0)
    ' The total cost and number of items purchased are printed.

3        Const ROGUEVALUE = 0

4        'Variables
5        Dim Count As Integer
6        Dim Amount As Decimal
7        Dim Total As Decimal

9        'set variables
10        Count = 0
11        Total = 0

12        Console.Clear()
13        Console.Write("Enter the value of the first Item (0 to exit)  :")
14        Amount = Console.ReadLine()
15        While Amount > ROGUEVALUE
16            Total = Total + Amount
17            Count = Count + 1
18            Console.Write("Enter the value of the next Item (0 to exit)  :")
19            Amount = Console.ReadLine()
20        End While

21        Console.WriteLine("{0} Items were purchased.", Count)
22        Console.WriteLine("The Total Cost was : {0:c}", Total)
23        Console.ReadKey()

24   End Sub
```

```
Enter the value of the first Item (0 to exit)  :10
Enter the value of the next Item (0 to exit)   :20
Enter the value of the next Item (0 to exit)   :30
Enter the value of the next Item (0 to exit)   :0
3 Items were purchased.
The Total Cost was :£60.00
```

Notice that the assignment instruction on line 17, Count: = Count + 1, is used as a means of counting the number of times the loop is executed. The instruction simply adds 1 to the variable, **Count,** each time the instructions within the loop are repeated.

The true/false expression on line *15* in the **while** statement uses the *relational operator, >*, meaning `greater than', to compare **Amount** with **ROGUEVALUE**. There are in fact six different relational operators that can be used in such logical expressions, and these are shown below

A set to 10 B set to 5 C set to 5 D set to 0

Expression	Example	Result
> greater than	A > B	True
	B > C	False
>= greater then or equal to	A >= B	True
	B >= C	True
< less than	A < B	False
	B < C	False
<= less than or equal to	A <= B	False
	B <= C	True
<> not equal to	A <> B	True
	B <> C	False
= equal	A = B	False
	B = C	True

The operators in Table 3.1 are used according to the relationship to be established between two values, Whatever Logical comparison is used the result is either true or false. If true the while loop will repeat –if false the loop will terminate. More examples of logical operators will be covered in the next section.

Exercises

1 Write a program using a loop to add up all the numbers from 1 to 10 and to display the result.
Which loop did you use and why?

2 Write a program that prints a table as shown below:

Number	Square	Cube
1	1	1
2	4	8
3	9	27
…	…	…
10	100	1000

3) Write a program to calculate and print the average of a set of decimal.
Which loop did you choose and why?