Decisions, Decisions... (Selection)

4.1 Introduction

In Topics 1–3, you learned the basics of VB programming – how to input and output data, and how to use pre-defined functions to manipulate numbers and strings. In this topic, we will reinforce these ideas, and explore how to develop programs that allow choices and alternatives.

4.2 What you should already know

This topic assumes that you have already used and understood the following VB constructs:

- basic input and output of data, and simple calculations
- declaring variables (single, string and integer)
- Mid$ and string concatenation

4.3 Learning Outcomes

After completing this topic, you should be able to:

- use If... Then... Else to branch a program
- use multiple Ifs and nested Ifs
- use Select... Case

4.4 Branching with If... Then... Else

So far, all the programs you have written follow the same list of steps from beginning to end, whatever data you input. This limits the usefulness of the program. Imagine how boring a game program would be if you had to make the same moves every time you ran it!

In this section, you will learn how to make programs that do different things depending on the data that is entered. This means that you can write program with choices for the user, and with different options and branches within them. To do this in Visual BASIC is very easy, as you will see.

Here are some examples of VB statements that use the keywords IF, THEN and ELSE.

If Number < 0 **Then** answer = "That was a negative number!"

If Reply = "No" **Then** MsgBox ("Are you sure?")

If Salary > 5000 **Then** Pay = Salary – Tax **Else** Pay = Salary

If Guess = Correct_Answer **Then** MsgBox ("Well Done!") **Else** MsgBox ("Wrong – try again!")

```
If warning = true Then
  turn_off_heating
  turn_on_warning_lights
End If
```

There are 3 basic patterns:

If *condition* **Then** *action*

If *condition* **Then** *action* **Else** *alternative action*

If *condition* **Then**
action 1
action 2
etc.
End If

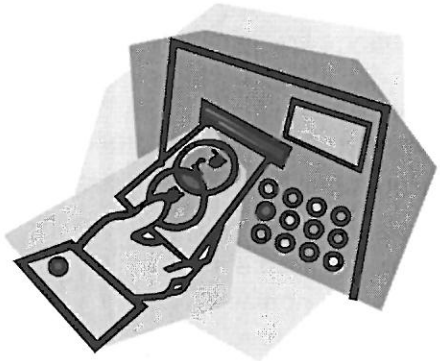
We will use the following comparison operators in this section:

symbol	meaning
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
=	equal to
<>	not equal to

4.5 If... Then... Else – example

Credit Limit

The problem:
When you try to take money out of an ATM (Automatic Teller Machine, commonly called a cash dispenser ot 'hole in the wall'), you are only allowed to withdraw cash up to your credit limit. For example, if your credit limit is £100, and you try to withdraw £50, then it should work fine. However, if you try to withdraw £150, you will not be allowed to, and a message will appear on the screen advising you that this is over your credit limit.

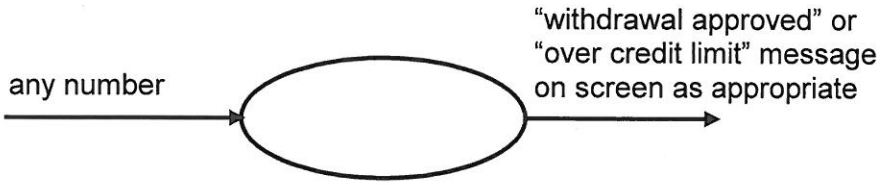


Stage 1 – Analysis: Program specification

Design, write and test a program to:

- take a number entered by the user
- compare it with a credit limit (100)
- report 'withdrawal approved' if the number is up to 100
- report 'over the credit limit' if the number is over 100

Stage 1 – Analysis: Data Flow Diagram



Stage 2 – Design: User interface

We will need a title, then an on-screen prompt for the user's input. We will use a message box to display the feedback message.

Stage 2 – Design: Pseudocode

Here is a list of steps (pseudocode) for what this program has to do:

1. display a title "VB ATM"
2. prompt the user to enter the amount to be withdrawn
3. store the number entered by the user
4. if it is less than or equal to 100, display the approved message, otherwise display the warning message

There is only one variable required – a numeric variable to store the amount entered by the user. As this could be something like 23.50, it needs to be a **Single** rather than an **Integer**.

Stage 3 – Implementation: Coding

Start a new VB Console Application and give it a suitable name

Enter the coding for the program, as below:

```
Sub Main()
    ' coding for the ATM program
    ' warns the user if the amount is over the credit limit (£100)
    ' written by A. Programmer on 09/04/10

    Dim Number as Single

    Console.WriteLine("VB ATM Program")
    Console.WriteLine()

    Console.Write("Enter the amount you want to withdraw: £")
    Number = Console.ReadLine()

    If Number <= 100 Then MsgBox ("withdrawal approved") Else _
    MsgBox (Number & " is over you credit limit")
End Sub
```

If a line of code is too long to fit on the screen, you can split it using an underscore symbol, then pressing return, but make sure there is a space before and after the underscore.

VB will treat the line of code as a single statement, even though it is displayed as two lines.

Stage 4 – Testing

Devise some test data. This should include:

- some **normal** data, like
  20 (clearly under the limit)
  120 (clearly over the limit)
- some **boundary** data, like
  99.99 (just under the limit)
  100.00 (exactly on the limit)
  100.01 (just over the limit)
- some **erroneous** data, like
  -5 (a negative number)
  999999.9999 (a ridiculously large number)
  A (a letter when a number is expected)

Run the program, using your test data, and record the results in a table.

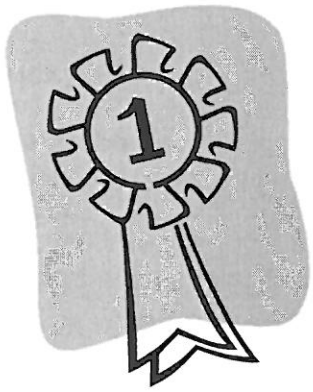
Stage 5 – Evaluation

Write a brief evaluation of the program in terms of its effectiveness, usability and maintainability.

Extra task

Modify the ATM program so that it asks your age, and gives you the message 'You are too young to drive' or 'You can learn to drive' as appropriate.

4.6 Multiple Ifs – example



Lucky Winner

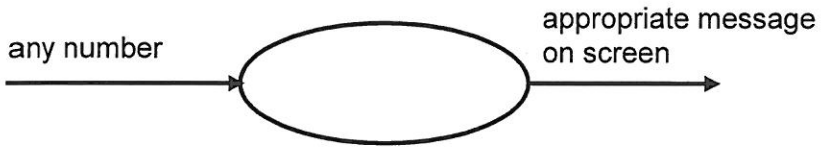
The problem: A program is required that will select a suitable prize, depending on which number between 1 and 5 is entered by the user.

Stage 1 – Analysis: Program specification

Design, write and test a program to:

- prompt the user to enter a number between 1 and 5
- store the number
- output an appropriate message: Enter a 1 -> "You have won a colour TV"
  Enter a 2 -> "You have won a mobile phone"
  etc. (No prize if the number is not between 1 and 5.)

Stage 1 – Analysis: Data Flow Diagram



Stage 2 – Design: User Interface

There will be an on-screen prompt for data entry, and a message box to display the response.

Stage 2 – Design: Pseudocode

Here are the steps for the program.

1. display a title
2. prompt the user to enter a number between 1 and 5
3. store the number entered by the user
4. if the number is 1, display "You have won a colour TV"
5. if the number is 2, display "You have won a mobile phone"
   etc.

There is only one variable required – a numeric variable to store the number entered by the user. As this must be 1, 2, 3, 4 or 5, it should be declared as an Integer.

Stage 3 – Implementation

Create a new VB Console Application, and name it Prize Draw

Enter the coding for the program, as below:

```
Sub Main()
    ' coding for the luck winner program
    ' displays an appropriate message for each possible number entered
    ' written by A. Programmer on 09/04/10

    Dim Number As Integer

    Console.WriteLine("Prize Draw Program")
    Console.WriteLine()

    Console.Write("Enter a whole number in the range 1 to 5:  ")
    Number = Console.ReadLine()

    If Number = 1 Then MsgBox (Number & " wins you a colour TV")
    If Number = 2 Then MsgBox (Number & " wins you a mobile phone")
    If Number = 3 Then MsgBox (Number & " wins you a holiday in Spain")
    If Number = 4 Then MsgBox (Number & " wins you 10p")
    If Number = 5 Then MsgBox (Number & " wins you a day at the beach")

    If Number < 1 Then MsgBox (Number & " is too small")
    If Number > 5 Then MsgBox (Number & " is too large")
End Sub
```

Save the program.

Stage 4 – Testing

Devise some test data. This should include:

- some normal data
- some boundary data
- some erroneous data

Run the program, using your test data, and record the results in a table.

Stage 5 – Evaluation

As usual, you should:

- print out hard copies of your coding
- save your program
- write a brief evaluation of the program in terms of its effectiveness, usability and maintainability

4.7 Conditional statements with multiple actions

Usually, a conditional statement is of the format:

If *condition* **Then** *action*

where multiple actions are required, a conditional statement is of the format:

If *condition* **Then**
Action 1
Action 2
Action 3 (and so on)
End If

Properties such as the console colour can also be changed during the running of the program, as the next example shows.

Lucky Winner v2 (with colours)

As well as the program displaying the prize that has been won, we want the colour of the console to change.

All we need to do is change the coding, so that each **If** statement has 2 actions to be performed – changing the text colour **and** displaying the appropriate message.

This section of code sets the background colour to grey, and the foreground (text) colour to dark blue.

This section changes the text colour to blue if the number chosen by the user is 1. The background colour will remain unchanged.

Similar code will be required for number = 2, 3, 4 or 5.

```
Sub Main()
 ' coding for the prize draw program (version 2)
 ' displays an appropriate message for each number
 ' and changes the background and text colours
 ' written by A. Programmer on 09/04/10

 Dim Number As Integer

 Console.BackgroundColor() = ConsoleColor.Gray
 Console.Clear()
 Console.ForegroundColor() = ConsoleColor.DarkBlue

 Console.WriteLine("Lucky Winner Program")
 Console.WriteLine()

 Console.Write("Enter a whole number in the range 1 to 5: ")
 Number = Console.ReadLine()

 If Number = 1 Then
   Console.ForegroundColor() = ConsoleColor.Blue
   Console.Write(Number & " wins you a colour TV")
   Console.ReadLine()
 End If

 If Number = 2 Then
   ........
   ........
   etc

 If Number < 1 Then MsgBox(Number & " is too small")
 If Number > 5 Then MsgBox(Number & " is too large")
End Sub
```

Don't start coding this program yet!

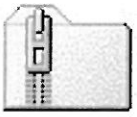
It is possible in VB to use an existing project as the basis for a new project. You will learn how to do this in Topic 4.8, which means you won't need to start from scratch when implementing Lucky Winner (v2).

4.8 Exporting a template

Unfortunately, VB doesn't allow you to simply change a program and save it under a new name. Instead, you need to save the original program as a template, then build the new program from the **template**. Here is how to do it:

- open your Prize Draw project
- choose File ... Export Template
- in the Template wizard, choose Project template, then click Next
- change the template name to Lucky_winner, then Finish

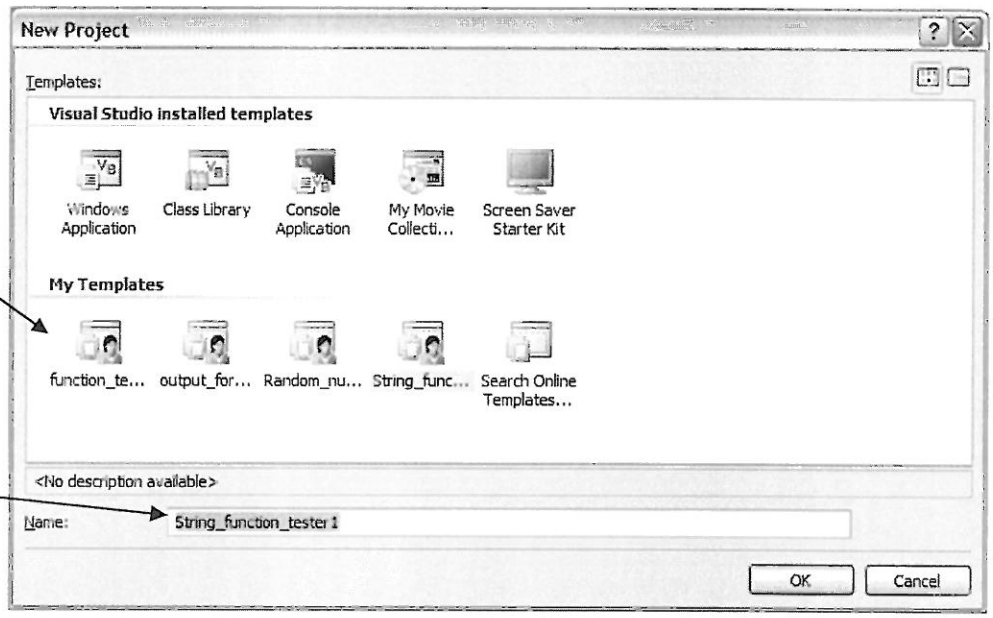
A template has now been created called Lucky_winner. The template contains the coding that you created for the Prize_draw program.



Lucky_Winner.zip
20KB

Now start a new project, but select the Lucky Winner template from the list of templates...

...and call it LuckyWinner2 by changing the default name here:



Use the template technique to create version 2 of the Prize Draw program (as in topic 4.7).

Decide and choose text colours to be displayed for each number chosen by the user.

(note: when you enter ConsoleColor. The development environment should give you a drop down list of available colours to choose from.)

Test to see that it correctly changes the console and text colours.

4.9 Practical task using Multiple Ifs

Use your 'lucky winner' program template to fulfil this specification:

Design, implement and test a program that asks the user to enter a grade (A, B, C, D or F), and displays messages like 'A means you got over 70%', 'B means you got between 60% and 70%', and so on. The text should change colour depending on the grade entered.

4.10 Test Grader Program

Example: Test Mark Grader

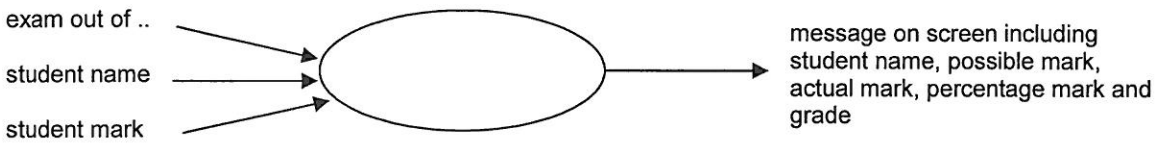
The problem: A program is required that could be used to assign grades to test marks automatically. Over 80% is an A+, over 70% is an A, over 60% is a B, over 50% is a C, over 45% is a D, and less than 45% is a fail. The program should work for any test, whatever the maximum score. It should display a 'certificate' on screen, showing the student's name (in the form 'A. Einstein'), the possible mark, student's mark, percentage and grade.

Stage 1 – Analysis: Program specification

Design, write and test a program to:

- prompt the user to enter the highest possible score for an exam (e.g. 80)
- prompt the user to enter a student's name (first name and surname)
- prompt the user to enter the student's mark (e.g. 63)
- calculate the percentage mark
- display a message giving the student's name, possible and actual marks, percentage and grade

Stage 1 – Analysis: Data Flow Diagram



Stage 2 – Design: User Interface

The user interface will use input boxes for data entry, and display the "certificate" in a cleared console window.

Stage 2 – Design: Pseudocode for the program

Here is a list of steps from the program:

1. prompt for and store the possible score for the exam
2. prompt for and store the student's first name
3. prompt for and store the student's surname
4. prompt for and store the student's mark
5. calculate the percentage mark
6. calculate the grade
7. extract the initial letter from the first name
8. concatenate initial and surname
9. display the name, possible and actual marks, percentage and grade in a cleared window

The program will use several variables. It is useful to write them down as a table.

Complete the table by assigning the most appropriate variable type.

Variable name	Variable type	Used to store
Max_mark		What the test is out of (e.g. 80)
First_name		Student's first name (e.g. Albert)
Surname		Student's surname (e.g. Einstein)
Mark		Student's actual mark (e.g. 63)
Percent		Student's percentage (e.g. 53.7)
Grade		Student's grade (e.g. D)
Init		Student's first initial (e.g. A)
Display_name		Name to be displayed (e.g. A. Einstein)

Convert these into variable declarations, and the pseudocode into VB code. Some of this has been done, to give you a start.

First, the initial comment lines and variable declarations ...

```
Sub Main()
  ' exam certificate program
  ' by A. Programmer 16/04/10

  ' variable declarations
  Dim max_mark As Integer
... etc.
```

Note use of a comment line to indicate the purpose of the code.

Next, get the user inputs using input boxes and store them in variables (pseudocode steps 1 to 4).

```
  ' store user inputs
  max_mark = MsgBox("Enter the maximum mark for this exam")
... etc.
```

Next, code to calculate the percentage mark (pseudocode step 5) ...

```
  ' calculate percentage mark
  percent = (mark / max_mark) * 100
```

Step 6 is the calculation of the grade from the percentage, using a series of If statements (some with complex conditions).

```
' calculate grade
  If percent >= 80 Then grade = "A+"
  If percent >= 70 And percent < 80 Then grade = "A"
  etc.
```

Steps 7 and 8 use **Mid$** and string concatenation to create the display name ...

```
' create display name
  init = Mid$(first_name, 1, 1)
  display_name = init + ". " + surname
```


Finally, step 9 displays the results. Use your previous experience to display the information in cleared window – you may want a title at the top, and to use colours.

```
' display message in format A.Einstein, 42/60, 70%, grade A
.....
.....
```

Stage 3 – Implementation

- Start a new VB Console Application
- Enter the coding (as above, but completed)

Stage 4 – Testing

Devise some test data.

This should include:

- some **normal** data
- some **boundary** data
- some **erroneous** data

Run the program, using your test data, and record the results in a table.

Stage 5 – Evaluation

Print out your program and write an evaluation report in the usual way.

4.11 Branching with CASE

If... Then... Else is not the only way to implement branching in a VB program. There are two alternatives which you should know about. The choice of which method to use is up to you as the programmer. It is partly a matter of style, but one method may have particular advantages, depending on the context.

Your coding for the previous example should have looked something like this:

```
Sub Main()
  ' exam certificate program
  ' by A. Programmer 16/04/10

  ' variable declarations
  Dim max_mark, mark As Integer
  Dim first_name, surname, grade, init, display_name As String
  Dim percent As Single

  ' store user inputs
  max_mark = InputBox("Enter the maximum mark for this exam")
  first_name = InputBox("Enter the candidate's first name")
  surname = InputBox("Enter the candidate's surname")
  mark = InputBox("Enter the candidate's mark")
```



```
    ' calculate percentage mark
    percent = (mark / max_mark) * 100

    ' calculate grade
    If percent >= 80 Then grade = "A+"
    If percent >= 70 And percent < 80 Then grade = "A"
    If percent >= 60 And percent < 70 Then grade = "B"
    If percent >= 50 And percent < 60 Then grade = "C"
    If percent >= 40 And percent < 50 Then grade = "D"
    If percent < 40 Then grade = "fail"

    ' create display name
    init = Mid$(first_name, 1, 1)
    display_name = init + ". " + surname

    ' display message in format A.Einstein, 42/60, 70%, grade A
    Console.BackgroundColor() = ConsoleColor.Gray
    Console.Clear()
    Console.ForegroundColor() = ConsoleColor.DarkBlue
    Console.WriteLine()

    Console.WriteLine("AQA AS Level Computing")

    Console.WriteLine(display_name & ", " & mark & "/" & max_mark & ", " & _
                                        Format(percent, ".0") & "%, " & "grade " & grade)

    Console.ReadLine()

End Sub
```

Note use of underscore to split this long line

Format(percent, ".0") displays the answer as a percentage to 1 decimal place

This method of coding the solution works perfectly well, but it is inefficient. Suppose the value of percent is 85.

The first If statement is 'triggered', so grade is assigned the value 'A+'.
However, the program continues to execute each of the other If statements in turn, although none of them will be triggered. This is wasteful of processor time.

There are two alternative methods, each of which is more efficient: these are

- **Method 1:** using 'nested ifs'
- **Method 2:** using Select... Case

Method 1: Using Nested Ifs (Elseif)

```
If percent >= 80 Then
    grade = "A+"
Elseif percent >= 70 Then
    grade = "A"
Elseif percent >= 60 Then
    grade = "B"
Elseif percent >= 50 Then
    grade = "C"
Elseif percent >= 40 Then
    grade = "D"
Else
    grade = "fail"
End If
```

In this case, if percent is 85, then the first If is triggered as before. None of the Elseif statements will be executed.

Similarly, for other inputs, once a statement is triggered, the following conditions do not need to be tested.

Note also that complex conditions are not required. However, using Elseif is not quite so readable, and it is sometimes more tricky to debug the program, as the logic is more complex.

Method 2: Using Select Case

```
' calculate grade
Select Case percent
    Case Is >= 80
    grade = "A+"
    Case Is >= 70
    grade = "A"
    Case Is >= 60
    grade = "B"
    Case Is >= 50
    grade = "C"
    Case Is >= 40
    Grade = "D"
    Case Else
    grade = "fail"
End Select
```

In this case, if percent is 85, then the first **Case** is triggered and all other Cases are ignored.

Similarly for other inputs, once a statement is triggered, the following conditions do not need to be tested.

Again, note that complex conditions are not required. **Case Else** is used at the end to catch any values of percent below 40.

Select Case is more readable that using nested Ifs, and is also very efficient. Usually Select Case is the best method for multi-way branching in a program.

More efficient Mark Grader programs

- Create a template from Mark_grader_v1 (topic 4.10)
- Use it to create Mark_grader_v2, using Select Case

Some notes on Select Case

The general format of a Select Case in VB is:

Select Case *variable*
Case *value1*
    action to be executed if value 1 matches variable
Case *value2*
    action to be executed if value 2 matches variable
...
Case Else
  action to be executed if variable does not match any of the values
End Select

Where:

- *variable* is an actual variable or some other expression which can be evaluated
- *value1*, *value 2*, etc. are values of the same type as variable (values can be strings or numbers)

Examples:

```
Select Case Code_letter
Case "A"
  MsgBox("You have chosen option A")
Case "B"
  MsgBox("You have chosen option B")
Case  Else
  MsgBox("Sorry, that is not a valid option")
End Select
```

In this example, *variable* is a String variable called Code_letter

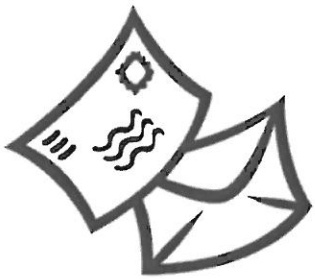
```
Select Case Code_number
Case 1
  MsgBox(Code_number & " is too small")
Case 2 to 19
  MsgBox(Code_number & " is in range")
Case Is > 19
  MsgBox(Code_number & " is too big")
End Select
```

In this example, *variable* is a numeric variable called Code_number – 3 different types of condition are illustrated

```
Select Case Code_letter="A"
Case true
  MsgBox("You have chosen option A")
Case false
  MsgBox("You have chosen option B")
End Select
```

In this example, *variable* is a **Boolean** expression (an expression which is either true or false)

4.12 Practical Task using Select Case



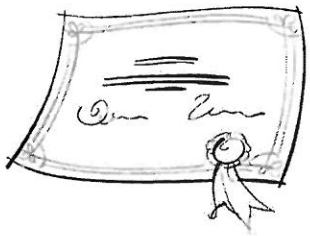
Design, implement and test a program to calculate the cost of posting a package by first class mail, based on the following table:

Weight up to	Cost
60g	28p
100g	42p
150g	60p
200g	75p
250g	88p
300g	£1.01
350g	£1.15
400g	£1.33
450g	£1.50
500g	£1.68
600g	£2.03
700g	£2.38
750g	£2.55
800g	£2.73
900g	£3.10
1000g	£3.45
each extra 250g	add 86p

You should provide evidence of all the stages of development:

- Analysis (program specification and data flow diagram)
- Design (user interface and pseudocode)
- Implementation (coding)
- Testing (results from normal, boundary and erroneous test data)
- Documentation (user and technical guides)
- Evaluation

Topic 4 Summary

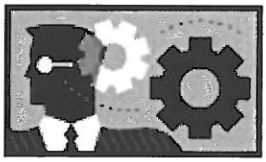


Well done! You have completed Topic 4.

In this topic, you have learned (or revised) how to:

- ☑ Branch using If... Then... Else
- ☑ Use multiple and nested Ifs
- ☑ Use Select... Case

Check all the items on this list. If you are not sure, look back through this section to remind yourself.



When you are sure you understand all of these items, you are ready to continue with the next topic.

