

# Branching(Testing) and Repeating(Looping) Statements



| [**Branching(Testing) Statements:**](#gjdgxs)   * [If...Then](#30j0zll) * [Case Select](#1fob9te) | [**Repeating(Looping) Statements:**](#3znysh7)   * [Do...Loop Until](#2et92p0) * [Do...Loop While](#tyjcwt) * [Do While...Loop](#3dy6vkm) * [Do Until...Loop](#1t3h5sf) * [For...Next](#4d34og8) * [While...Wend](#2s8eyo1) |
| --- | --- |

Visual Basic like other programming languages executes lines of code from the top of the program downwards. However, often a programmer does not wish all lines to be executed every time the program is run. Instead of writing several different programs for this job, branching and repeating commands can be used.

Branching commands are used to determine when to execute a small section of code and when not to run it. One example is the ‘If’ command. If something is true then the code will be run, if not it will not be executed and VB will continue further down the program.

Repeating commands are useful for running small sections of code several times. If a programmer needs to read 200 lines of text from a file he could use 200 lines of code, one for every read statement, or alternatively a ‘For Next’ loop could do a similar job using just three or four lines!

## ***Branching(Testing) Statements***

### If...Then

Single line If (Example):

**If bytDayOfWeek = 1 Then Label1.Caption = "Sunday"**

Multi-line If (Example):

**If blnLeapYear = True And bytMonth = 2 Then  
 Print "Current year is a leap year"  
 Print "Month is February"  
 Print "Thus number of days in month = 29"  
 End If**

The multi-line If command requires the extra statement ‘End If’ to define the end of the If block.

In the following code a check is made to determine the price of cinema seats. Normal seats are priced at �4.40 while luxury seats are �6.00. Since there are only two types of seats the code only checks for standard and assumes luxury if this is false.

If and Else (Example):

**If strSeatType = "normal" Then  
 Label1.Caption = "Price = �4.40"  
 Else  
 Label1.Caption = "Price = $6.00"  
 End If**

However, if we adapt this cinema example to the theatre where there are more types of seats then we will require an ‘ElseIf’ statement.

If and ElseIf (Example):

**If strSeatType = "stalls" Then  
 Label1.Caption = "Price = �8.25"  
 ElseIf seat = "circle"  
 Label1.Caption = "Price = �10.50"  
 Else ' Assume the Upper Circle  
 Label1.Caption = "Price = �12.00"  
 End If**

### Case Select

If more branches are required then a ‘Select Case’ command can be used. Notice that in the following example, several values can trigger the same case statement. For example, the months 1, 3, 5, 7, 8, 10 and 12 all are 31 days long. Instead of writing 12 separate case statements, all these values can be included in one case by using commas.

Case (Example):

**Select Case bytMonth  
 Case 1, 3, 5, 7, 8, 10, 12  
 number\_of\_days = 31  
 Case 2  
 number\_of\_days = 28  
 Case 4, 6, 9, 11  
 number\_of\_days = 30  
 End Select**

The ‘End Select’ statement defines the end of the Case block. To ‘trap’ any values not picked up by earlier cases used the ‘Case Else’ construct.

**...  
 Case 2  
 number\_of\_days = 28  
 Case 4, 6, 9, 11  
 number\_of\_days = 30  
 Case Else  
 MsgBox "ERROR: Month must be between 1 and 12!"  
 End Select**

## ***Repeating(Looping) Statements***

The easist way to understand repetition is with the aid of a practical example. Consider the steps a milkman performs has he delivers pints of milk to all the houses in a street. There are four basic steps: 1) check number of pints requested on customer’s note, 2) fetch required pints from milk float, 3) carry back empties, and 4) move to next house in street. If the street has 20 houses then the milkman has to carry out 80 individual steps to deliver all the milk. A computer simulation of this could be written in Visual Basic using 80 lines of code, one line for each step. However, this is very inefficient. A much better way is to ‘wrap’ the four basic steps in some kind of repeating structure. Visual Basic supports the following types:

* Do...Loop Until
* Do...Loop While
* Do While...Loop
* Do Until...Loop
* For...Next
* While...Wend

### Do...Loop Until

This type of repeating statemente will always be executed at least once because the loop exit statement is at the end. The milkman problem could be solved in the following way:

**intHouseNumber = 1  
 Do  
 ' Check the order  
 ' Fetch correct order  
 ' Carry back empties  
 intHouseNumber = intHouseNumber + 1  
 Loop Until intHouseNumber > 20**

### Do...Loop While

A ‘Do...Loop While’ loop is very similar to the previous ‘Do...Loop Until’. The difference is in the logic of the exit condition. For example, the ‘Loop While’ will repeat a block of instructions while a condition *remains* true, but the ‘Loop Until’ loop will exit when the exit condition *turns* true.

### Do While...Loop

This loop is functionally equivalent to the ‘Do...Loop While’ except that the loop test condition is tested at the start of the loop. In practice this means that the ‘Do While...Loop’ might not be executed at all (if condition is false), whereas the ‘Do...Loop While’ will always be executed at least once.

Do While...Loop (Never executes loop):

**Dim a as Byte  
  
 a = 1  
 Do While a > 10  
 a = a - 1  
 Loop**

Do While...Loop (Example which will execute loop):

**Dim a as Byte  
  
 a = 20  
 Do While a > 10  
 a = a - 1  
 Loop**

### Do Until...Loop

This loop, like the previous ‘Do While...Loop’ type, tests the condition at the beginning of the loop. However, whereas the previous loop repeated *while* the condition was true, the ‘Do Until...Loop’ repeats *until* the condition is true. The milkman simulation, discussed above, could be re-coded to use a ‘Do Until...Loop’ as follows:

**intHouseNumber = 1  
 Do Until intHouseNumber > 20  
 ' Check the order  
 ' Fetch correct order  
 ' Carry back empties  
 intHouseNumber = intHouseNumber + 1  
 Loop**

### For...Next

All the loops discussed so far have been **non-deterministic**, that is the number of times the loop will execute is *unknown*. However, there is one type of loop where the number of times it will execute is already known, this is known as a **deterministic** loop. The ‘For...Next’ loop can be used in any situation where the number of times it will repeat is known and fixed. For example, in the milkman situation it is known that there are 20 houses. In other situation it is known that there are so many miles between two towns, 7 days in a week, 10 floors in a particular office block, etc.

**Dim intHouseNumber As Integer  
  
 For intHouseNumber = 1 To 20  
 ' Check the order  
 ' Fetch correct order  
 ' Carry back empties  
 Next**

If this ‘For...Next’ loop is compared with the other non-deterministic loops then one omission becomes apparent - a statement adding one to the house number is missing. This line is no longer needed because this type of loop automatically increments the value of its condition variable.

The ‘For...Next’ loop condition variable does not always have to be incremented by one. Larger increments can be set by using the ‘Step’ parameter. Using this the milkman simulation could be changed so that he walks up one side of the street first (odd numbered houses), stops at number 19, crosses over the street to number 20, and works his way back down the even numbered houses. ‘Print’ statements have been added so that the order of houses visited can be seen.

**Dim intHouseNumber As Integer  
  
 For intHouseNumber = 1 To 19 Step 2  
 ' Check the order  
 ' Fetch correct order  
 ' Carry back empties  
 Print intHouseNumber  
 Next intHouseNumber  
  
 For intHouseNumber = 20 To 2 Step -2  
 ' Check the order  
 ' Fetch correct order  
 ' Carry back empties  
 Print intHouseNumber  
 Next intHouseNumber**

### While...Wend

The ‘While...Wend’ loop is functionally equivalent to the ‘Do While...Loop’ discussed above. Once again this type of repeating structure should be used in situations where the loop will execute at least once. For example, a program which checks ban PIN numbers would always execute at least once:

**Private Sub Form\_Activate()  
 Dim intEntered\_PIN As Integer  
 Const intCorrect\_PIN = 1927  
   
 While intEntered\_PIN <> intCorrect\_PIN  
 intEntered\_PIN = InputBox("Please enter your PIN:", "PIN", "")  
 Wend  
 End Sub**

**Note:** In reality an InputBox would not be used since the typed PIN can be seen on the screen and also the correct PIN would not be held in a constant but instead retrieved from an encrypted file, but as an example it demonstrates the potential of ‘While...Wend’ loops.

### Nesting loops

In the milkman example above the milkman performs the basic 4 steps at each of the 20 houses he serves. However, consider the more complex problem of simulating the pouring of cups of tea at a Church meeting. The problem is more complex because the tea pot holds only 5 cups of tea and there are 100 to serve. To do solve this problem two loops can be used, one inside another (nested):

**Dim cups\_filled as Byte  
  
 bytCupsFilled = 0  
 Do  
 ' boil water  
 ' add tea bags to pot  
 ' fill pot with boiling water  
 For bytPour = 1 to 5  
 ' pour tea into cup  
 ' move to next cup  
 bytCupsFilled = bytCupsFilled + 1  
 Next bytPour  
 Loop Until bytCupsFilled = 100**

**Note:** Different types of loops can be nested inside one another. In the above example the ‘For...Next’ loop is nested within a ‘Do...Loop Until’ structure.

 [Menu](http://docs.google.com/dvbhome.html)