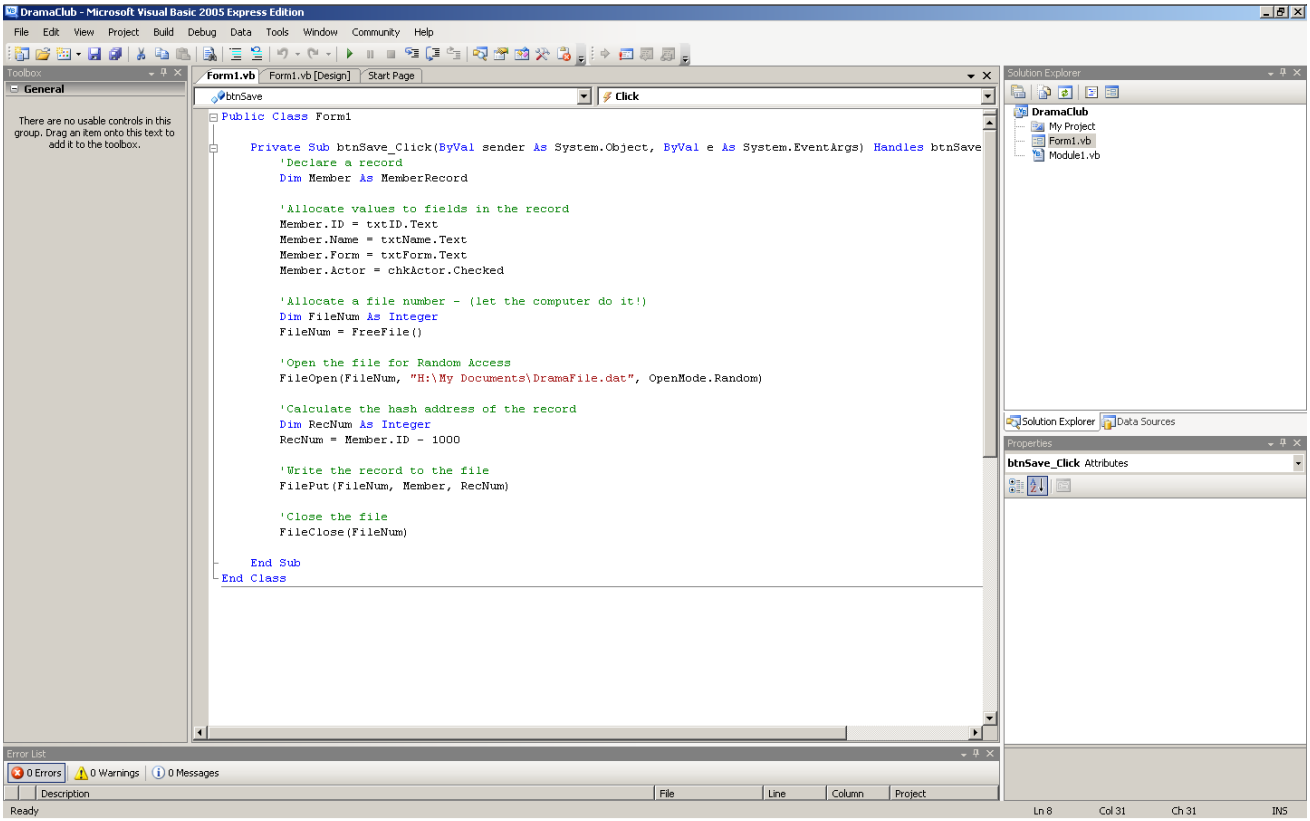
Visual Basic

2005 Express Edition



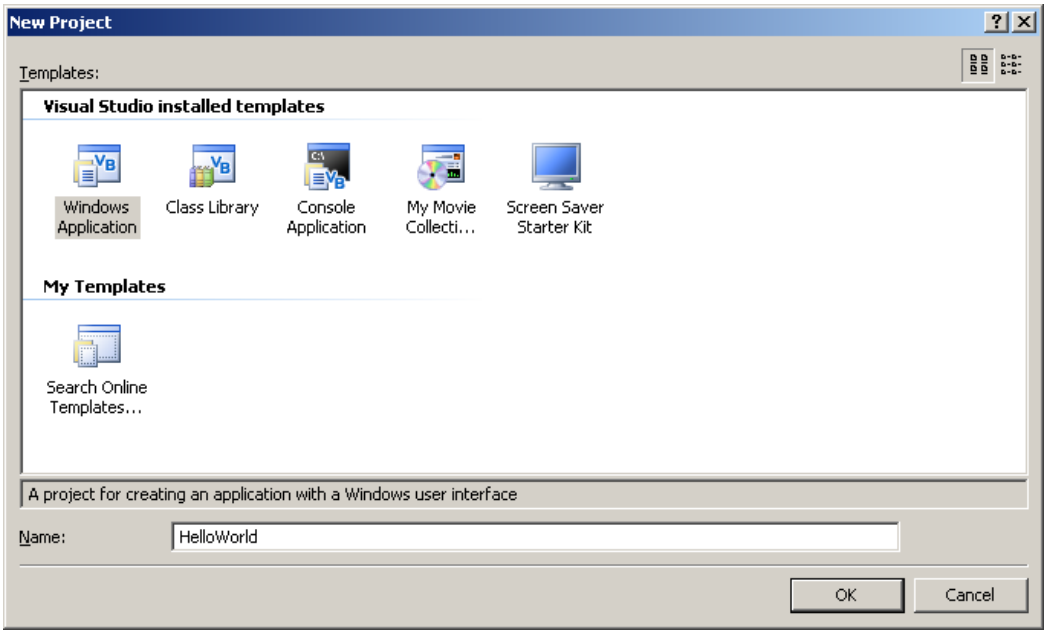
Year 12 : Visual Basic Tutorial.

Our First Program

(Objects and Properties)

HANDS ON

[1] Create a New Visual Basic Project. (Select Windows Application)
Name it **HelloWorld**.



When VB opens, the first **Form** of the project will be displayed in the **Designer** area.

If you click on the Form, the **Properties** of the form are displayed in the Properties box.

Set the following values for the properties of the **Form**:

Property	Value
Name	frmHelloWorld
BackColor	AliceBlue
Text	Hello World
Size	350,180
StartPosition	CentreScreen

It is important to set the properties of the form first.

[1] Form names should always start with **frm**.... Though this is not vital it is important to conform with commonly accepted practices.
In a similar way, names (such as HelloWorld) are usually made up from a number of words where the first letter of each word is in capitals.

[2] Onto the form drag a **Label** from the ToolBox.

…and set its properties as follows….

Property	Value
Name	lblMessage
BackColor	DarkBlue
ForeColor	White
Font Size	24
Text	Hello World

All other properties you can leave as their default values.

[3] SAVE the project.

[4] To run the program, click on the 'Start Debugging' button – 
Or press [f5]
This compiles (VB calls it **building**) and then runs the program.

If you have not made any errors, you should see the program running in a window…

You can stop the program running by closing the window in the normal way, or using the  button

Summary

Windows applications are created by…

- Creating Forms
- Placing Objects on Forms
- Setting the default properties of the objects
- Writing code (see next section)
- Compiling (building) and running the program.

Year 12 : Visual Basic Tutorial.

STUDY THIS

Input and Output
(Text Boxes)

The three stages of a computer process…

- Input
- Processing
- Output

Data is usually input using TextBoxes.

HANDS ON

[1] Create a new Windows Application Project called '**Calculator**'.

Set the Form properties:

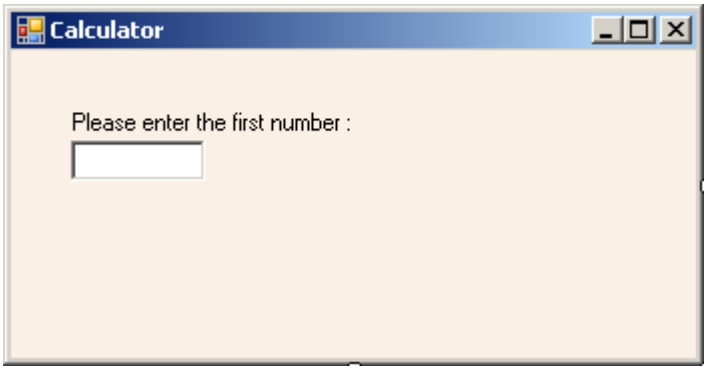
Property	Value
Name	frmCalculator
BackColor	Linen
Text	Calculator
Size	350,180
StartPosition	CentreScreen

[2] Place a TextBox on the form with the following properties:

Property	Value
Name	txtFirstNumber
Location	30,45
Size	67,20

And a Label with the properties:

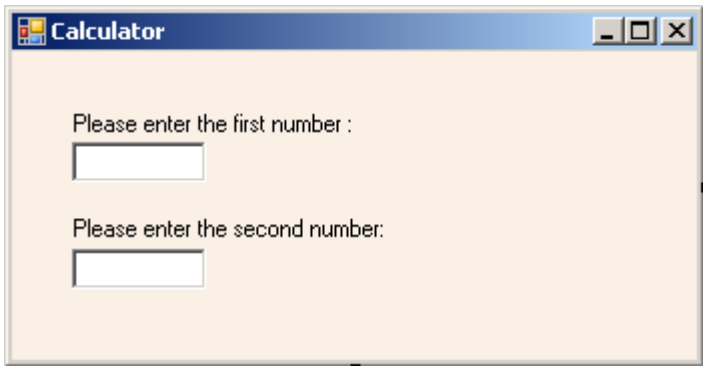
Property	Value
Name	lblFirst
Location	27,29
Text	Please enter the first number:



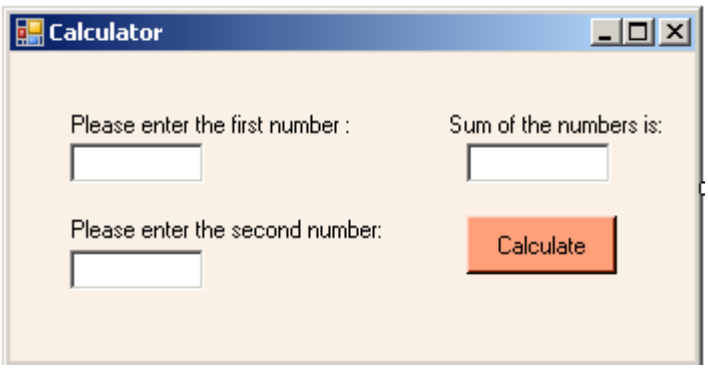
Your form should look like this.

[3] Add another TextBox (txtSecondNumber) and Label (lblSecond) to the form and line them up so the form looks like :

(**NOTICE** the lines on the form that help you line objects up)



[4] Add another TextBox (txtAnswer), a Label (lblAnswer) and a Button (btnAnswer) to your form…



Arrange your objects to look similar to this diagram.

This is the end of the first stage of your program - Creating the interface by adding objects to your form and setting the properties.

[5] Stage 2 – Adding the code for the event handlers….

Only one event-handler to write. The Click event of the Button btnAnswer.

Double-click on btnAnswer to open the Code Window…and type this subroutine….

```
Private Sub btnAnswer_Click

    'Declare the variables
    Dim First As Integer
    Dim Second As Integer

    'Assign values to the variables from the Inputs
    First = txtFirstNumber.Text
    Second = txtSecondNumber.Text

    'Output the answer
    txtAnswer.Text = First + Second

End Sub
```

STUDY THIS

Explanation:

[a] The Green lines are **comments**. These are important! They explain what each section of code does. These are a useful reminders for YOU…and should always be included.

Also useful is 'white space' the blank lines between sections of the code.

[b] **Variables** are quantities that may be different each time a program is run. The computer needs to know what variables you are going to use and what type they are. (see List at the end of this section)

Your subroutine uses two variables called **First** and **Second**. They are both of **integer** type.

[c] First = txtFirstNumber.Text
The input line. This line assigns the value of the Text property of txtFirstNumber to the variable **First**. In other words the value of **First** becomes the number in the Textbox at the time the button is clicked.

Make sure you fully understand how an assignment statement works – you will be using them a lot!

A = B
….assigns the value of B to the variable A. This means that the value of A changes …but the value of B does not.

[d] txtAnswer.Text = First + Second
The output line - txtAnswer will display the result of the calculation.

This is really another assignment statement, where the value of the Text property of txtAnswer is given the value that is the sum of the two numbers.

Phew! – Some important stuff there!

HANDS ON

[6] **Run** the program and check that it works.
(Input two numbers and click the button)

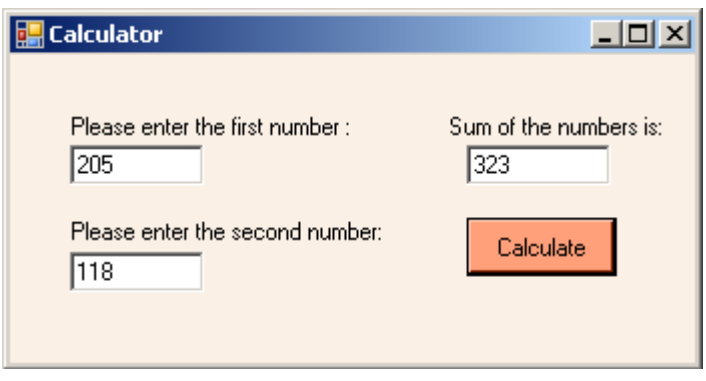


Table of Data Types.

STUDY THIS

Data Type	Comment	Size	Example
Char	Any character	1 byte	'A'
String	Up to about 2 billion characters	2 bytes per character	'Tom Jenkins'
Byte	0 to 255	1 byte	29
SByte	-128 to 127	1 byte	-3
Short	-32,768 to 32,767	2 bytes	3278
UShort	0 to 65,535	2 bytes	49312
Integer	-2,147,483, to 2,147,483,647	4 bytes	629,439
UInteger	0 to 4,294,967,295	4 bytes	3,120,000,000
Long	Massive whole numbers	8 bytes	7,444,555,666,777
ULong	Massive whole numbers	8 bytes	32,456,457,645,999
Single	Real numbers	4 bytes	125.99
Double	Real numbers	8 bytes	3.14159265
Decimal	Real Numbers	16 bytes	36,689.87514
Boolean	True or False	1 bit	True
Date	Jan 1st, 0001 to Dec 31st, 9999	8 bytes	6/3/2012

When a variable is declared in a program, it is really an instruction to the computer to reserve some space in memory, where the value of that variable will be stored. The amount of memory space reserved depends on the type of the variable. (see above table)…so it is good programming practice to declare variables as small types whenever possible.

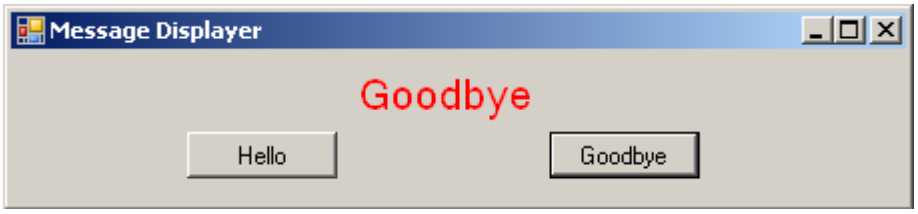
Example - Don't use an Integer when a Byte would do.

When a subroutine has run and finished, the space reserved for local variables declared in that subroutine will be released.

Visual Basic Challenges 1

HANDS ON

[1] Create an application that has a Label and two buttons. When one of the buttons is clicked, the message reads 'Hello' in Green, and when the other button is clicked the message reads 'Goodbye' in Red.



[2] Create an application where a button displays the message "Hello World" in RED when the mouse button is pressed down, and in GREEN when the button is released. (HINT : Use The MouseDown and MouseUp events)

[3] Create an application which looks like this when run…

There is an invisible Label below the button.

When the program is run, the user enters a name into the TextBox and clicks on the button to reveal the message…

HINT : You can add text strings together…
"BULL" + "FROG" is the string "BULLFROG"

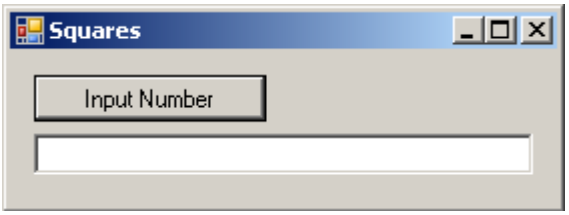
Other Methods of Input and Output
(InputBox; MsgBox)

HANDS ON

Another method of input involves using an **InputBox**….

[1] You are going to write a program that allows the user to input a number, and outputs its square (Eg Input :7 and output:49)

Create an application with a form that has a Button (btnDisplay) and a TextBox (txtMessage). Arrange the objects to look like this…



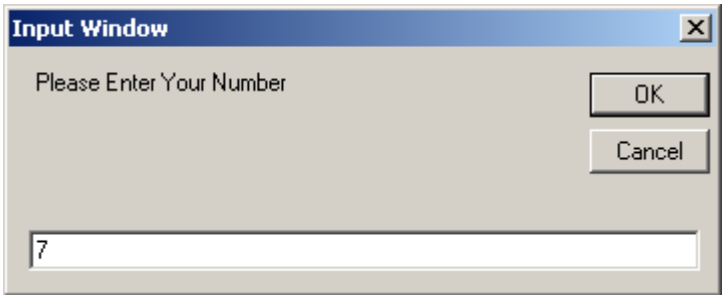
Enter the following event handler for the Click event of btnDisplay…

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        'Declare variables
        Dim Num As Integer
        Dim Answer As String

        'Open the Input Box and assign the value of Num
        Num = InputBox("Please Enter Your Number", "Input Window")

        Answer = "The square of " + Num.ToString + " is " + _
            (Num * Num).ToString

        'Display the message in the TextBox
        txtMessage.Text = Answer

    End Sub
```

The **InputBox** statement in your subroutine has two **parameters** – both are strings. The first string ("Please Enter Your Number") is the message prompt, the second ("Input Window") is the Window title.

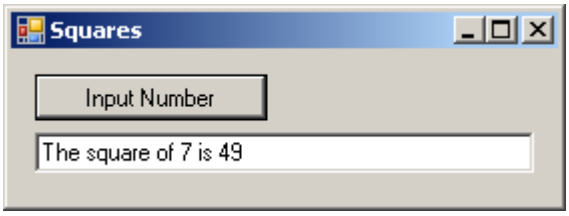
When the program is run, whatever is typed into the InputBox is returned as the value of variable Num… and this variable can then be used in your program.

[2] Run the program….



..and type in your number

When you click the OK button, the message should appear in your form….



NOTE : Strings may be added together, but sometimes you need to turn a number into a string first.

That is why you will see **Num.ToString** in the message

Also…If a line of code is too long, place a <space> and a _ character at the end of the line _ and continue on the next. (like the line above)

Using an MsgBox …. Another way to output data.

HANDS ON

[1] Create a new application and place a Button (btnOutput) on the Form.

Type in the event handler for the Click event of btnOutput…

```
Private Sub btnOutput_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnOutput.Click
        MsgBox("Keep Smiling!")
    End Sub
```

Run the program.

[2] Try changing the subroutine to…

```
Private Sub btnOutput_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnOutput.Click
        MsgBox("Are you still smiling?", MsgBoxStyle.Question,
"Output Demo")
    End Sub
```

Year 12 : Visual Basic Tutorial.

Events

(To give functionality to your program; Button)

HANDS ON

[1] Add a **Button** to your form and set the following properties:

Property	Value
Name	btnDisplayMessage
BackColor	DarkBlue
ForeColor	White
Location	110, 120
Size	120, 25
Text	Display Message

[2] Change the following property of the Label **lblMessage**

Property	Value
Visible	False

This will make the 'Hello World' message invisible when the program first runs.

[3] **Run** the program now. The label should be invisible…but the button will do nothing when you click on it.

The next step is to write the code that causes the label to become visible when the button is clicked….

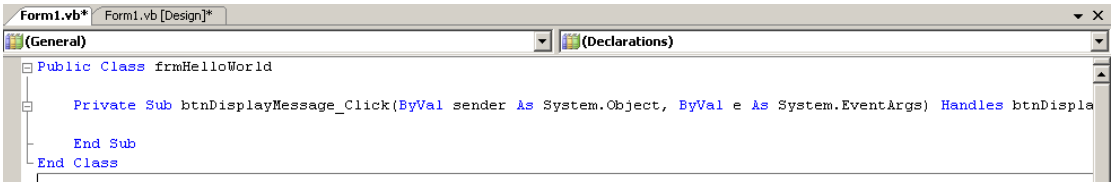
Stop the program running.

An **Event** is an action (such as clicking a mouse) that causes a small program called a **subroutine** to run.

This subroutine is often referred to as an **event handler**.

[4] Double-click on the Button.

The Code Window should open as a tabbed window….



Some program lines have already been added for you. All the subroutines for the **form** are grouped into a **Class**…and you can see the start and end statements for this.

We'll worry about **Public** and **Private** later…
Sub stands for **Subroutine**.

The subroutine is called **btnDisplayMessage_Click**…because it is the event handler for the Click event of the button btnDisplayMessage.

Other items in the Subroutine header do not concern us at the moment…

[5] **Type** in the one line of code so the subroutine looks like…
(to keep things simpler, the subroutine heading is not complete)

```
Private Sub btnDisplayMessage_Click
  lblMessage.Visible = True
End Sub
```

Note how Visual Basic tries to help you as you type the code. This is a really, really, really useful feature and should always be used. If it does not … then you have made a mistake!

[6] **Run** the program and click the button…all should be revealed!

Summary

- **Subroutines** are small programs that can be called (run) at any time.
- **Event-handlers** are subroutines that are run when an event associated with an object occurs.

Year 12 : Visual Basic Tutorial.

More about … Identifiers : Variables and Constants.

STUDY THIS

Computers process data – that's what they do!

Data is input, then it is processed, then the results are output. The data that is processed may be of a number of different types, but every item of data used by a program must be **declared** – ie. The computer must be told beforehand what data is used, what it is called and what type it is.
This is done using a variety of different statements…

Dim

Eg. If an integer variable is going to be used to store an exam mark, we may use…

```
Dim ExamMark As Integer
```

'ExamMark' is the identifier name; and it is of type Integer.

Const

Eg. If a constant is going to be used to store the VAT percentage rate, we may use …

```
Const VATPercentageRate As Single = 17.5
```

HINTS :

- Always use **self-documenting code** - meaningful names for your identifiers. This will be a good habit to adopt, and will help you develop your programs. (Don't be lazy about typing in long identifier names like 'CustomerFirstName'.)
- Always use a constant if possible. This will make it easier to change the values of the data later. In fact only one change should be made – instead of changing the values all the way through the program!

Local and Global Variables

If a variable is declared inside a **subroutine** then it is only allowed to be used inside that subroutine. This is called a **local variable**. Once the subroutine has been run, the space used to store the variable is released by the computer to be used by other processes.

If a variable is declared inside a **class**, it may be used in **any** of the subroutines inside that class. This is called a **global variable**. The computer reserves space and protects it for the whole time the form is opened.
If you want a global variable (or constant) that can be used throughout **all forms** (classes) of a project use the **Public** declaration…

Eg.

```
Public FilePath As String
Public Const Pi As Double = 3.1415927
```

Operators.

The basic operators that can be used are shown in the table below:

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
\	Integer division
Mod	The remainder when numbers are divided
^	Exponent (power)
&	String concatenation (joining)

Examples : (Assuming these declarations and values…)

```
'Variable declarations
Dim Num1 As Single, Num2 As Integer

'Assign values to the variables
Num1 = 13
Num2 = 5
```

Then…

Num1 / Num2	=	2.6
Num1 \ Num2	=	2
Num1 **Mod** Num2	=	3
Num1 ^ Num2	=	371293

String **concatenation** is the correct word for 'adding' two strings together.

Eg. "TOM" **&** " " **&** "JONES" = "TOM JONES"

(**NB** You can use the operator '**+**' to concatenate strings if you prefer…)

Summary

All **data** used in a program is labelled with an **identifier** - a name that makes it easy for us to recognise.

A **variable** is an identifier that may change each time we run a program.

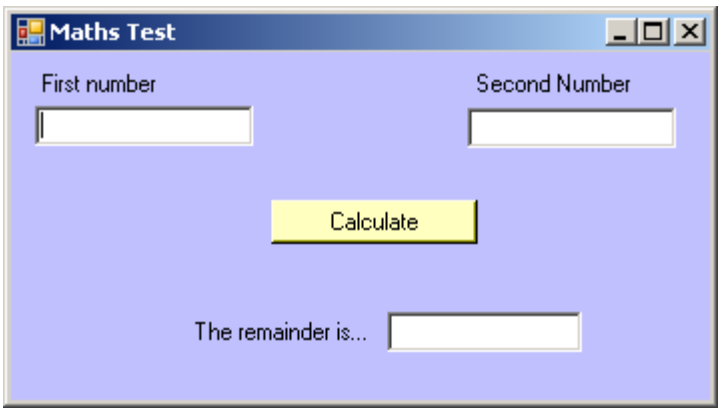
A **constant** is an identifier that is the same every time the program is run.

Visual Basic Challenges 2

HANDS ON

[1] Create a **Factor Test** program that displays the remainder when one number is divided by another number.

The interface should look like the form below…

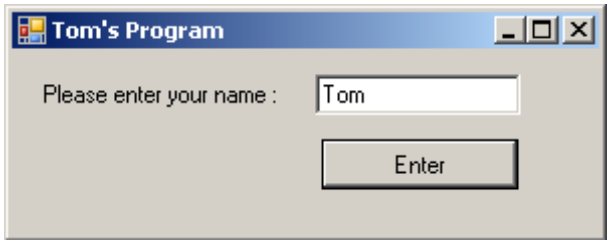


Test Data : 24 divided by 5 has a remainder of 4.
30 divided by 6 has a remainder of 0 (6 is a factor of 30)

Use your program to find the factors of 189
(HINT A factor will give a remainder of 0)

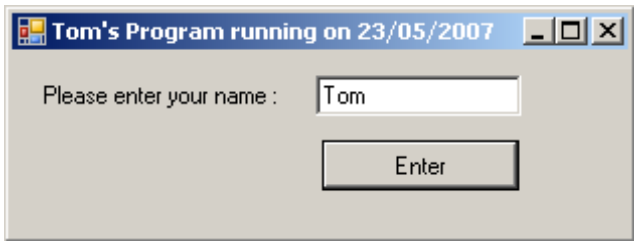
[2] (a) Create an application that allows the user to input their name (Eg. **Tom**), and when an 'Enter' button is clicked, the name of the form at the top changes to '**Tom's Program**'

HINT : When coding the program, the Form is referred to as Me.



RESEARCH NEEDED

(b) Now try adding the current **Date** as well…



Year 12 : Visual Basic Tutorial.

STUDY THIS

Conditional Statements

If [Condition is true] **Then** [Statement]

The **Statement** will only be executed if the **Condition** is true.

The **Condition** must be an expression that is either **True** or **False**.
Eg.

```
If (txtMark.Text > 20) Then txtGrade.Text = "Winner!"
```

Sometimes, usually if more than one statement is to be executed, this may be written as a block…

Eg.

```
If (txtMark.Text > 20) Then
       txtGrade.Text = "Winner!"
       txtGrade.ForeColor = Color.Red
End If
```

Comparisons that can be used in the Conditions :

Comparison	Meaning
=	Equal to
<>	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

A more complex version of the Conditional statement…

If [Condition is true] **Then** [Statement1] **Else** [Statement2]

If the **Condition** is true then **Statement1** will be executed…if not, then **Statement2** will be executed.

Eg.

```
If (txtMark.Text > 50) Then
            txtGrade.Text = "Winner!"
            txtGrade.ForeColor = Color.Red
        Else
            txtGrade.Text = "Loser!"
            txtGrade.ForeColor = Color.Black
            MsgBox("Try again!")
End If
```

Testing several conditions….

Several conditional expressions may be evaluated using If..Then…ElseIf…Else..End If . The syntax for this is shown in the box below…

```
If (conditon1) Then
        Statements executed if conditon1 is true
ElseIf (condition2) Then
        Statements executed if conditon2 is true
ElseIf (condition3) Then
        Statements executed if condition3 is true
Else
        Statements executed if none of the conditions is true
End If
```

Example :

A shop offers a 10% discount if a customer buys more than £100 worth of goods, 5% discount if a customer buys more than £50 worth and no discount otherwise.

The code for this may look something like…

```
If (TotalAmount > 100) Then
            Discount = 10
   ElseIf (TotalAmount > 50) Then
            Discount = 5
   Else
            Discount = 0
End If
```

Select Case

Another method of selection is provided by the Select Case structure. Here is an example…

```
Select Case ExamGrade
       Case "A"
           Label1.Text = "Excellent"
       Case "B"
           Label1.Text = "Brave attempt"
       Case "C"
           Label1.Text = "Average"
       Case Else
           Label1.Text = "Room for improvement"
End Select
```

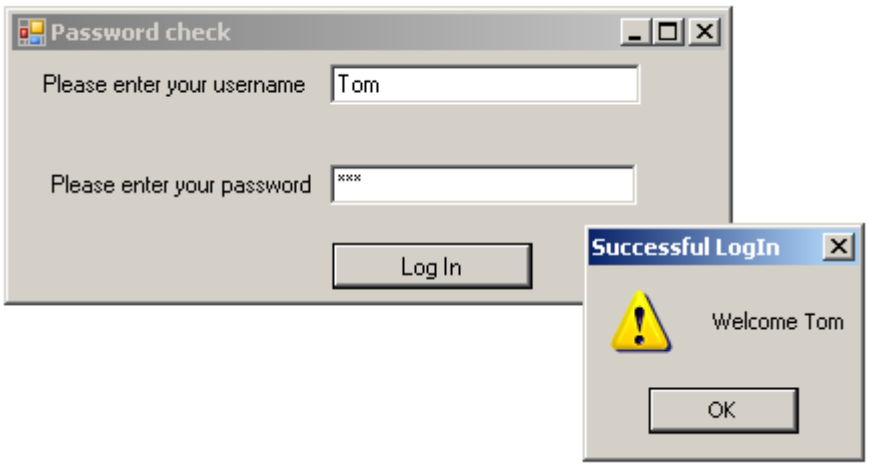
This is a better method when the action depends on the **value** of a **variable**. In the example above, the Text property of Label1 depends on the value of the variable **ExamGrade**.

HANDS ON

Visual Basic Challenges 3

[1] When running a program, a user has to enter their name and a password. Write a program that outputs the message "Welcome" when the correct password is entered. You may choose the password yourself, but it should be hidden when it is being typed in (Check the properties of a TextBox carefully!)

Enhance the program so that the message is personalised. For example if the username is 'Tom' and the password is incorrect, the message "Welcome Tom" should be output.



[2] Computing exam marks are graded as follows:

'A' if the mark is 80% or more,
'B' if the mark is between 70 and 79,
'C' if the mark is between 50 and 69,
'D' for marks less than 50.

Write a program that allows a user to enter an exam mark and display the appropriate grade.

[3] Write an application that allows the user to input a number between 1 and 30 and outputs it as a date in September.

Eg. Input 2 and the output should be 'September 2nd'
Input 23 and the output should be 'September 23rd'

STUDY THIS

Logical Operators

The logical operators **AND**, **OR**, **XOR** and **NOT** can be used in conditional statements.

Logical operator	Meaning
AND	If both conditions are TRUE, the result is TRUE
OR	If either of the conditions is TRUE, or both, then the result is TRUE
XOR	If only one of the conditions is true (not both) then the result is TRUE
NOT	If the condition is TRUE, the result is FALSE. If the condition is FALSE, the result is TRUE

Example : Tom's password is 'Hedgehog'. He must enter his name and his password to gain access to a program…

```
If (txtName.Text = "Tom") And (txtPassword.Text = "Hedgehog")
Then
    MsgBox("Successful LogIn")
End If
```

Example : A message "Welcome" is displayed but not if it is Saturday or Sunday.

```
If Not ((Now.DayOfWeek.ToString = "Saturday") Or
(Now.DayOfWeek.ToString = "Sunday")) Then
            MsgBox("Welcome")
End If
```

HINT : It is good practice to put each condition in brackets to avoid confusion!

Visual Basic Challenges 3 (continued)

HANDS ON

[4] A customer can buy a carpet online by entering the Length, Width and Type of carpet required. The types of carpet are summarised in this table…

Type	Cost per sq. metre	Discount
A	£12.49	10%
B	£10.99	5%

Write a carpet cost calculator program and test it with the following data…

Test Data :

(1) Type A, 4.5 metres by 9.5 metres – Total cost = £480.55

(2) Type B, 3 metres by 8 metres – Total Cost = £250.57

Year 12 : Visual Basic Tutorial.

STUDY THIS

Loops.

A **Loop** is a section of code that needs to be repeated a number of times. The posh term for this repetition is **ITERATION**.

There are two situations…

A. You know how many times to repeat the loop
   (Use a **For…Next** loop)
B. The loop is repeated until a certain condition is met
   (Use a **Do While** or **Do Until** loop)

A : For…Next Loops

An integer variable is needed to count the number of times the loop is run.

The syntax is…

```
For variable = start value to end value
        statements to be repeated
Next [variable]
```

HANDS ON

[1] Create a new Windows Application project.

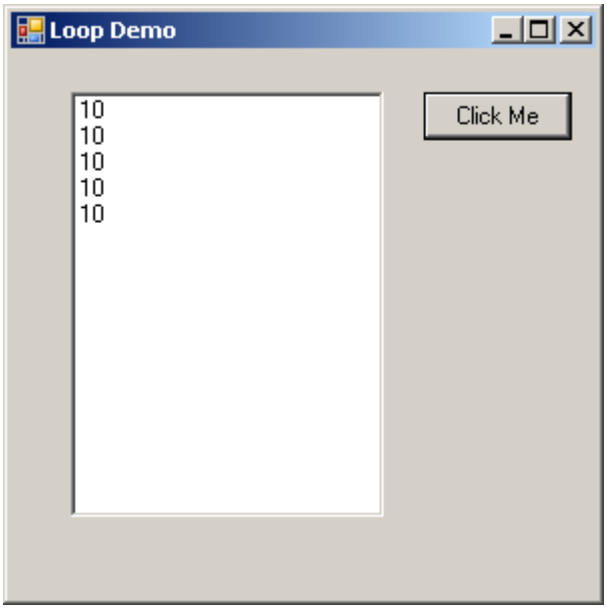
Place on the Form a Listbox and a Button.
(Leave them called ListBox1 and Button1)

Add the event handler for the Click event of Button1:

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click

        Const Num As Integer = 10

        Dim i As Integer

        For i = 1 To 5
            ListBox1.Items.Add(Num)
        Next

End Sub
```

The variable **i** is called the control variable for the loop – it MUST be an integer variable and, basically it counts from 1 to 5.
The loop adds the number 10 to the ListBox 5 times.

Run the program and click the button to see this.

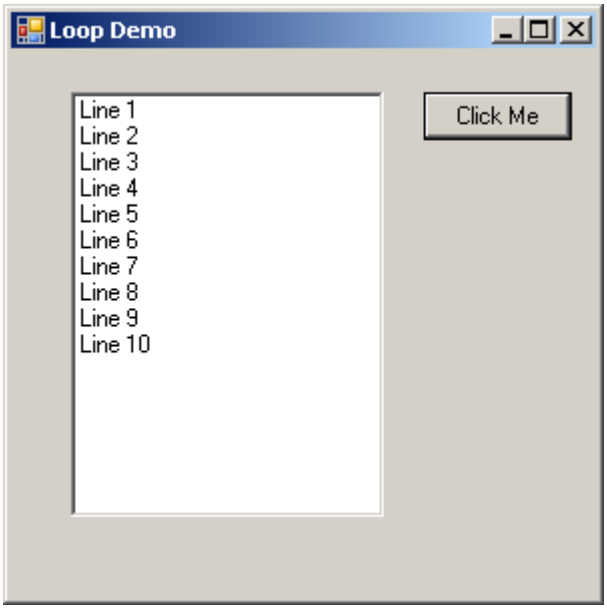


[2] Change the program so the name 'Tom' appears 3 times in the ListBox.

[3] You can also use the value of the control variable inside the loop…

…See if you can output all the numbers from 1 to 10 inside the ListBox.

…and you should even be able to output this…



[4] For even more complex loops, try using the **Step** instruction…

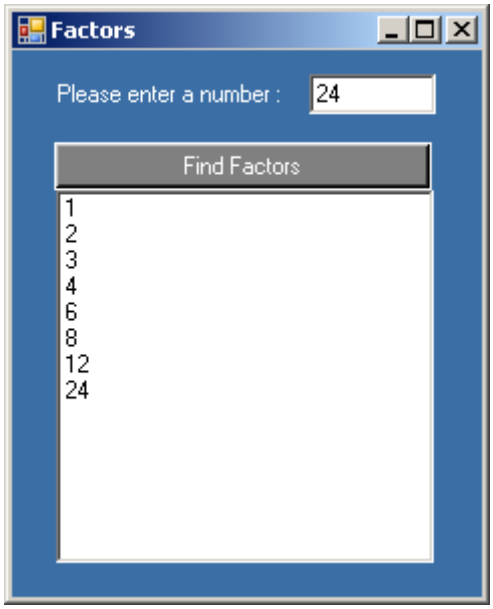
```
Dim i As Integer

For i = 0 To 50 Step 5
    ListBox1.Items.Add("Line " & i)
Next
```

HANDS ON

Visual Basic Challenges 4

[1] Create a new Windows Application project called 'Factors'.

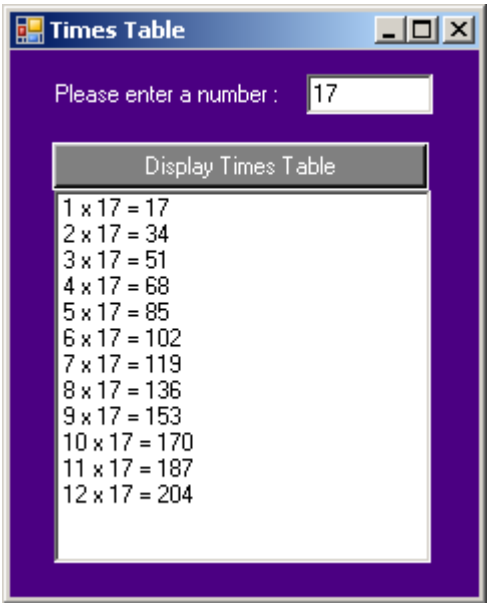
Write a program that allows the user to enter an integer, and find all the factors of that integer. You need to do this by checking every number between 1 and the input number to see if there is a remainder when they are divided.



HINTS :

- Remember to clear the ListBox of items.
- To find whether **R** is a factor of a number **N**, you need to check there is no remainder when N is divided by R.
  i.e. if **N Mod R = 0** then R is a factor of N.

[2] Write an application that allows the user to input a number, and the times table (up to 12) is displayed.



STUDY THIS

Do Loops

(Loops that are repeated until a condition is TRUE)

The syntax is:

```
Do While [condition]
        statements to be repeated
Loop
```

or…

```
Do
        statements to be repeated
Loop Until [condition]
```

In the first case, the condition is checked BEFORE the loop (so the loop may never be executed)…
In the second case, the condition is checked AFTER the loop (so the loop will be executed at least once).

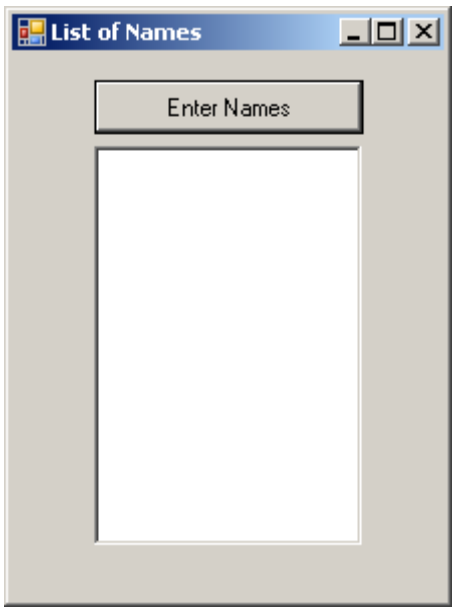
HANDS ON

Example.

You are going to write a program that allows the user to input a list of names, adding each one to a list, until the name 'XXX' is input.

This is an important example of a **ROGUE VALUE** – a data value that tells the computer that a sequence of data input has finished. The rogue value must be a value that would not normally occur.

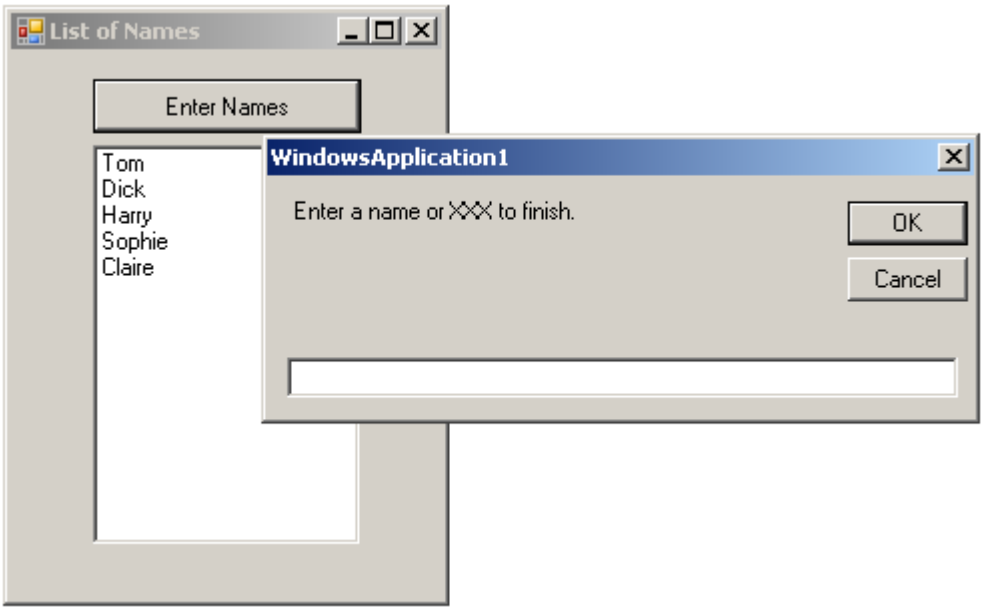
[1] Create a new application, and place a ListBox(lstNames) and a Button(btnNames) as shown…



[2] Enter the Event handler for the Click event of the button btnNames as follows…

```
    Private Sub btnNames_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnNames.Click

        Dim Name As String

        'Make sure the List is empty
        lstNames.Items.Clear()

        'Enter the names
        Do While Name <> "XXX"
            Name = InputBox("Enter a name or XXX to finish.")
            If Name <> "XXX" Then lstNames.Items.Add(Name)
        Loop
    End Sub
```

Note that as many names can be input as necessary until the rogue value of "XXX" is entered.



[3] Run the program and add names. Use the rogue value to end the program.

Summary

A **Loop** is a section of program code that is repeated a number of times.

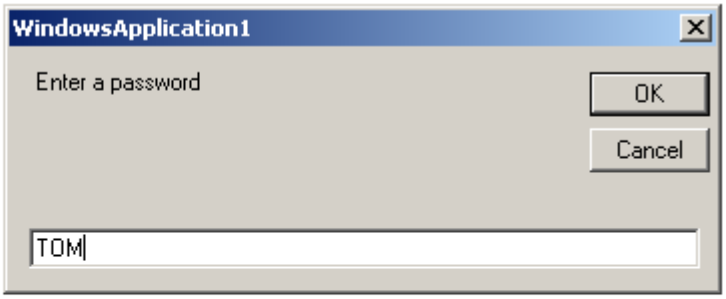
If the number of iterations is known, use a **FOR…NEXT** loop.
If the loop is to be repeated until a condition is TRUE, use a **DO…WHILE** or **DO…UNTIL** loop.

A **Rogue Value** is an item of data that is used to indicate the end of a sequence of data.

HANDS ON

Visual Basic Challenges 4 (cont'd)

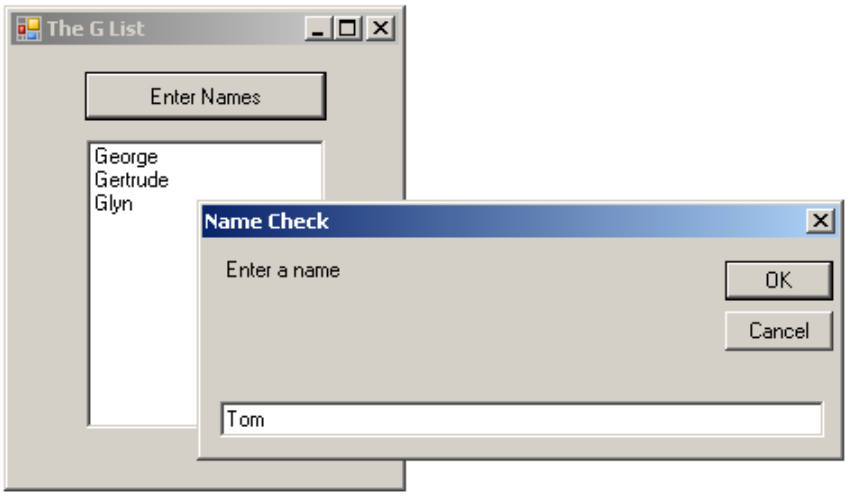
[3] Write a program that asks the user to enter a password.



The user can try entering as many passwords as they like, but only when the password "FRED" is entered a message is displayed saying "WELCOME".

RESEARCH NEEDED

[4] Write a program that allows the user to enter a sequence of names. Only those names beginning with the letter 'G' are added to a list. Use a suitable rogue value to end the program.



Year 12 : Visual Basic Tutorial.

STUDY THIS

Counts, Totals and Averages.

An **algorithm** is a sequence of steps needed to complete a task.

One way of writing down an algorithm is by using **pseudo-code**. It's like a computer program but written in English. It must, however, display the **structure** of the program.

Example :

A sequence of exam marks is input, terminated by a rogue value. The number of exam marks greater than 50 needs to be output.

This is an example of a Counting program. To count, we need to declare an integer variable to do the counting, but we must make sure it is initialised to 0.

The pseudo-code algorithm for this is as follows :

```
counter = 0
repeat
        input(mark)
        if mark > 50 then
                increment counter
        end if
until the end of the data
output(counter)
```

HANDS ON

Visual Basic Challenges 5

[1] Write a program for the example above, which inputs a sequence of exam marks, terminated by a suitable rogue value. The program counts the number of exam marks greater than 50.

Try inputting these marks :
65, 32, 41, 75, 88, 90. 27

The output should be "There are 4 marks greater than 50"

> **HINT** :
>
> To increment a variable means to add 1 to its value.
>
> To add 1 to a variable called 'Counter' use this code…
>
> ```
> Counter = Counter + 1
> ```
>
> (…this means that 'the new value of Counter is the old value plus 1.)

STUDY THIS

Example :

A sequence of prices is input. The total amount of the bill is to be output.

To code this, you need a variable for the Total.

The pseudo-code algorithm for this would be :

```
set the Total to 0
repeat
        input(Price)
        Add Price to Total
until the end of the data
output(Total)
```

HANDS ON

Visual Basic Challenges 5 (cont'd)

[2] Write a program for the example above, that calculates a total bill for any number of input prices.

Test data :

[a] Prices £10.20, £3.50, £2.10
Total is £15.80

[b] Prices £0.20, £0.75, £1.90, £2.30
Total is £5.15

HINT :

To add the value of one variable ('Fred') to the value of another variable ('Jim')....

```
Jim = Jim + Fred
```

CAREFUL : It is important to understand that it is the value of 'Jim' that is changing here. 'Fred' remains unchanged.

[3] A pupil wants to input all his exam marks and output the **average** exam mark. Can you write a suitable program for this?

(You will need to have a Count and a Total.)

Test Data :

Exam Marks : 78, 52, 80, 63, 49, 71.
Average is 65.5

STUDY THIS

Debugging.

A problem in a computer program is called a **bug**. The process of getting rid of bugs is called **debugging**….so what do you do if your program does not work?

There are three different types of error that may occur…

[1] **Syntax Error**
This is when the programmer (you!) breaks the rules of the syntax of the language. For example, you may spell an instruction or property incorrectly…

Example : txtMessage.Txet instead of txtMessage.Text

Syntax errors are usually picked up by the compiler before the program is run.

[2] **Logical Error**
The program runs fine…but gives the wrong results.

Example : The program may add a discount amount instead of subtracting it.

[3] **Run-time Error**
The program compiles fine, but an error occurs when the program is run.

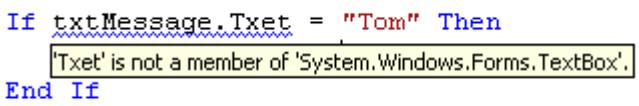
Example : The program may try to open a file of data that is not in the expected place.

Another example : trying to divide a number by 0 may cause a run-time error.

Finding Errors

To fix your errors, you first have to find them.

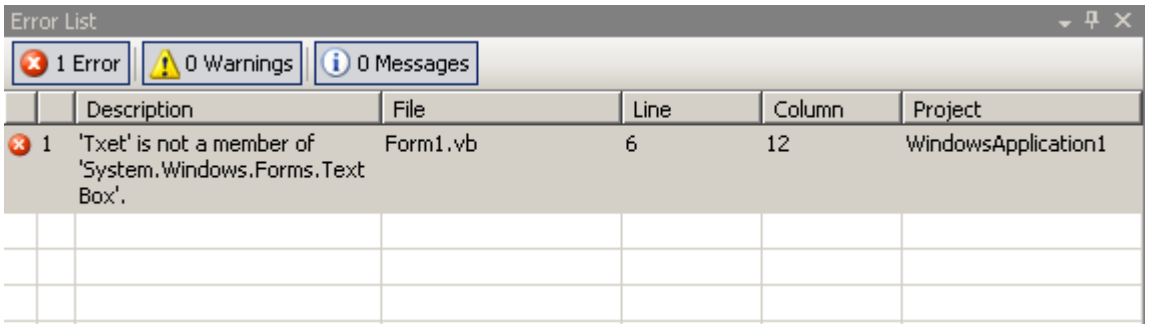
Syntax errors in VB are usually shown by a blue squiggly line. Hover your mouse cursor over the error and a helpful diagnostic error message should be displayed….



For beginners, some of these error messages take some getting used to! – but they should at least give you a clue about what the error is.

Logical errors can be much harder to track down…

There will also be an entry in the 'Error List' box if it is displayed at the bottom of the screen.

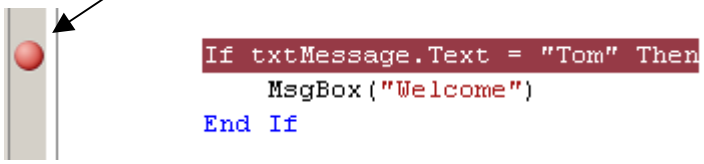


If you double-click on the error, it will take you to where it is. This is useful in a long program.

Breakpoints

If a program is not working it is possible to stop the program running at a specified line of code. To do this you need to insert a **Breakpoint**.

Click on the grey border on the left edge of the line where you want the program to stop running. A Red marker will appear and the line of code will be highlighted in red. (To remove the breakpoint – click it again)



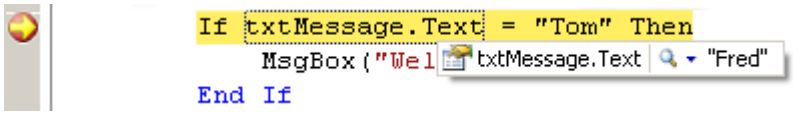
Run the program and execution will stop at this line. The line is highlighted in yellow and an arrow placed in the margin to show the line at which the program stopped.

You can now do one of two things…

[1] Check the values of variables or object properties.
[2] Single-step through the program, running one line at a time.

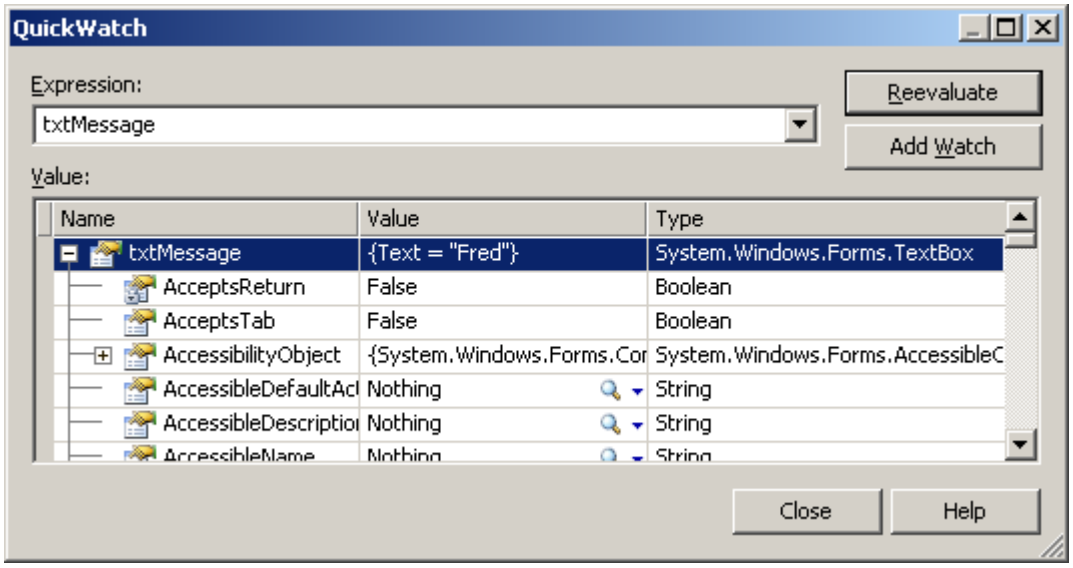
Checking Values of Variables or Object properties

When the program execution stops at a breakpoint, you can place the mouse cursor over a variable, and the value will be displayed…



Is it the value you expected it to be? If not, it may give you a clue as to what the problem is…

You could also add a Watch. (Use the Debug window). This would list the values of all the properties of an object…
Click on the 'Add Watch' button to display this in the Watches window at the bottom of the screen.



You can also add expressions (such as txtMessage.Text = "Tom") to the Watch window…to see if they are TRUE or FALSE.

You can add as many watches as you need.

Single-stepping

Use the Step Into button  to execute the next program statement. (The line highlighted in yellow is the **NEXT** line to be executed.)

Keep an eye on the values of your watches as each line is executed and it should give you a clue about what the problem is.

HINTS : If your program is not working…

1. Place a breakpoint at the start of the section.
2. Add Watches to look at values of object properties.
3. Single-step through the program – keeping an eye on your watches.

Year 12 : Visual Basic Tutorial.

STUDY THIS

Simple Error Trapping.

Programs should never crash!

The best way to deal with run-time errors is to trap them with an error handler. This is a sections of code that handle these errors when they occur.

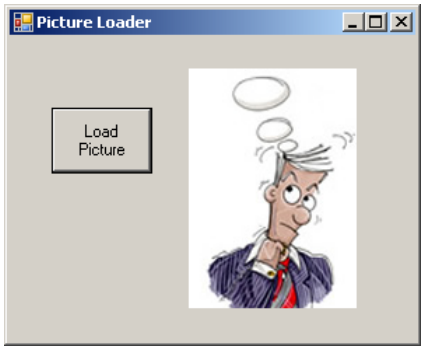
Run-time errors are referred to as Exceptions.

An error handler uses the Try…Catch…Finally code block.

This is what you do…

HANDS ON

[1] Create a new Windows Application.
On the form place a PictureBox (picPhoto) and set the Image property to an existing graphic and a Button (btnChange).



The program is going to change the picture when the button is pressed.

[2] Add this event handler to the Click event of the button.
(You will need to put the full path of a valid picture into the red text)

```
Private Sub btnChange_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnChange.Click
    picPhoto.Image = System.Drawing.Bitmap.FromFile("R:\homer.gif")
End Sub
```

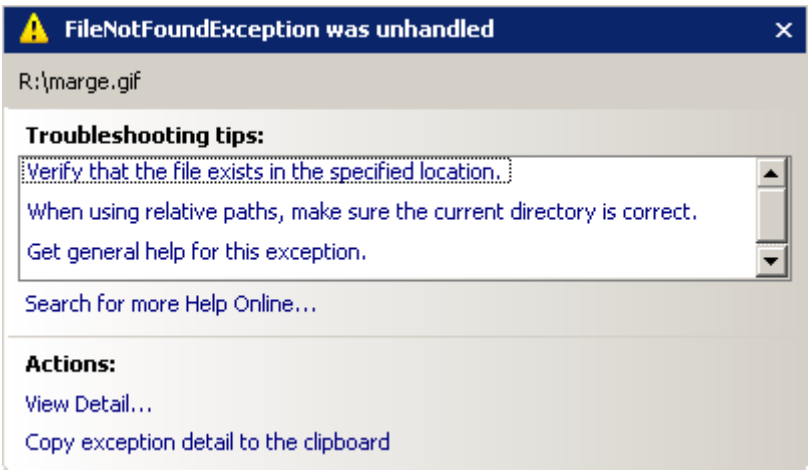
Run the program and see if the picture changes…(fix it if it doesn't!)

[3] Now, suppose the path of the picture is not correct…

Change the path to a picture file that does **NOT** exist…

```
Private Sub btnChange_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnChange.Click
    picPhoto.Image = System.Drawing.Bitmap.FromFile("R:\marge.gif")
End Sub
```

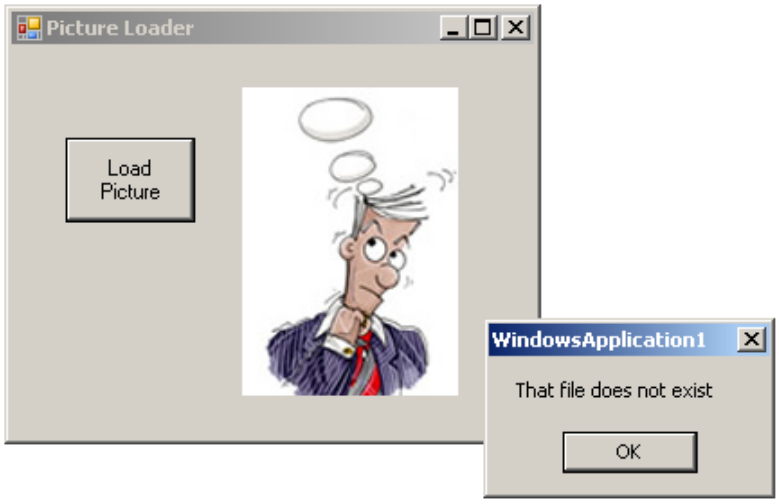
…and run the program. This time you will get a run-time error, and a box like this will point to the line where the error was found:



[4] Change the code of the event handler to this…

```
Private Sub btnChange_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnChange.Click
  Try
    picPhoto.Image = System.Drawing.Bitmap.FromFile("R:\marge.gif")
  Catch ex As Exception
    MsgBox("That file does not exist")
  End Try
End Sub
```

Running the program now, an error message should pop up…



Year 12 : Visual Basic Tutorial.

STUDY THIS

String Handling.

Strings can be added together. This is called string concatenation.

Example :

```
Dim Surname, Forename As String

Surname = "Thomas"
Forename = "Danny"
MsgBox("Hello " & Forename & " " & Surname)
```

This message box would output "Hello Danny Thomas"

Strings can be compared alphabetically so …

"A" < "B" is TRUE
"Martin" < "Paul" is TRUE
"Apple" > "Ball" is FALSE

The table below shows a list of string methods with explanations and examples:

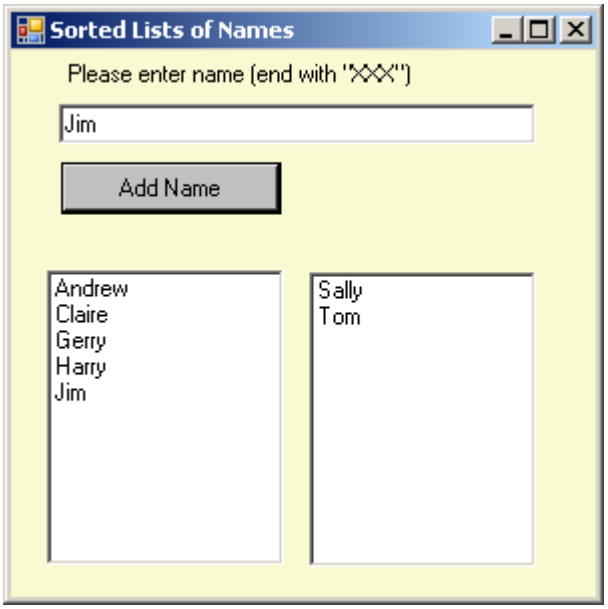
Method	Explanation	Example
Length	The number of characters in a string.	If Name = "John" then Length(Name) is 4
ToUpper	Changes letters to upper case	If Name = "John" then Name.ToUpper is "JOHN"
ToLower	Changes letters to lower case	If Name = "John" then Name.ToLower is "john"
Substring	Returns a string from inside another. The starting point and number of characters is given.	If Name = "Thomas Jones" then Name.Substring(3,2) is "ma"
Trim	Removes spaces from start and end of a string.	If Name = " Tom Jones " then Name.Trim is "Tom Jones"
IndexOf	Returns the starting position of one string inside another.	If Name = "Tom@Jones" then Name.IndexOf("@") is 3
Insert	Adds a string into the middle of another. The start point and the string must be given.	If Name = "Tom Jones" then Name.Insert(4,"Bart") is "Tom Bart Jones"
Remove	Deletes characters from a string. The start point and the number of characters to be deleted is given.	If Name = "Tom Bart Jones" then Name.Remove(4,5) is "Tom Jones"

HANDS ON

Visual Basic Challenges 6

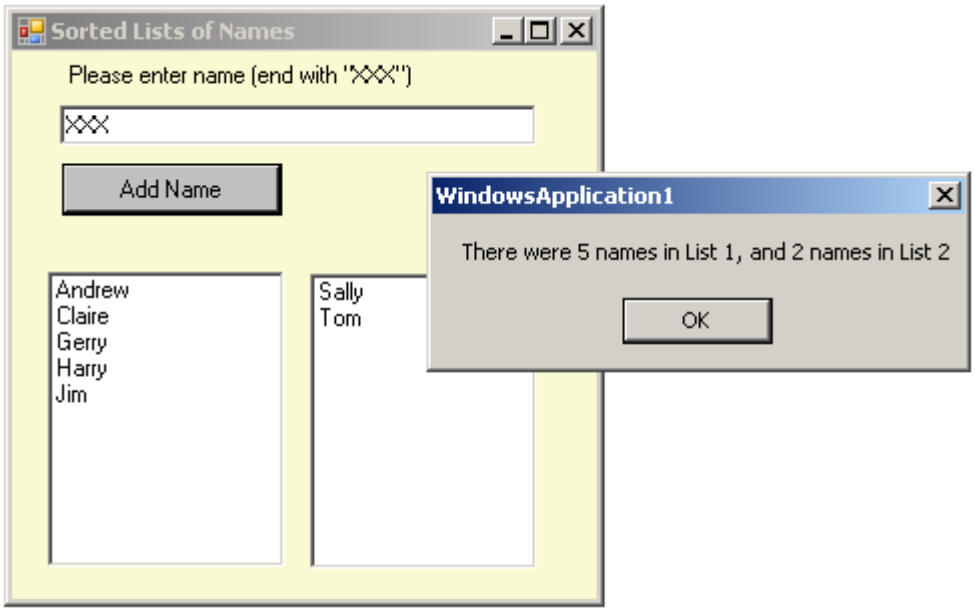
[1] Create an application that allows the user to enter a sequence of names ending with a rogue value of "XXX".

The program should sort the names into two lists. One with all the names that start with letters "A" to "L", and the other list with the remaining names.



Can you keep the lists sorted in alphabetical order even though the names are not entered that way?

Make sure your program exits properly when "XXX" is entered, and outputs a message saying how many names are in each list.



[2] The email address of employees in a company called BizzyBee Ltd is made from the first two letters of their first name, their surname and the department they work in.

For example, Mary Smith in the Accounts department has an email address:
ma.smith@accounts.bizzybee.com

James Davies in the Sales department has an email address:
ja.davies@sales.bizzybee.com

Write a program that allows an employee to enter their full name and the department they work for (Sales, Accounts, or Maintenance) and outputs their email address.

[3] An online music and computer games company, codes each item it sells with a unique 6-character code. The first two characters must be either **CD** (for Cds) or **DV** (for DVDs). The remaining 4 characters must be numerical digits.

Valid codes : CD56321 DV6700 CD0018

Invalid codes : CD431 DW7891 DV567G

Write a program that inputs a code and fully validates it. If an incorrect code is found then the user must enter another code. When a valid code is entered, the program ends.

Year 12 : Visual Basic Tutorial.

STUDY THIS

Arrays.

An array is a list of data items.

All the data items must be of the same data type.

Arrays must be declared before you use them….

Example :

```
Dim PartyList(3) of String
```

This would declare an array of 4 strings called PartyList. Each string in the array is identified by a subscript. The subscripts in this example go from 0 to 3…

Refer to each string as PartyList(0), PartyList(1), PartyList(2) and PartyList(3).

If you want the subscripts to start from a number other than 0, then declare the first and last subscript…

```
Dim PartyList(1 to 5) of String
```

…would allow the 5 strings PartyList(1), PartyList(2),…, PartyList(5)

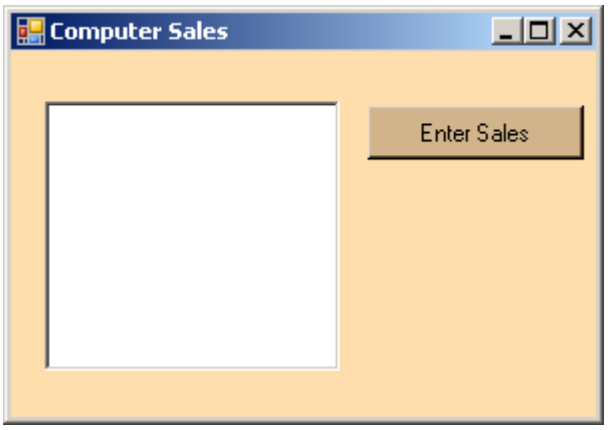
For global arrays use the declaration…

```
Public PartyList(1 to 5) of String
```

HANDS ON

[1] Create a new Windows Application. You are going to create a program that allows a computer salesman to enter the value of sales for each day of the week, and output the value on the best day.

Place a Button (btnEnter) and Listbox (lstTemps) on the form.



[2] Enter the array declaration…(directly after the Public Class Form1 declaration)

```
Public Class Form1

    Dim Sales(4) As Single
```

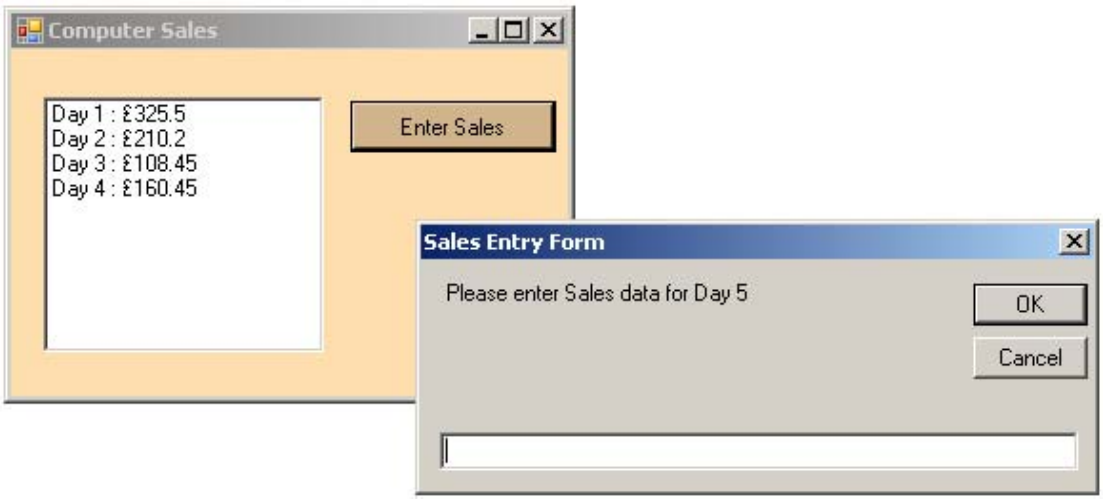
This is placed here so that we can use the array in any of the subroutines in the form.

[3] On the Click event of the Button, enter the following event handler…

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnEnter.Click
   Dim i As Integer
   Dim SaleData As Single

   For i = 0 To 4
       'Enter Sales data
       SaleData = InputBox("Please enter Sales data for Day "  &
(i + 1), "Sales Entry Form")
       Sales(i) = SaleData

       'Add Data to ListBox
       lstSales.Items.Add("Day " & (i + 1) & " : £" & Sales(i))
   Next
End Sub
```

Note that the subscripts for the array are 0 to 4, but the user sees them in the display as 1 to 5.

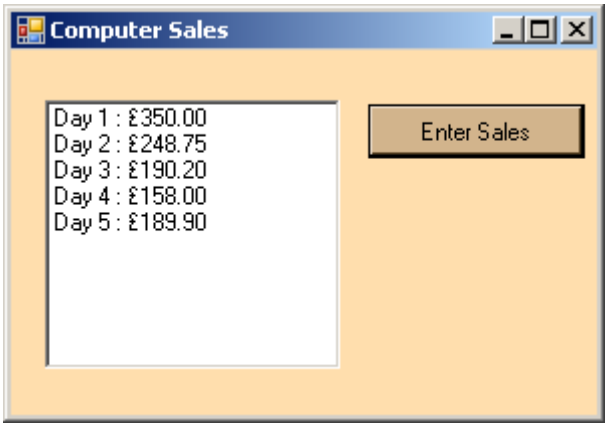
Run the program and you should be able to enter five sales amounts:



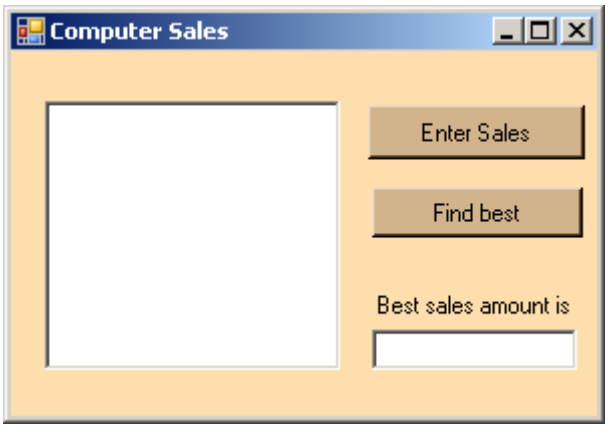
[4] The currency amounts in the ListBox do not show the pence to 2 decimal places. To format the numbers, use the FormatCurrency method…

```
'Add Data to ListBox
lstSales.Items.Add("Day " & i + 1 & " : " & FormatCurrency(Sales(i), 2))
```

Run the program again and enter five sales amounts…



[5] To find the largest sales amount of the week, we need to another Button (btnCalculate), a TextBox (txtBest) and a Label.

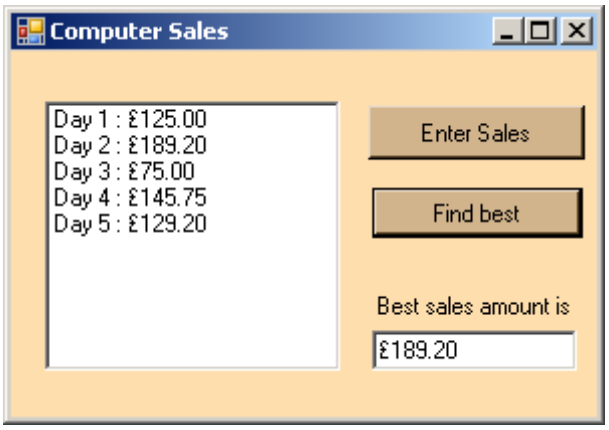


On the Click event of btnCalculate…

```
Private Sub btnCalculate_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnCalculate.Click
        Dim i As Integer
        Dim Largest As Single

        'Initialise to 0
        Largest = 0

        'Check each sales amount for largest so far
        For i = 0 To 4
            If Sales(i) > Largest Then
                Largest = Sales(i)
            End If
        Next

        'Display largest sales amount
        txtBest.Text = FormatCurrency(Largest, 2)

End Sub
```

This is the standard algorithm for finding the largest number in an array. The variable 'Largest' stores the largest so far…as each number is checked in turn.

[5] Run the program and enter five sales amounts…



[6] Run several test runs.

Save the Application – you need it in the next set of Challenges.

An Array is a really useful data structure that has a number of built-in methods already.

For example, you can sort an array into order…

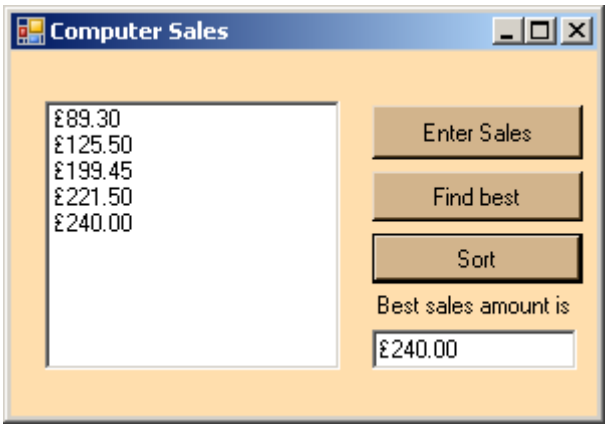
```
Array.Sort(Sales)
```

Other methods you may wish to investigate include Array.Find, Array.Reverse, Array. Copy and Array.Clear

---

Visual Basic Challenges 7

[1] Extend the previous application to produce a sorted list of the Sales amounts as shown below…



[2] Write a program that stores an array of 5 items and an array that stores their 5 prices. Use the code below in the **Form1_Load** event for setting up the arrays of data.

```
Item(0) = "T-Shirt"
Item(1) = "Pencil case"
Item(2) = "Ruler"
Item(3) = "Paper weight"
Item(4) = "Folder"

Price(0) = 7.99
Price(1) = 2.2
Price(2) = 1
Price(3) = 3.99
Price(4) = 0.4
```

The user should be able to enter the name of an item, and your program should display its price.

If the user enters "**Pencil case**", the price displayed should be **£2.30**

[3] Extend the exercise [2] to allow the user to calculate the bill for the purchase of a number of one of these items, allowing 5% discount.

Test data : 5 Pencil cases should cost **£10.45**

[4] Set up a password entry program that allows the user to enter a name and a password. If the name matches the password then a 'Welcome' message is displayed.

The user is allowed three attempts before the program ends.

HINTS :

Use two arrays – one for the names and the other for the passwords.

Year 12 : Visual Basic Tutorial.

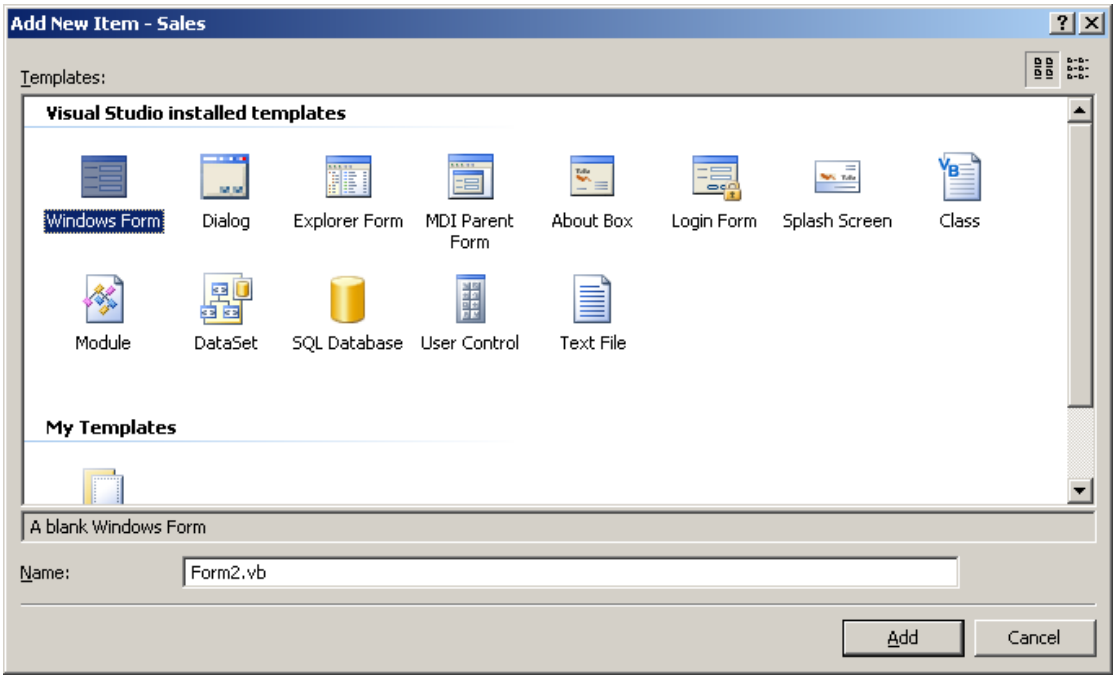
STUDY THIS

Forms.

Most applications have more than one form. Make sure you call them meaningful names (not Form1, Form2 etc!!!). Each form is saved as a different file on disc.

Each form has its own objects, properties, methods and event handlers.

To add a new form to a project…
In the [Project] menu – Add Windows Form…Select Windows Form…



…and change the name of the form to something meaningful.

The new form should appear in the Project Explorer window of your project

My

There is a special object called **My**. This object allows you to access the forms, computer and application of your project easily.

To open a form (called MyForm) in a subroutine, use…

```
My.Forms.MyForm.Show()
```

and to close a form….

```
My.Forms.MyForm.Hide()
```

…but you need to be careful!! A form cannot refer to itself…you need to use…

Me

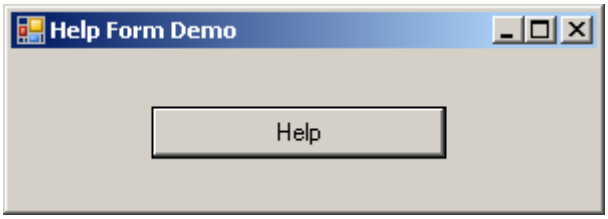
The **Me** object refers to the currently active form. So if you have a form with a button on it, and you want to close the form when the button is clicked you need to use…

```
Me.Hide()
```

HANDS ON

[1] Create a new Windows application.

Add a **Button** (btnHelp)



[2] Add another **Form** to the application and name it frmHelp.

On this form, place a **Button** (btnReturn).

[3] On the Click event of btnHelp enter the code…

```
    Private Sub btnHelp_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnHelp.Click
        My.Forms.frmHelp.Show()
    End Sub
```

[4] On the Click event of btnReturn enter the code…

```
Private Sub btnReturn_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnReturn.Click
        Me.Hide()
End Sub
```

[5] Run the form and you should be able to open the new form …and then close it.

HINT :

You can set the **StartPosition** property of a form to **CenterScreen** to place it in the middle of the screen when it is first opened.

STUDY THIS

Dialog Boxes

There are some special forms already created for you. A form that collects information from the user is called a **Dialog Box**.

An example of a Dialog Box is the **ColorDialog** object that allows the user to select a colour from a palette. The dialog box has its own properties that the programmer can set and then it is opened using the **ShowDialog** method. We will use this one in the next example.

Other Dialog Boxes available are:

- **OpenFileDialog** – for opening files
- **SaveFileDialog** – for saving files
- **FontDialog** – for setting font properties
- **FolderBrowserDialog** – for navigating through a disc's hierarchical folder structure.
- **PrintDialog** – Sets printing options
- **PrintPreviewDialog** – displays a print preview
- **PageSetupDialog** – for setting the propertuis of a page.

HANDS ON

[1] Create a new Windows Application

Add a **Button** (btnColour; Text = 'Set Font Colour') , and a **Label** (lblMessage; Text = "Test Message")

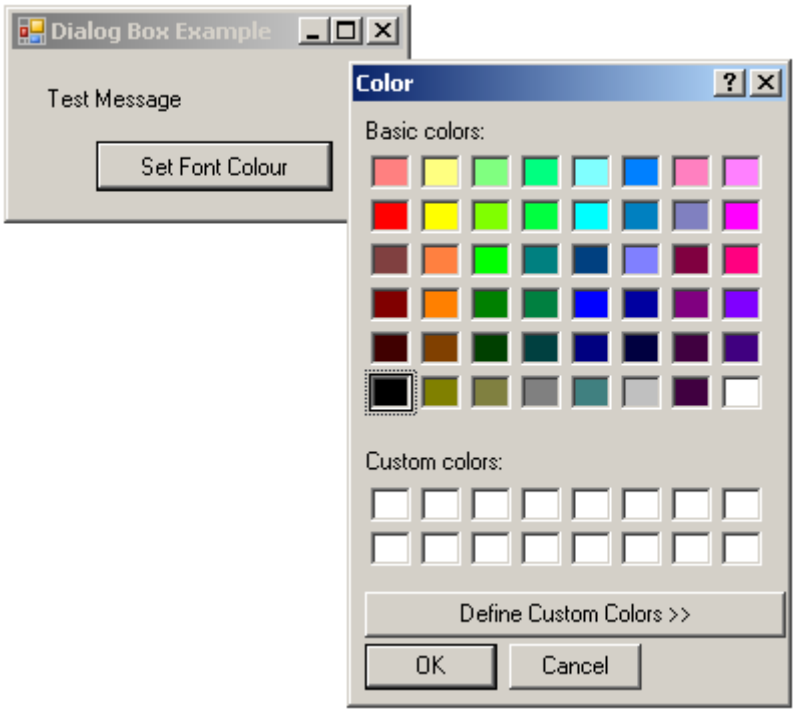
Also drag a **ColorDialog** object from the Toolbox onto the form. It should appear in the space below with the name ColorDialog1

[2] On the Click event handler of the button, enter the following code:

```
Private Sub btnColour_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnColour.Click
        ColorDialog1.ShowDialog()
End Sub
```

This code should open the Colour Dialog box.

Run the program and you should see the standard Windows colour selection dialog box....



[3] There is one more line needed in the code – one that sets the colour of the font of the label to the colour selected in the dialog box....

```
Private Sub btnColour_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnColour.Click
        ColorDialog1.ShowDialog()
        lblMessage.ForeColor = ColorDialog1.Color
End Sub
```

Run the program now, and you should be able to set the label's font colour...

Year 12 : Visual Basic Tutorial.

STUDY THIS

Subroutines.

A **subroutine** is a small program that performs a specific task. It can be '**called**' from anywhere in a larger program. When the subroutine has been run, program execution returns to the larger program.

There are two main types of subroutine:

Procedures – that perform a specific task.

Functions – perform a task and return a **value**. Functions are frequently used for calculating something.

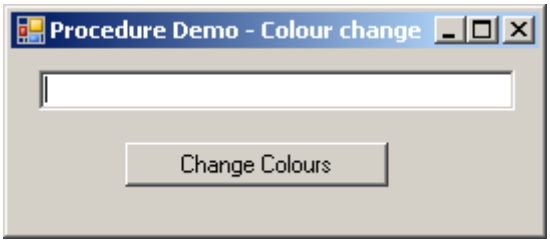
You have already met procedures because the event handlers are examples of procedures… but you can make your own. You are encouraged to do this because it creates a better structure to your program.

Bad programs have lots of repeated code. Good programs have lots of subroutines.

HANDS ON

[1] Create a new Windows Application.

On your form place a TextBox (TextBox1) and a Buttons (btnChange). Arrange them like this…



You are going to write a program using procedures, that toggles the colour schemes between two different schemes.

[2] You will need a variable to keep track of the current colour scheme, so make this declaration immediately after the Public Class Form1 line…

```
    Dim CurrentScheme As Integer = 1
```

Remember this means that we can use this variable in any subroutine on this form (Class).

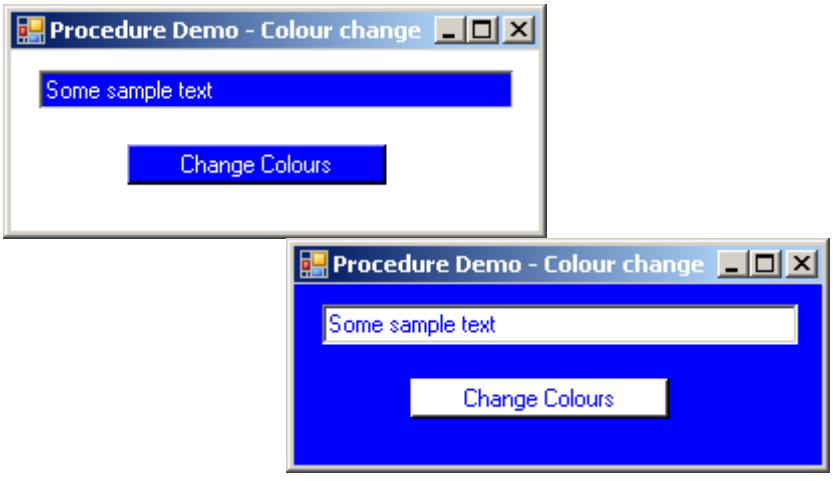
It is initialised to the value 1 when the program is run.

[3] The program consists of two subroutines (procedures). They are called SetColourScheme1 and SetColourScheme2. They are both called from the event handler btnChange_Click.

Type in the rest of the program as you see it here:

```
Public Class Form1

    Dim CurrentScheme As Integer = 1

    Private Sub SetColourScheme1()
        Me.BackColor = Color.Blue
        TextBox1.BackColor = Color.White
        TextBox1.ForeColor = Color.Blue
        btnChange.BackColor = Color.White
        btnChange.ForeColor = Color.Blue
        CurrentScheme = 1
    End Sub

    Private Sub SetColourScheme2()
        Me.BackColor = Color.White
        TextBox1.BackColor = Color.Blue
        TextBox1.ForeColor = Color.White
        btnChange.BackColor = Color.Blue
        btnChange.ForeColor = Color.White
        CurrentScheme = 2
    End Sub

    Private Sub btnChange_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
btnChange.Click
        If CurrentScheme = 1 Then
            SetColourScheme2()
        Else
            SetColourScheme1()
        End If
    End Sub
End Class
```

[4] **Run** the program, and enter text in the text box before pressing the button to change the colours.



STUDY THIS

Functions

Functions are procedures that return a value – in other words they work something out and assign the answer to the function name.

A **function** must have two things…

[1] a declared **type** for the returned value

[2] an **assignment** to the function **name** (saying what the value of the function is).

This example is a function that works out the largest of two numbers entered into two text boxes (txtFirst and txtSecond).

```
Private Function Largest() As Integer
    If txtFirst.Text > txtSecond.Text Then
        Largest = txtFirst.Text
    Else
        Largest = txtSecond.Text
    End If
End Function
```

The function is called inside an event handler by name and then, for example assigning it to a variable of the correct type…

```
Dim BestMark As Integer

BestMark = Largest()
```

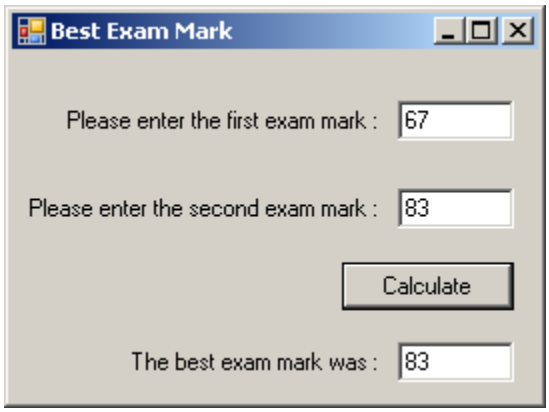
…or assigned to the property of an object…

```
txtBestMark.Text = Largest()
```

HANDS ON

Visual Basic Challenges 8.

[1] Use the function above to create a program that allows the user to enter two exam marks and displays which of the two is the highest mark.



Parameters

Subroutines only really become useful when we pass **parameters** to them.

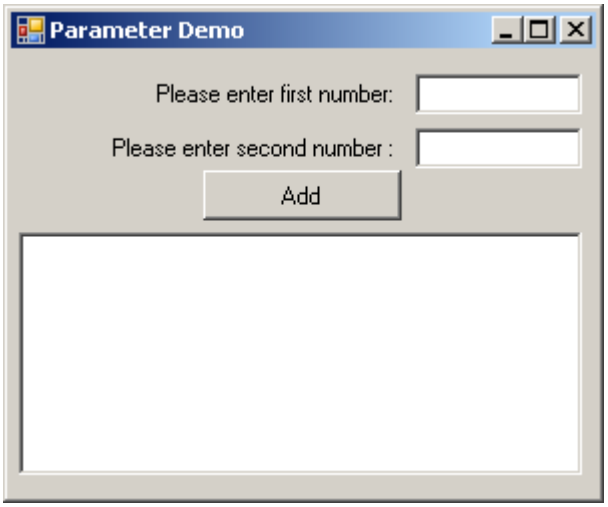
A **parameter** is a value that is passed to the subroutine. When the subroutine is executed, it will use this value.

Example : A procedure that draws a line of Xs in a TextBox…

HANDS ON

[1] Create a new Windows application.

Add two Labels, two TextBoxes(txtFirst and txtSecond) and a Button (btnAdd). Also add RichTextBox(rtbAdd)…

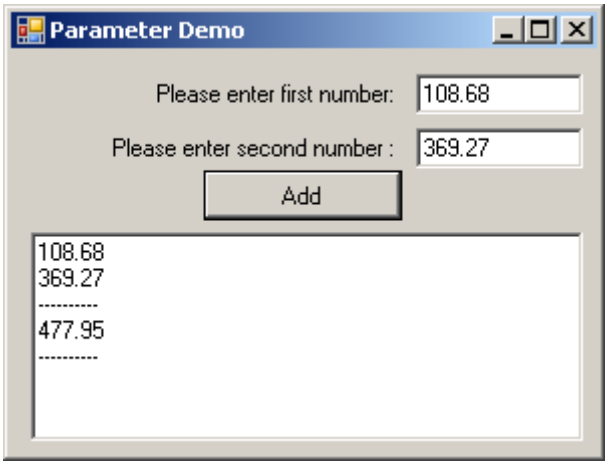


[2] On the Click event of button btnAdd add the event handler…

```
Private Sub btnCalculate_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnCalculate.Click

Dim Answer As Double
   Answer = Val(txtFirst.Text) + Val(txtSecond.Text)

   'Display calculation in TextBox
   rtbAdd.Clear()
   rtbAdd.AppendText(txtFirst.Text)
   rtbAdd.AppendText(vbCrLf) 'takes a new line
   rtbAdd.AppendText(txtSecond.Text)
   rtbAdd.AppendText(vbCrLf)
   rtbAdd.AppendText("----------") '10 dashes
   rtbAdd.AppendText(vbCrLf)
   rtbAdd.AppendText(Answer)
   rtbAdd.AppendText(vbCrLf)
   rtbAdd.AppendText("----------")
   rtbAdd.AppendText(vbCrLf)
End Sub
```

There are two lines that are repeated here for drawing the line of dashes – this is never a good thing and you should avoid repeated code in programming.

[3] Run the program and enter two numbers. The addition calculation should be displayed.



[4] To avoid repeated code … create your own subroutine…Change your code to the following:

```
    Private Sub DrawLine()
        rtbAdd.AppendText("----------")
        rtbAdd.AppendText(vbCrLf)
    End Sub

    Private Sub btnCalculate_Click(ByVal sender As
    System.Object, ByVal e As System.EventArgs) Handles
    btnCalculate.Click

        Dim Answer As Double
        Answer = Val(txtFirst.Text) + Val(txtSecond.Text)

        'Display calculation in TextBox
        rtbAdd.Clear()
        rtbAdd.AppendText(txtFirst.Text)
        rtbAdd.AppendText(vbCrLf) 'takes a new line
        rtbAdd.AppendText(txtSecond.Text)
        rtbAdd.AppendText(vbCrLf)
        DrawLine()
        rtbAdd.AppendText(Answer)
        rtbAdd.AppendText(vbCrLf)
        DrawLine()
    End Sub
```

Note that your procedure is called DrawLine and is called twice by the event handler.

[5] Now for some improvements….

First, it is better to use a loop in the DrawLine procedure, so change it to…

```
    Private Sub DrawLine()
        Dim i As Integer
        For i = 1 To 10
            rtbAdd.AppendText("-")
        Next i
        rtbAdd.AppendText(vbCrLf)
    End Sub
```

[6] The procedure is fine for drawing lines of 10 dashes…but maybe sometimes we would like lines of 20 dashes, …or 25 dashes etc…

To make the procedure more useful we pass a **parameter** to it….

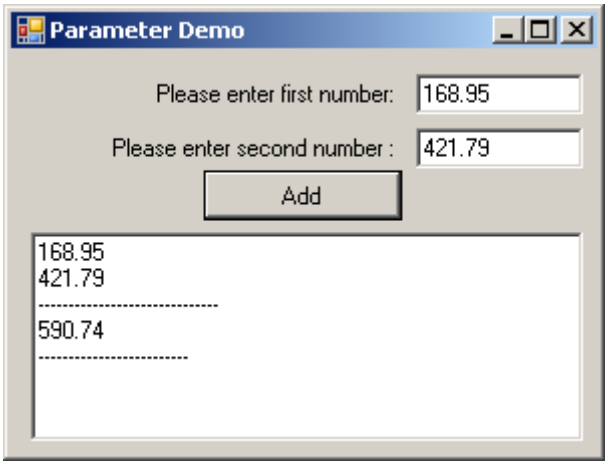
```
Private Sub DrawLine(ByVal NumDashes As Integer)
    Dim i As Integer
    For i = 1 To NumDashes
        rtbAdd.AppendText("-")
    Next i
    rtbAdd.AppendText(vbCrLf)
End Sub
```

NumDashes is the parameter. It is declared in the heading of the procedure- the data type of the parameter must also be declared.

In the event handler you will need to pass a value for the parameter – this must match the data type (integer in this case)…so change the lines that call the procedure to…

```
.....
        DrawLine(30)
.....
        DrawLine(25)
```

Running the program now should result in a display similar to this…



Summary

A subroutine is a small section of program code that can be called from other parts of a program. There are two types:

Procedure – that performs a specific task
Function – that performs a task and **returns a value**.

Parameters are passed to subroutines to make them useful.

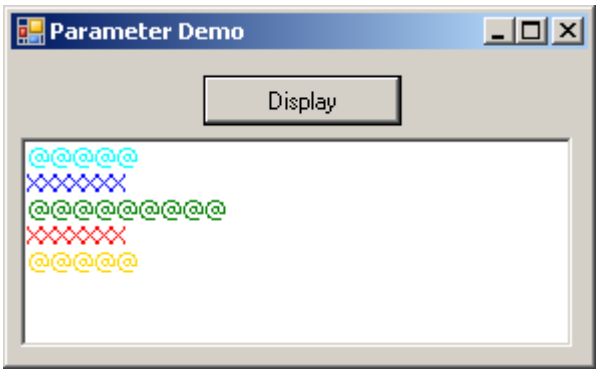
Visual Basic Challenges 8

[2] Enhance the above program so that you can pass two parameters to the DrawLine procedure –

- the number of characters to be drawn
- the character to use

So the instruction DrawLine(12,"@") would produce "@@@@@@@@@@@@"

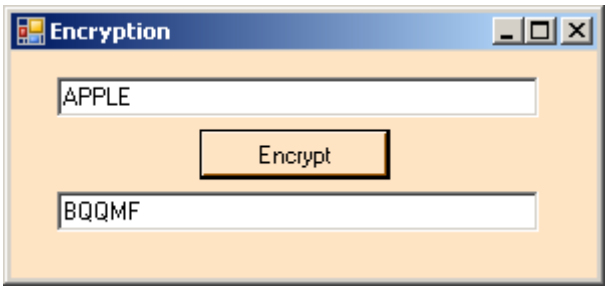
Test your program by seeing if you can reproduce this screen display…



[3] **(a)** Write a new application that allows the user to input a string and encodes it by taking the 'next' character in the alphabet for each letter.

Include a function in your application that encrypts a string.

Test data : Input – APPLE Output – BQQMF



(b) Add a new section that decrypts a coded string.

Test Data : Input – BQQMF Output – APPLE

STUDY THIS

Modules.

A module is a library of subroutines that can be used in other programs. It is a really good idea to use modules because…

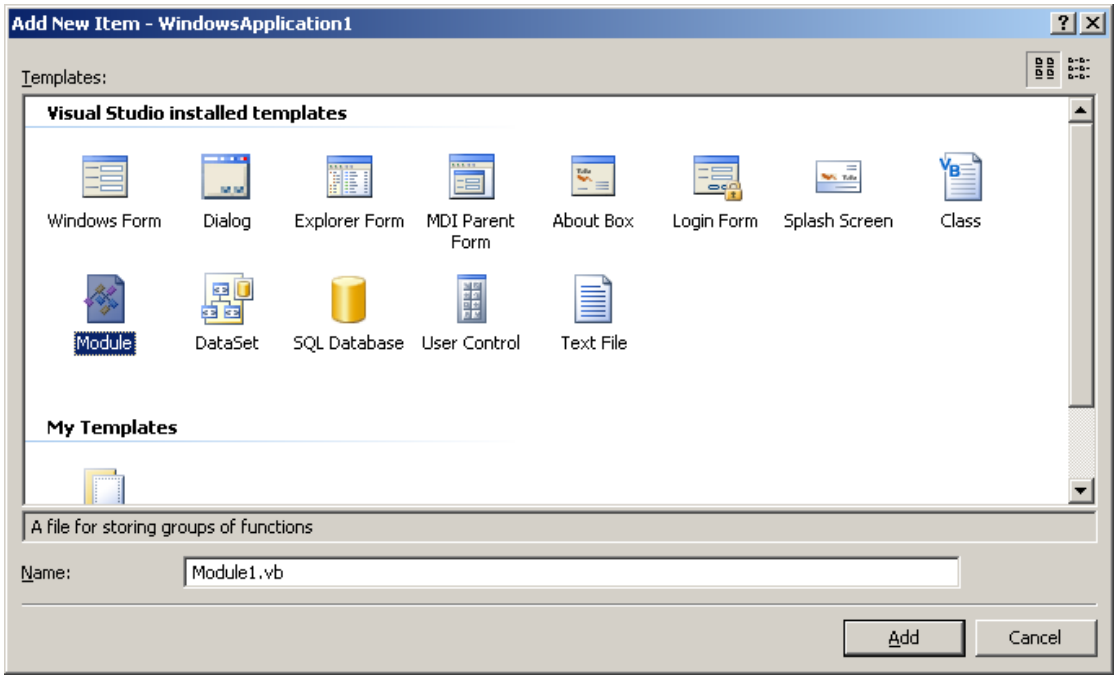
- it saves time programming if you can use subroutines you have created in other programs
- you know they will work because they have already been tested.

Any subroutine in a module can be called from anywhere in your program.

Global variables and constants can be declared in a module and used anywhere in the program.

Using constants is also a good idea because if their value changes, then you only have to change the value once in the module, and not in all the places the value is used in the program.

To add a module to a program click the [Project] menu and the [Add New Item…] option. Make sure you select the [Module] option…



The Module should appear in your Solution Explorer window.

Year 12 : Visual Basic Tutorial.

STUDY THIS

Files.

A **file** is a place for storing data that you do not want to lose when the power of your computer is switched off.

There are two main types of file…

1) **Serial file** – data is appended onto the end of the file.
2) **Random Access file** – data is stored in the file at a place calculated from the data.

In applications where data is needed to be accessed quickly then you need a Random Access file.

In a Random Access File of data, a calculation (hashing algorithm) is performed on the key field, resulting in an address (hash address) where the data is stored in the file.

Sometimes a Random Access File is called a Direct Access File. This is specifically designed to confuse you!

HANDS ON

Text Files

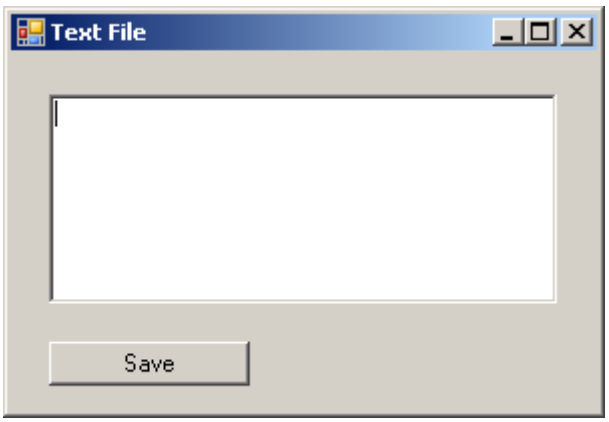
You are going to write a program that allows text to be input and then saved into a file. Later, you will write a program that loads it back.

[1] Create a new Windows application.

On your form place a TextBox (txtData) and a Button (btnSave).

Set the following properties for the txtData…

Property	Value
MultiLine	True



[2] To save the work, you will use a SaveFileDialog control, so drag one from the Toolbox onto your project. It will appear at the bottom of the screen.

Enter the subroutine below into the Click event handler of the btnSave.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
   'Set the Dialog box to only display Text files
   SaveFileDialog1.Filter = "Text files (*.txt)|*.txt"

   'Open the Dialog box
   SaveFileDialog1.ShowDialog()

   'Check that a filename has been entered
   If SaveFileDialog1.FileName <> "" Then
   'Write the text to the file
   My.Computer.FileSystem.WriteAllText(SaveFileDialog1.FileName, txtData.Text, False)

   End If
End Sub
```

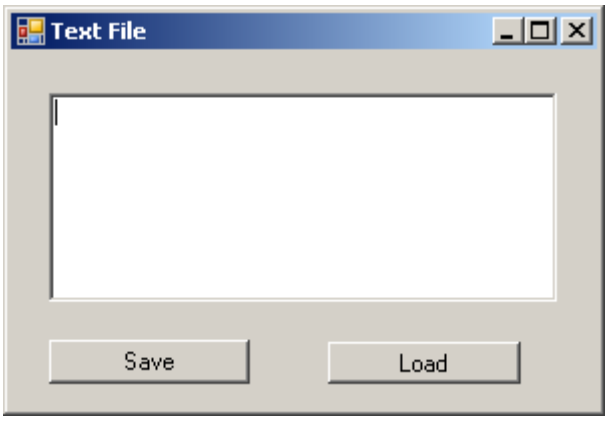
NB : There is a Boolean parameter in the WriteAllText command…This will be True if you want to append the text onto the end of the file…or False, if you want to overwrite any existing text in the file.

[3] Run, the program, enter some text into the TextBox and click on the Save button. Enter a filename in the SaveFileDialog, and click on OK.

Your text should now be saved in a text file. (Check it by opening with Notepad.)

[4] Now let's try to get it back!

Stop the program running and add a new Button (btnLoad) to your form.



You will also need to drag an OpenFileDialog control into your project.

This should appear at the bottom of your screen with the name OpenFileDialog1.

[5] On the Click event handler of btnLoad…

```
Private Sub btnLoad_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnLoad.Click

    'Set the Dialog filter to display only text files
    OpenFileDialog1.Filter = "Text files (*.txt)|*.txt"

    'Open the Dialog
    OpenFileDialog1.ShowDialog()

    If OpenFileDialog1.FileName <> "" Then

     'FileOpen(1, OpenFileDialog1.FileName, OpenMode.Input)
    txtData.Text = My.Computer.FileSystem.ReadAllText(OpenFileDialog1.FileName)

    End If

End Sub
```

Run the program and see if you can load the text back.

HANDS ON

Visual Basic Challenges 9

[1] The latest school trip is going to Paris to see the Eiffel Tower and to practice their French.

Write an application that allows pupils to enter their names, one at a time.

The whole list of names should be printed at the end.

[2] Write an application that allows the user to enter a paragraph of text and store it in a file.

The text file can then be loaded in a coded version where all the vowels are removed from the text.

Test data : If the Text "Sing a song of sixpence is entered", then when it is loaded back, the text "Sng sng f sxpnc" is displayed.

STUDY THIS

Random Access Files

To illustrate Random Access Files, you are going to create a file of records for the members of a school drama society.

Each record will have 4 fields in :

Fieldname	Data type
ID	Integer (Key field)
Name	String
Form	String
Actor	Boolean

The records will be stored in a random access file. The ID numbers will start at 1000, and the hashing algorithm will find the address of each record by subtracting 1000. For example, the record with ID 1004 will be stored as record number 4.

HANDS ON

[1] Create a new Windows application.

Records are called Structures in Visual Basic, and the first thing you need to do is define the record structure. Do this in a Module.

```
Structure MemberRecord

        Dim ID As Integer
        Dim Name As String
        Dim Form As String
        Dim Actor As Boolean

End Structure
```

[2] On your Form, add three TextBoxes (txtID, txtName, txtForm), a CheckBox (chkActor), and a Button (btnSave)…and anything else to make the display look appealing…

[3] Here is one of the member records to be entered :

ID	Name	Form	Actor
1004	Tom Jones	12G	True

The program will look at the ID number (1004) and store this record as record number 4.

When creating a Random Access File, you need to :

- Assign values to the fields of a record
- Open a file for random access
- Calculate the hash address of the record
- Save the record at that hash address
- Close the file

The event handler for the Click event of btnSave is here :

```
Private Sub btnSave_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSave.Click
   'Declare a record
   Dim Member As MemberRecord

   'Allocate values to fields in the record
   Member.ID = txtID.Text
   Member.Name = txtName.Text
   Member.Form = txtForm.Text
   Member.Actor = chkActor.Checked

   'Allocate a file number - (let the computer do it!)
   Dim FileNum As Integer
   FileNum = FreeFile()

   'Open the file for Random Access - Change the file path if needed
   FileOpen(FileNum, "H:\My Documents\DramaFile.dat", OpenMode.Random)

   'Calculate the hash address of the record
   Dim RecNum As Integer
   RecNum = Member.ID - 1000

   'Write the record to the file
   FilePut(FileNum, Member, RecNum)

   'Close the file
   FileClose(FileNum)

   'Clear the TextBoxes
   txtID.Text = ""
   txtName.Text = ""
   txtForm.Text = ""
   chkActor.Checked = False

End Sub
```

Run the program and enter the record shown.

(If you open the file Dramafile.dat in Windows Notepad, you should see the data – only the text will be recognisable amongst other garbage!)

[3] Use your program to add these records to your file :

ID	Name	Form	Actor
1002	Alice Springs	11C	True
1004	Tom Jones	12G	True
1005	Jack Flash	12B	False
1007	Rhian Lord	11B	True
1008	Elvis May	12G	False

[4] Now you will try to retrieve the data…You are going to add a new form to your application and search for a particular record.

Add a new Windows Form to your application and name it frmSearch.

On this form, place 3 TextBoxes (txtID, txtName, txtForm), a Button (btnSearch) , a CheckBox (chkActor) and 4 Labels…

Save this form and add a Button (btnSearchForm) to the original form, and add the event handler.

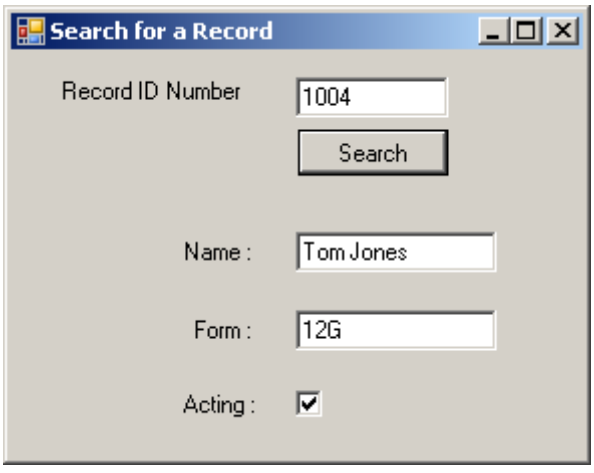
```
Private Sub btnSearchForm_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
btnSearchForm.Click
        frmSearch.Show()
End Sub
```

It would be a good idea to run the program and check that you can open the Search Form. It does nothing yet - but you are going to be able to enter an ID number, click the Search button and find and display the appropriate record.

On frmSearch, add this event handler to the Click event of btnSearch…

```
Private Sub btnSearch_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSearch.Click

  'Declare a record
  Dim Member As MemberRecord

  'Allocate a file number - (let the computer do it!)
  Dim FileNum As Integer
  FileNum = FreeFile()

  'Open the file for Random Access - Change the file path if needed
  FileOpen(FileNum, "H:\My Documents\DramaFile.dat", OpenMode.Random)

  'Calculate the hash address of the record to be read
  Dim RecNum As Integer
  RecNum = txtID.Text - 1000

  'Read the record from the file
  FileGet(FileNum, Member, RecNum)

  'Display the fields in the TextBoxes
  txtID.Text = Member.ID
  txtName.Text = Member.Name
  txtForm.Text = Member.Form
  chkActor.Checked = Member.Actor

  'Close the file
  FileClose(FileNum)

End Sub
```

[5] Run the program and enter an ID number…Click the search button and you should see the fields of the record displayed.



Save the application – you will need it in the Challenge exercises….

Visual Basic Challenges 9

[3] Add a new form to the Drama Club application, that displays the names of all the members in a ListBox.

HINT : Use a loop to read each record. You can test when you get to the end of a file by using …and be careful you don't try to display the blank records.

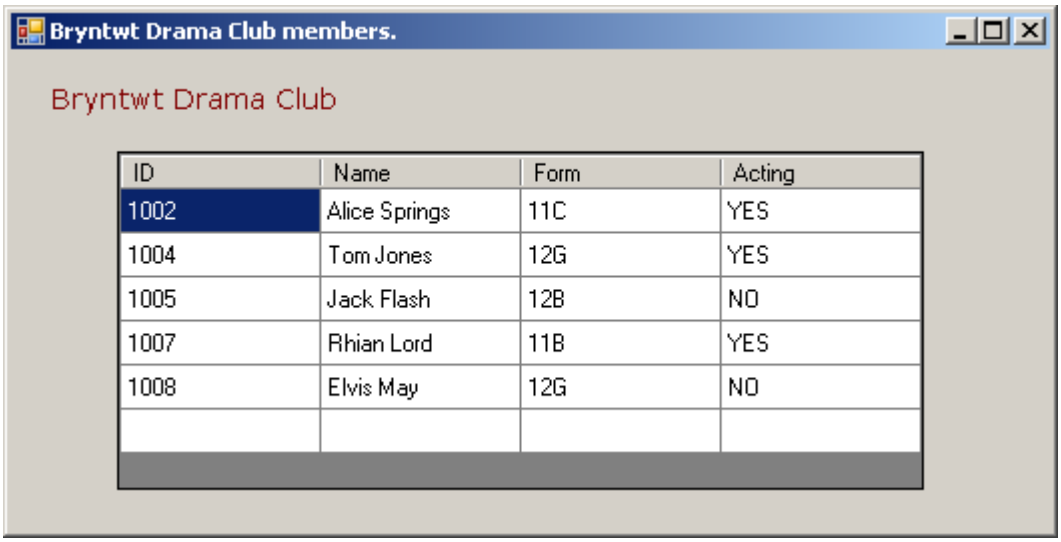
```
While Not EOF(FileNum)
      …
      …
End While
```

[4] For the brave!!…

RESEARCH NEEDED

One of the most useful objects in Visual Basic is the DataGridView.

See if you can display the Drama Club members on a DataGridView…



Year 12 : Visual Basic Tutorial.

ADO – Linking to Microsoft Access Databases.

IMPORTANT :
You will use a database called hospital.mdb for the walkthroughs and exercises presented here. The path to this database will always be **C:\hospital.mdb** in these notes – You will have to change this to the path of the database on your computer, whenever it occurs.

HANDS ON

There are two ways to link to a database in VB –

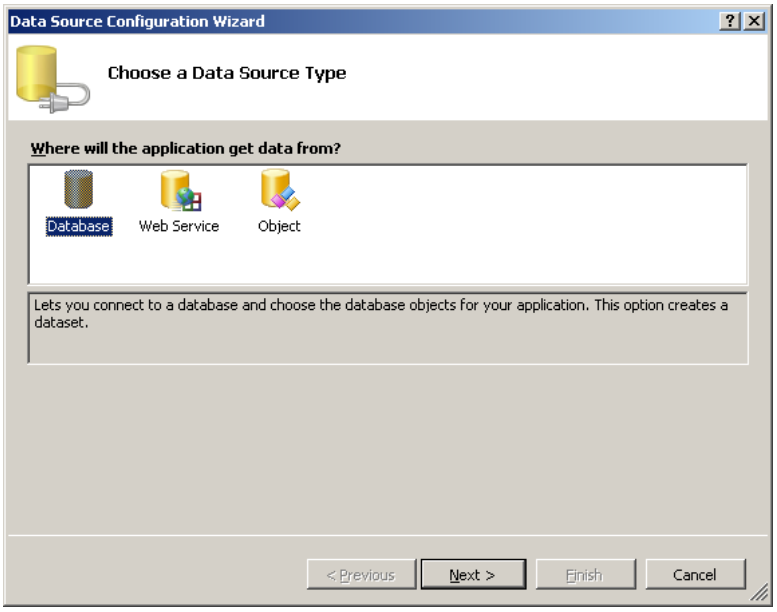
[A] The Easy Way (faster but not so versatile) or

[B] by writing Program Code.

We'll stick to the Easy Way for now….

[1] Create a new Windows application.

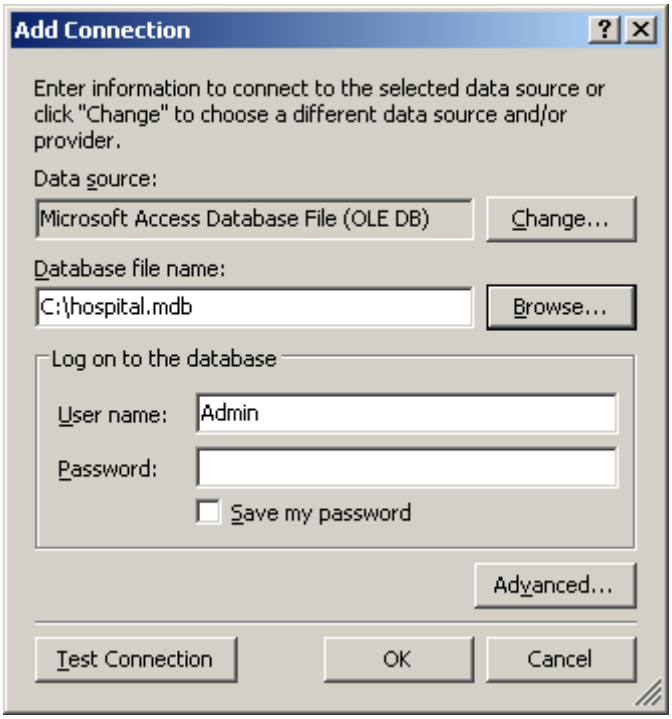
[2] In the Data menu, click the **Add New Data Source** command. The Data Source Configuration wizard should appear…



Select **Database** and click **‹Next›**

You now need to set up the connection to your database.

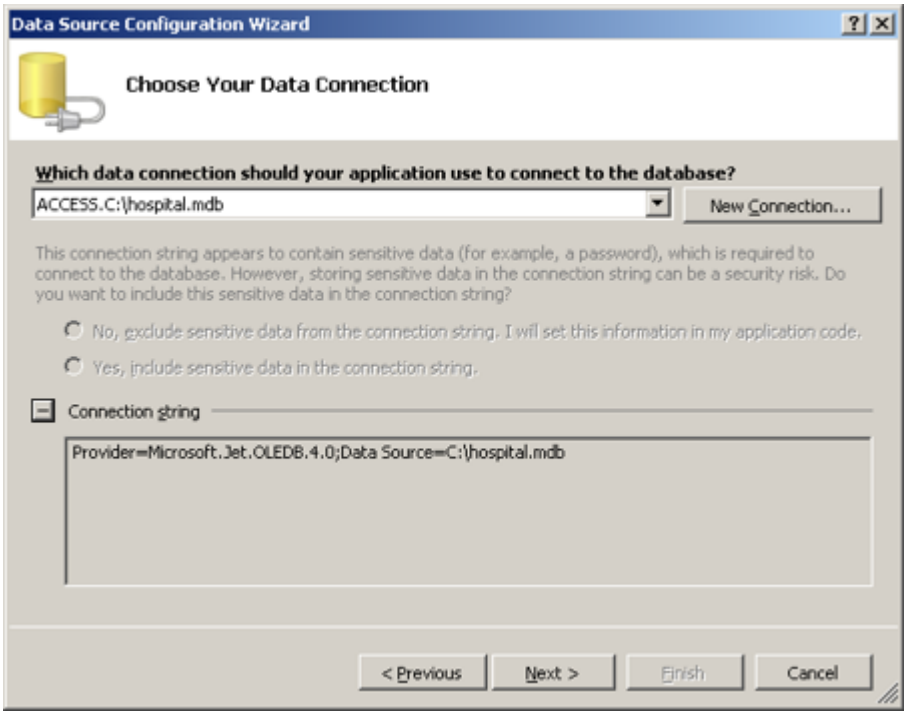
Click on the [**New Connection**] button.



Make sure the dialog box is set to the above options (the path to your database will probably be different – use the Browse button to find your database)

Test the connection before moving on by clicking [**OK**].

The details of the connection should now be filled in (Click the [+] to see the connection string.)

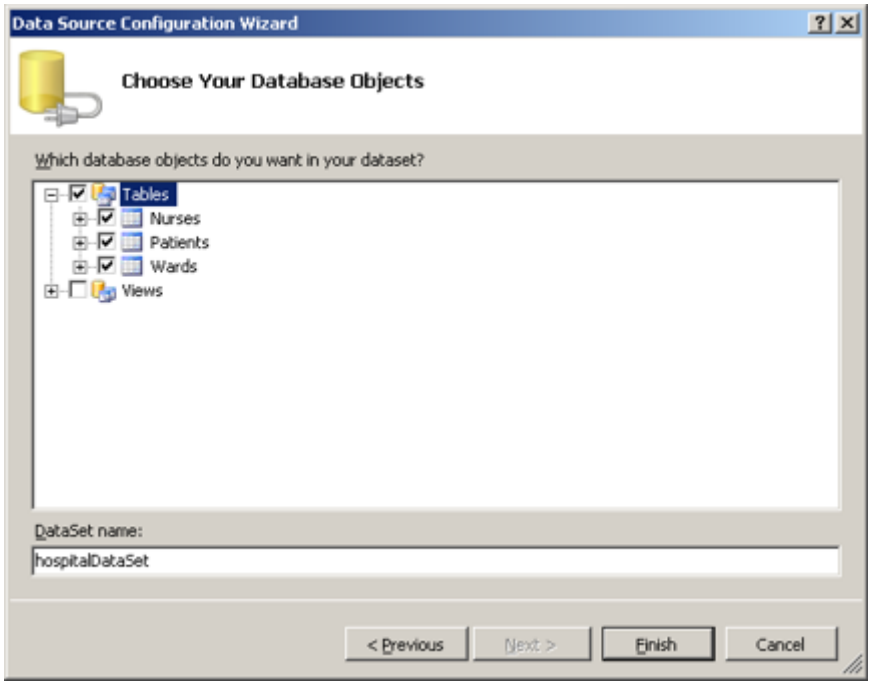


Click the <**Next**> button.

You will see a message displayed asking whether you want to make a copy of the database into your project. There is no need to do this so select [**No**].

You will now be asked if you want to save the Connection string..and [**Yes**] – you do! (If the location of your database changes then you only need to edit the string in the configuration file of your Solution explorer)

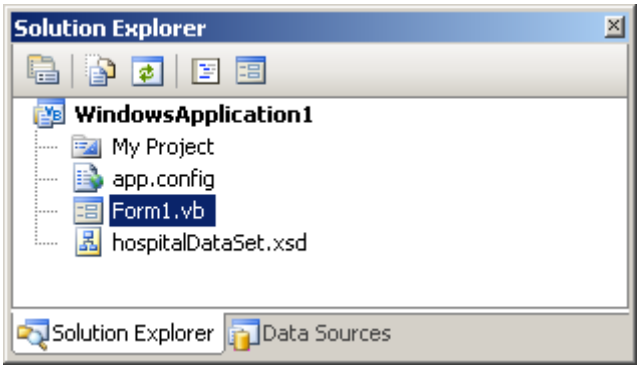
You now create the **dataset** to be used in your application. A dataset is a copy of some or all of the fields in the tables of your database.



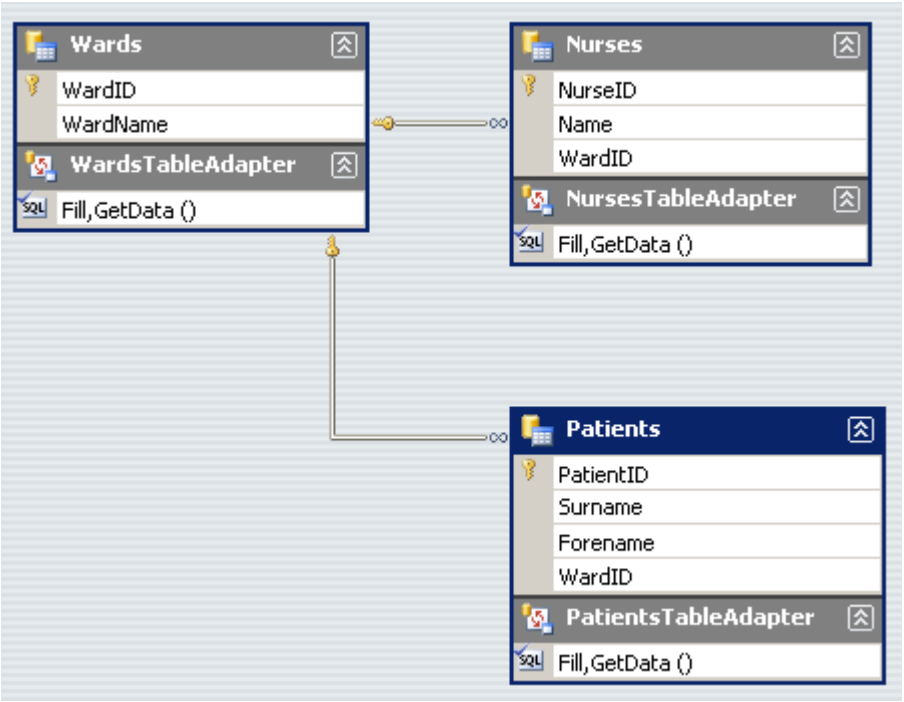
Select all the tables (as shown above).

Click [**Finish**]

[3] You should now see an extra element appear in the Solution Explorer.
There is now hospitalDataSet.xsd.

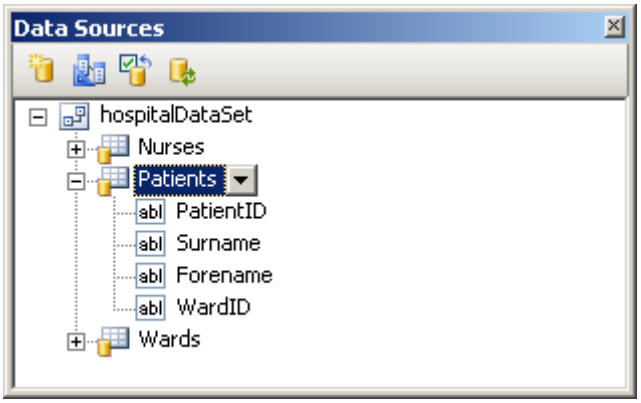


If you right-click hospitalDataSet.xsd and view the Designer you should see a visual representation of the schema of the database.



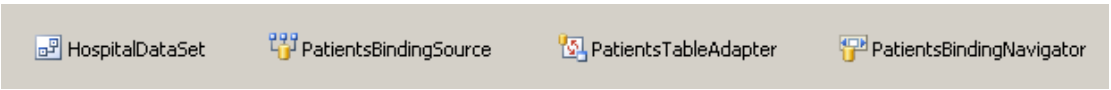
[4] You will now write an application that displays some of this data.

Find your **Data Sources** window. The tables and fields of your dataset should be displayed. Expand the Patients table…



Drag each of the four fields of the Patients table onto the Form. Labels and TextBoxes should be created for each.

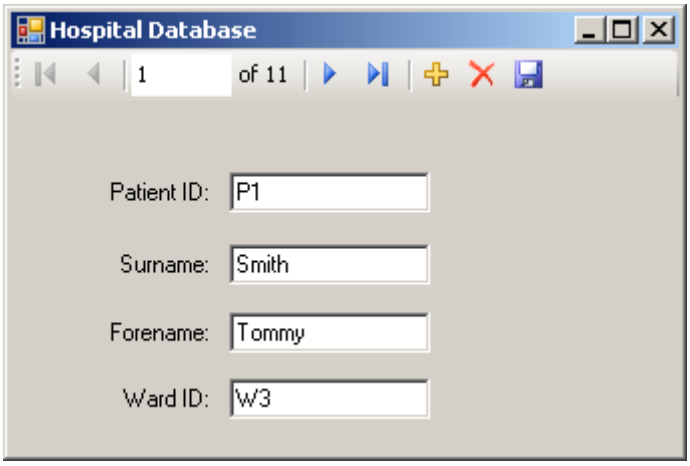
A **Navigation Bar** is also created at the top of the form, as well as these object displayed at the bottom of the screen…



A TableAdapter moves data between the database and the DataSet.

A BindingSource makes sure objects display data from the DataSet.

[5] Run the program and you should be able to see data from the Patients Table..



Using the Navigator Bar you can look at each of the records in the table, navigate to the First, Last, Next or Previous record.

You can also Add new patients to the table and save the data, or delete records (careful!).

Save this project – you will need it in the next chapter.

Year 12 : Visual Basic Tutorial.

STUDY THIS

ADO – Adding SQL statements.

SQL stands for Structured Query Language. It is a language used to select or update data in a database.
You need to become familiar with the structure of SQL **SELECT** statements.

Here is an example :

```
SELECT PatientID, Surname, Forename FROM [Patients] WHERE Surname = 'Jones'
ORDER BY Surname
```

This will filter out all the Patients with surname 'Jones', and sort the results in alphabetical order of Surname.

..but don't panic – there is a Query Builder wizard that will do this for you!

You will also look at a really useful object for displaying data – the **DataGridView**.

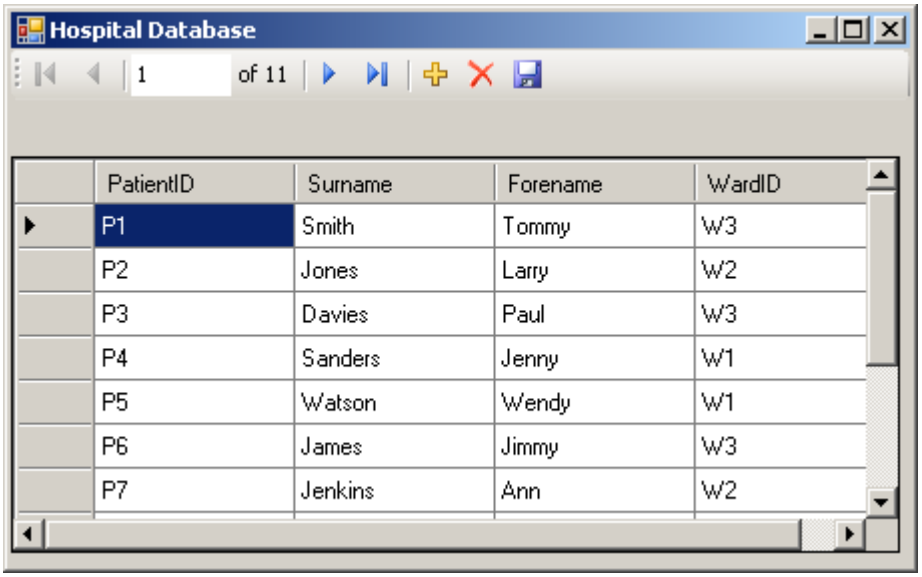
HANDS ON

[1] Open the **Hospital** application from the previous chapter.

Delete all the Labels and TextBoxes… you are going to replace it with a DataGridView.

From the Data Sources window, drag the Patients Table onto the form. A DataGridView will appear, but you may need to adjust its size.

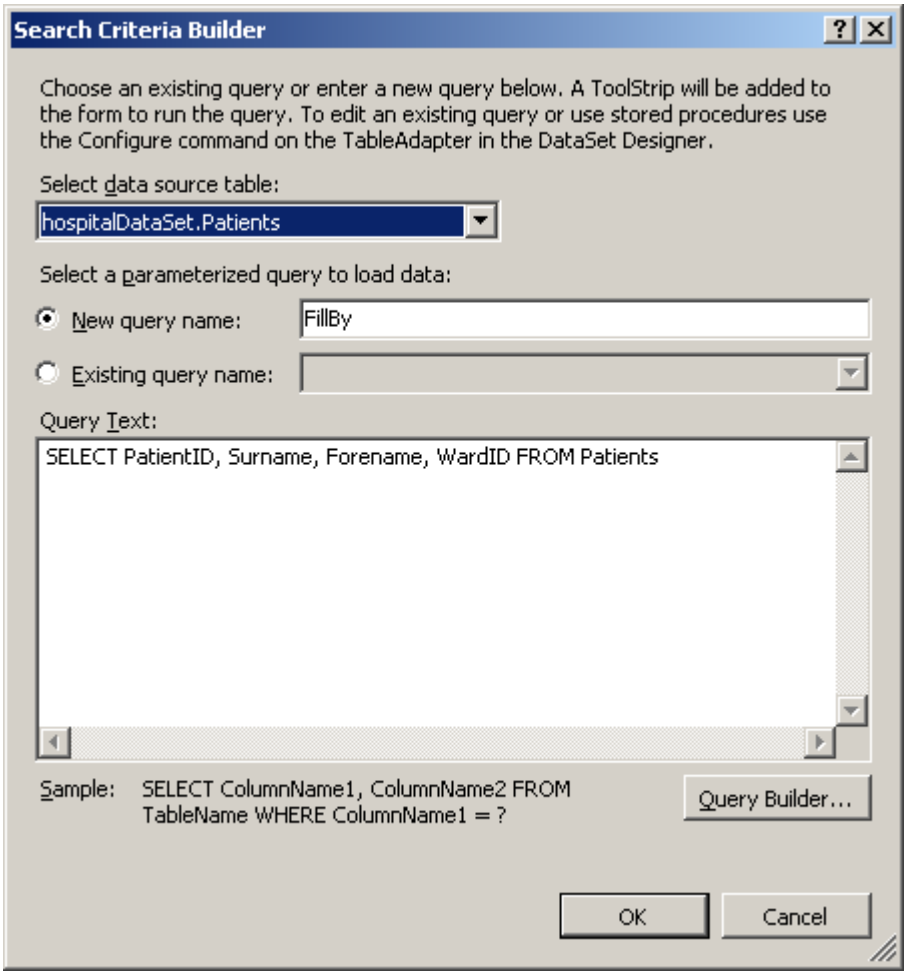
Run the program and all the data records should be displayed in a grid.



The DataGridView is a very powerful tool for displaying data and can be formatted in many ways – worth having a good look at this for your coursework!

[2] Let's filter the data now. Suppose we only wanted to view the Patients in Ward W1, and we would like them displayed in alphabetical order of surname.

In the Designer view, click on the DataGridView and then in the [Data] menu, select [Add Query]. This dialog box should appear…



The default Query for the Patients Table Adapter is displayed.

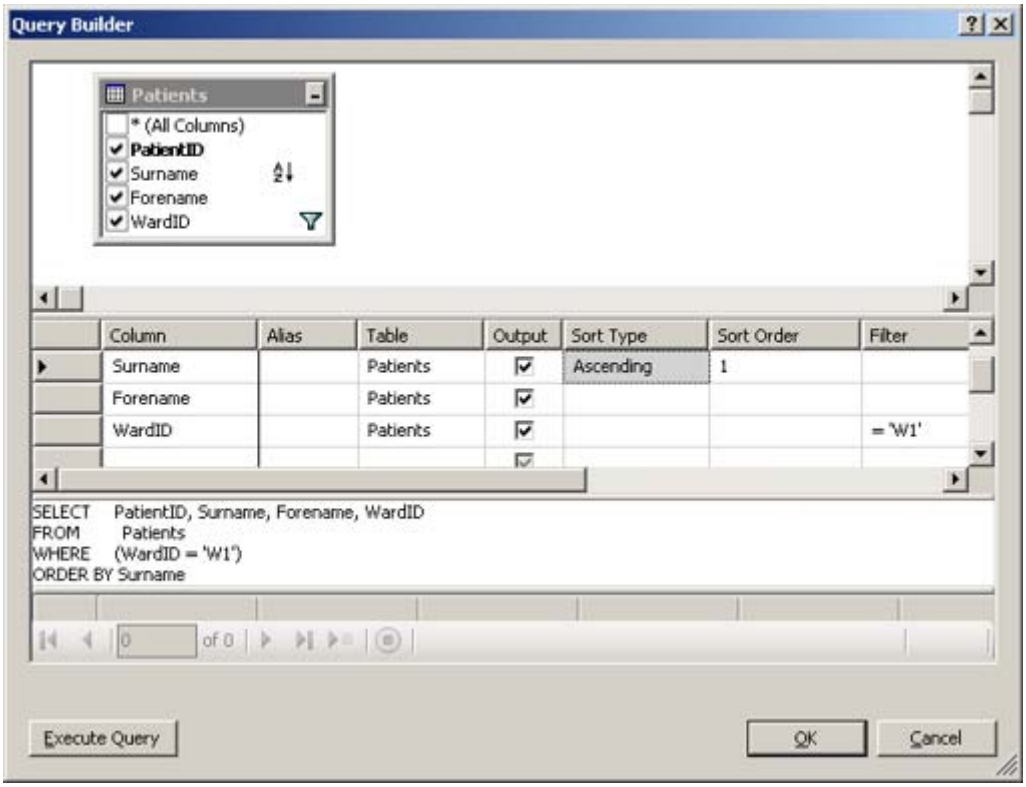
You could enter your SQL statement in the Query Text box, but let's use the Query Builder…

The data source table is displayed and should not be changed – we are displaying data from the Patients table.

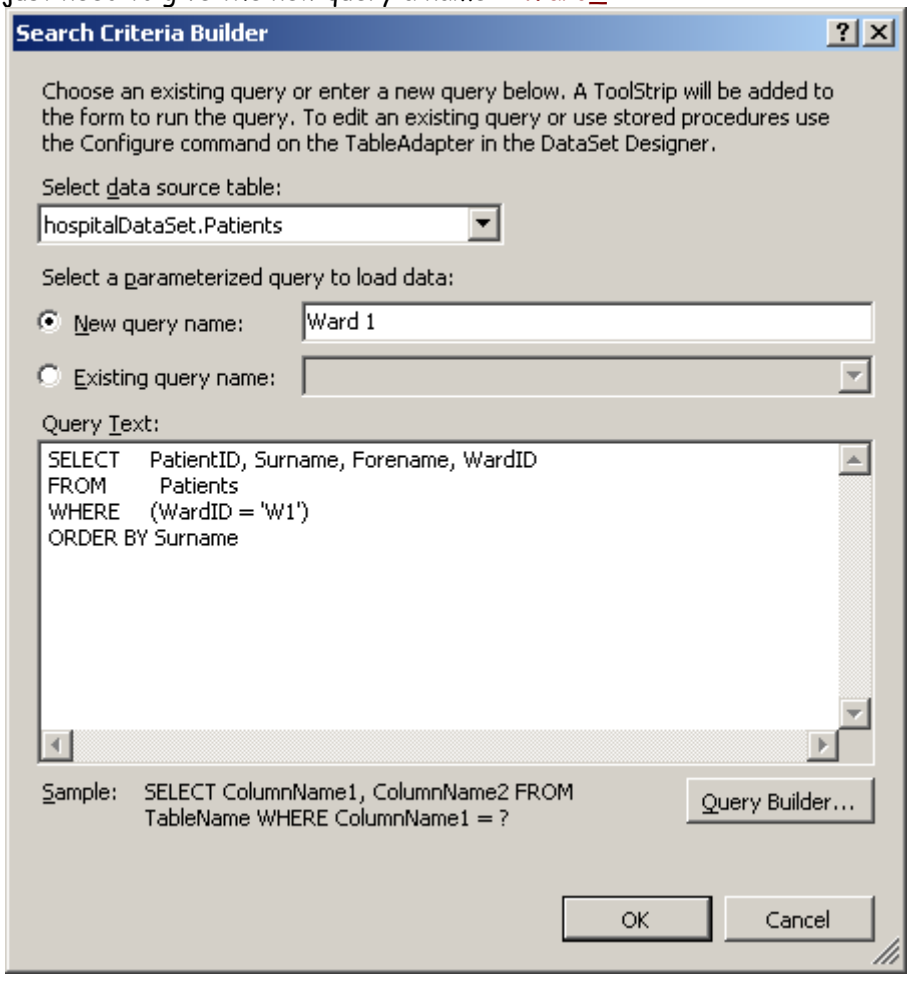
In the Query builder…

- set the Sort Type for the [Surname] field to be 'Ascending',
- set the Filter for the [WardID] field to be 'W1'

You will see the SQL text change automatically in the text box at the bottom.



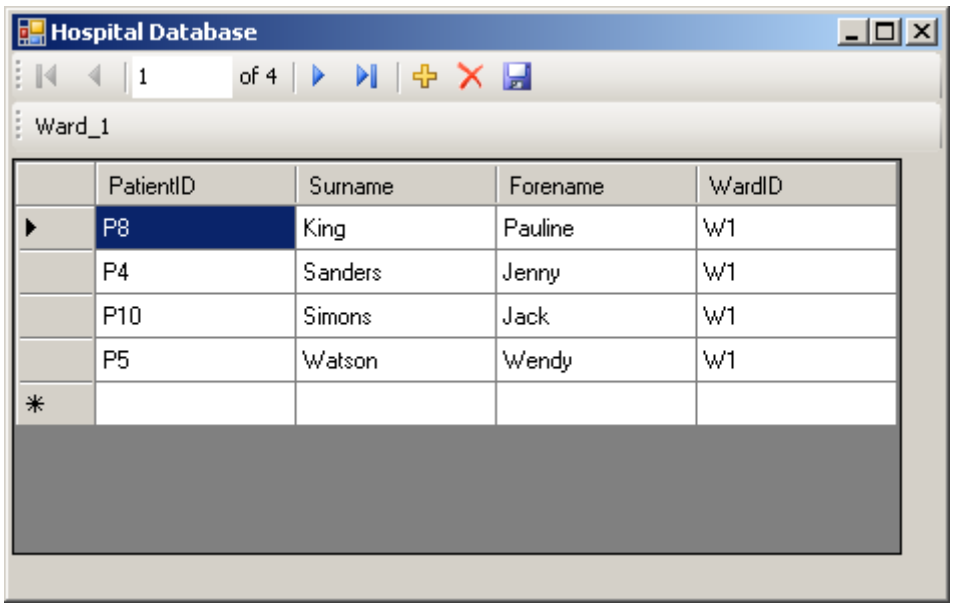
Click [OK]. The SQL text will be automatically transferred.
You just need to give the new query a name – 'Ward_1'



Make sure your screen looks like the one above ..and click [OK]

VB will add a new Toolbar to your program with a Button on it for the Query.

[3] Run the program and click the button and you should see the Patients in Ward W1 displayed in alphabetical order of surname.



Important Note : If you need to edit this query, go to the Dataset Designer, and use the Configure option on the Query.