Writing Conditional Code

The ability to write conditional code is a valuable addition to your VBA coding skills. Conditional code is code that can follow a different path depending on conditions you specify. It is sometimes called *branching code*. VBA offers you two ways to do this: the *If Statement* and the *Case Statement*. Conditional code usually requires you to specify one or more actions in response to certain conditions that you define.

If Statements

The *If Statement* is a very flexible technique and its syntax can be expanded to suit your needs. At its simplest, an if statement can be written in a single line:

```
If <CONDITION> Then <ACTION>
```

For example:

Listing 1:

```
Dim Response As VbMsgBoxResult
Response = MsgBox("Do you want to continue?", vbYesNo)
If Response = vbNo Then Exit Sub
' Code continues here...
```

The code (*Listing 1*) displays a Yes/No message box asking the user if they want to continue. Their response is stored in the variable named *Response*. The If statement includes the condition that the value of the variable is *vbNo* (*Response* = *vbNo*) meaning that the user pressed the "No" button on the message box. If the condition is satisfied the action occurs which is to terminate the macro (i.e. *Exit Sub*).

This kind of *If Statement* is normally only be used when the macro (*Exit Sub*) or a loop (*Exit For* or *Exit Do*) is to be terminated.

If you need to execute one or more code statements in response to the condition being satisfied then the syntax must be written on separate lines:

```
If <CONDITION> Then
    <ACTION>
End If
```

For example:

Listing 2:

```
Dim Response As VbMsgBoxResult
Response = MsgBox("Do you want to continue?", vbYesNo)
If Response = vbNo Then
    MsgBox "The macro will be terminated."
    Exit Sub
End If
' Code continues here...
```

In this example (*Listing 2*) there is still only one condition but the action consists of two code statements, one displaying a message and another terminating the macro. In this kind of *If Statement* the macro reads the condition and, if the condition is not satisfied, jumps straight to the *End If* line regardless of how many lines of code there are in between.

So far, the examples have only resulted in an action if the condition is satisfied, but you have the option to specify an alternative action to occur if the condition is not satisfied:

```
If <CONDITION> Then
    <ACTION 1>
Else
    <ACTION 2>
End If
```

To continue the message box theme, the next example (*Listing 3*) displays different messages depending upon whether the user answers Yes or No. This introduces the *Else* clause of the *If Statement*, which is used as kind of "catch all" and determines what is to happen in any condition that has not already been specified. Since there are only two buttons for them to press it is safe to

assume that if their response was not "No" it must have been "Yes", so there is no need to specify a second condition.

Listing 3:

```
Dim Response As VbMsgBoxResult
Response = MsgBox("Do you want to continue?", vbYesNo)
If Response = vbNo Then
    MsgBox "The macro will be terminated."
    Exit Sub
Else
    MsgBox "The macro will continue."
End If
' Code continues here...
```

When it is necessary to specify more than one condition the *ElseIf* clause is used:

```
If <CONDITION 1> Then
    <ACTION 1>
ElseIf <CONDITION 2> Then
    <ACTION 2>
End If
```

You may or may not want to include an *Else* clause to account for any unspecified conditions:

```
If <CONDITION 1> Then
    <ACTION 1>
ElseIf <CONDITION 2> Then
    <ACTION 2>
Else
    <ACTION 3>
End If
```

The following example (*Listing 4*) displays a Yes/No/Cancel message box and applies an action to each of the three possible conditions.

Listing 4:

```
Dim Response As VbMsgBoxResult
Response = MsgBox("Do you want to continue on a new sheet?", vbYesNoCancel)
If Response = vbYes Then
    Worksheets.Add
    MsgBox "The work will continue on a new sheet."
ElseIf Response = vbNo Then
    MsgBox "The work will continue the current sheet."
Else
    MsgBox "The macro was terminated."
End If
' Code continues here...
```

You can add as many *ElseIf* clauses as you need:

Listing 5:

```
Dim Mark As Integer
Dim Grade As String
Mark = InputBox("Enter your mark")
If Mark >= 90 Then
    Grade = "A+"
ElseIf Mark >= 80 Then
    Grade = "A"
ElseIf Mark >= 70 Then
    Grade = "B"
ElseIf Mark >= 60 Then
    Grade = "C"
ElseIf Mark >= 50 Then
    Grade = "D"
ElseIf Mark >= 40 Then
    Grade = "Pass"
Else
    Grade = "Fail"
End If
MsgBox "Your grade was: " & Grade
```

The example above (*Listing 5*) assigns a grade to a given mark. Anything over 90 gets an "A+", 80 to 89 gets a "B" and so on until the mark is below 40 when a "Fail" is given.

When the specified condition evaluates to *True* or *False*, such as when you are looking at the value of a *Boolean* variable, you might see a slightly different syntax being used. When you are asking if the condition is *True*, instead of:

```
If <BOOLEAN> = True Then
```

You can write:

```
If <BOOLEAN> Then
```

Similarly, when you are asking if the condition is False, instead of:

```
If <BOOLEAN> = False Then
```

You can write:

```
If Not <BOOLEAN> Then
```

The following listing (*Listing 6*) conditionally adds a new worksheet to the active workbook, naming it with the name of the current day (e.g. *Monday*) providing a worksheet with that name does not already exist.

Listing 6:

```
Dim sht As Worksheet
Dim SheetFound As Boolean
SheetFound = False
For Each sht In ActiveWorkbook.Worksheets
    If sht.Name = Format(Date, "dddd") Then
        SheetFound = True
        Exit For
    End If
Next sht
If Not SheetFound Then
    Worksheets.Add
    ActiveSheet.Name = Format(Date, "dddd")
End If
```

Case Statements

The *Case Statement* works in a slightly different way from an *If Statement*. Whereas the *If Statement* requires you to specify a complete condition and action each time, the *Case Statement* requires you to first specify what you are going to examine, then supply a list of actions for the various different conditions.

It is suggested that when your code requires several conditions it is better to use a *Case Statement* because it requires the macro to establish the condition only once, and therefore will run faster if there are many possible actions. Whether you use a *Case Statement* or an *If Statement* is often a matter of personal choice. For one or two conditions I usually use an *If Statement* but if there are many I tend to choose a *Case Statement*.

Case Statements split the condition into two parts, an object and a value. You first tell the code to examine the value of an object (such as a variable) then tell it what to do in different conditions (i.e. depending on that value). They have the following structure:

```
Select Case <OBJECT>
    Case <VALUE 1>
        <ACTION 1>
    Case <VALUE 2>
        <ACTION 2>
    Case <VALUE 3>
        <ACTION 3>
    Case <VALUE 4>
        <ACTION 4>
    Case Else
        <ACTION 5>
End Select
```

In the following example (*Listing 7*) a *Case Statement* performs the exact same task as the *If Statement* in the earlier example (*Listing 5*) and returns a grade for a given mark:

Listing 7:

```
Dim Mark As Integer
Dim Grade As String
Mark = InputBox("Enter your mark")
Select Case Mark
    Case Is >= 90
        Grade = "A+"
    Case Is >= 80
        Grade = "A"
    Case Is >= 70
        Grade = "B"
    Case Is >= 60
        Grade = "C"
    Case Is >= 50
        Grade = "D"
    Case Is >= 40
        Grade = "Pass"
    Case Else
        Grade = "Fail"
End Select
MsgBox "Your grade was: " & Grade
```

Note that the syntax varies slightly. If you are specifying an exact condition there is no need for an equals sign, for example:

```
Case "London"

Case 25

Case vbYes
```

If you need so specify that something is more or less than a particular value by means of the >, < and = operators you write *Case Is*, for example:

```
Case Is < Date

Case Is <> vbRed

Case Is >= 100
```

If you are not sure what to write simply use the word *Case* on its own and the Visual Basic Editor will correct it as necessary.