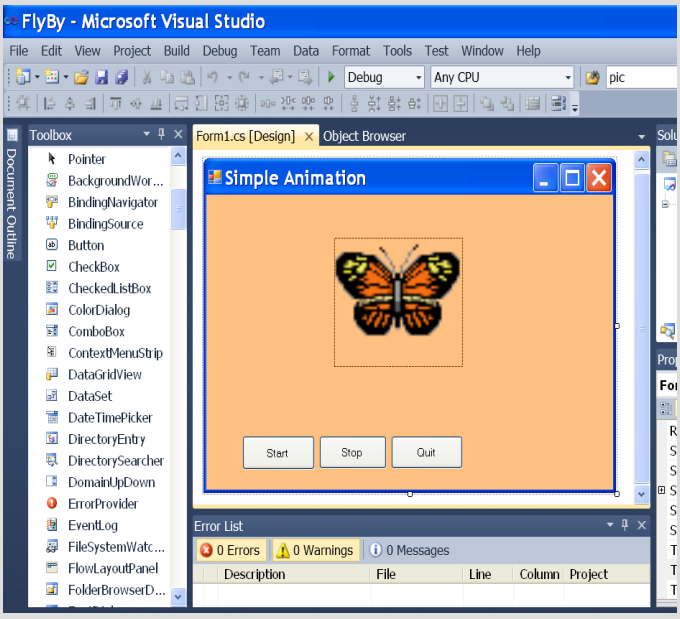
Windows Programming

bucks new university

.NET Windows Programming Using C#



Application Programming (CO453)

Part B Weeks 5-8

Using MS Visual Studio C#
A beginner's guide

Starting C#

- First get into Windows
- Double-click the **Visual Studio** icon on the desktop (or use Start/programs)
- The first time it is run, you will need to choose the **Visual C# Development Settings**

Starting a NEW Project

- Select **File** from the menubar, followed by **New then Project**
- Select a template: **Console Application**
- Click **Browse** to find suitable **Location** for storing your project
- Enter a **Name** for your new project, then click **OK**
- Now you can type your program code into the new window

Opening an EXISTING Project

- Select **File** from the menubar, followed by **Open** followed by **Project**
- In the **Open Project** dialog window, select the correct folder and **.csproj** file
- Click **Open** and the project should be loaded together with any source code **.cs** files
- To see the code, you may have to click the **.cs** file in the Solution Explorer window

Compiling a Program

- Make sure the project and program are open
- Select **Build** from the menubar, followed by **Build Solution** (or use **F6** key)
    - Edit any syntax errors that show up in the bottom of the editor window

Running (executing) a Program

- Make sure the project and program are open
- Select **Debug** from the menubar, followed by **Start Without Debugging** (or use **Ctrl** + **F5**)
- If the program doesn't work as expected, you have logical errors and have to correct these.

Saving a Program

- Choose **File** from the menubar, followed by **Save All**
    - This saves to the location previously chosen

Copying a Project

- If you want to take a copy of a project so that you can make changes while keeping the original, go to where the project has been saved and copy the whole project folder
    - This folder holds the many files required by the project .. paste it to a new location

Closing down Visual C#

- Select **File** from the menubar, followed by **Exit**

Running a Program without Visual Studio

- Open the project folder
- Open its **bin** folder
- Open its **debug** folder
- Find the **exe** file for the project .. you can copy and paste this to a different area and double-click this to run your project without using Visual Studio.

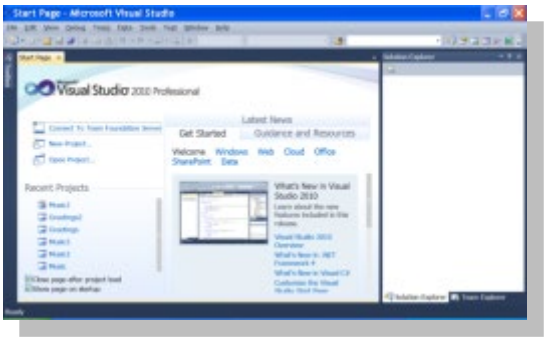
Introduction to Windows Programming

The MS **Windows** graphical platform provides a way of designing visual programs that have a high degree of **interaction** with the user.

- Visual Studio allows us to easily create Windows programs by putting predefined objects or **Controls** onto forms using the **.NET** framework libraries. Code can then be added to these controls to increase functionality
- **C#** is an **Object Oriented** language and every C# Windows program is made from one or more **Objects**. These **Objects** may be **Forms**, **Buttons**, **Text Boxes**, etc.
- All Objects have **Properties** (name, colour, size, text, etc.) which can change.
- Objects can respond to certain **Events**, like the clicking of the mouse or a key press. This type of program is **Event-Driven** as opposed to Process-Driven.
- Every C# Windows program has at least one **Form**, with other objects placed on it as required. **Code** is attached to objects so they do what is required when an event happens.
- Forms, objects and code together all make up a **Project**.

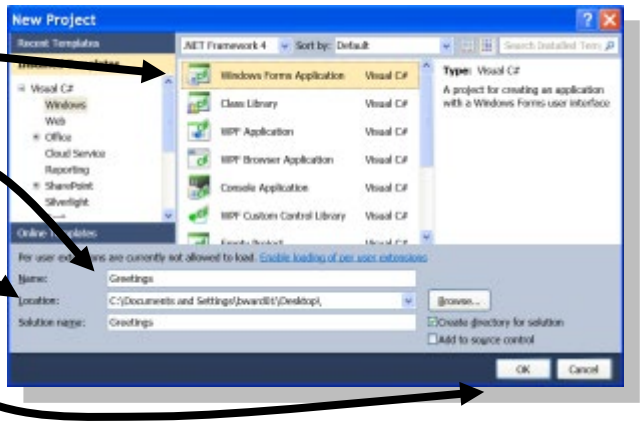
Starting Visual Studio

- If you are starting a new project, see below
- If you have used C# before, you may see a display of recent projects for you to select
- Alternatively you can open an existing file by selecting **Open Project**



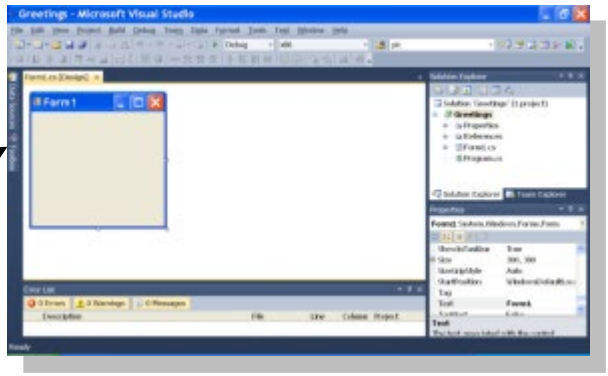
Starting a New Windows Project

- To create a new project, choose **File** from the menu, followed by **New Project**.
- In the **New Project** window, choose **Windows Forms Application**.
- Type a suitable name for the project .
- Use **Browse** to find a suitable **Location** for storing the project
- Finally click the **OK** button .. and you will be taken to the design area of your project .. where a blank form awaits you!



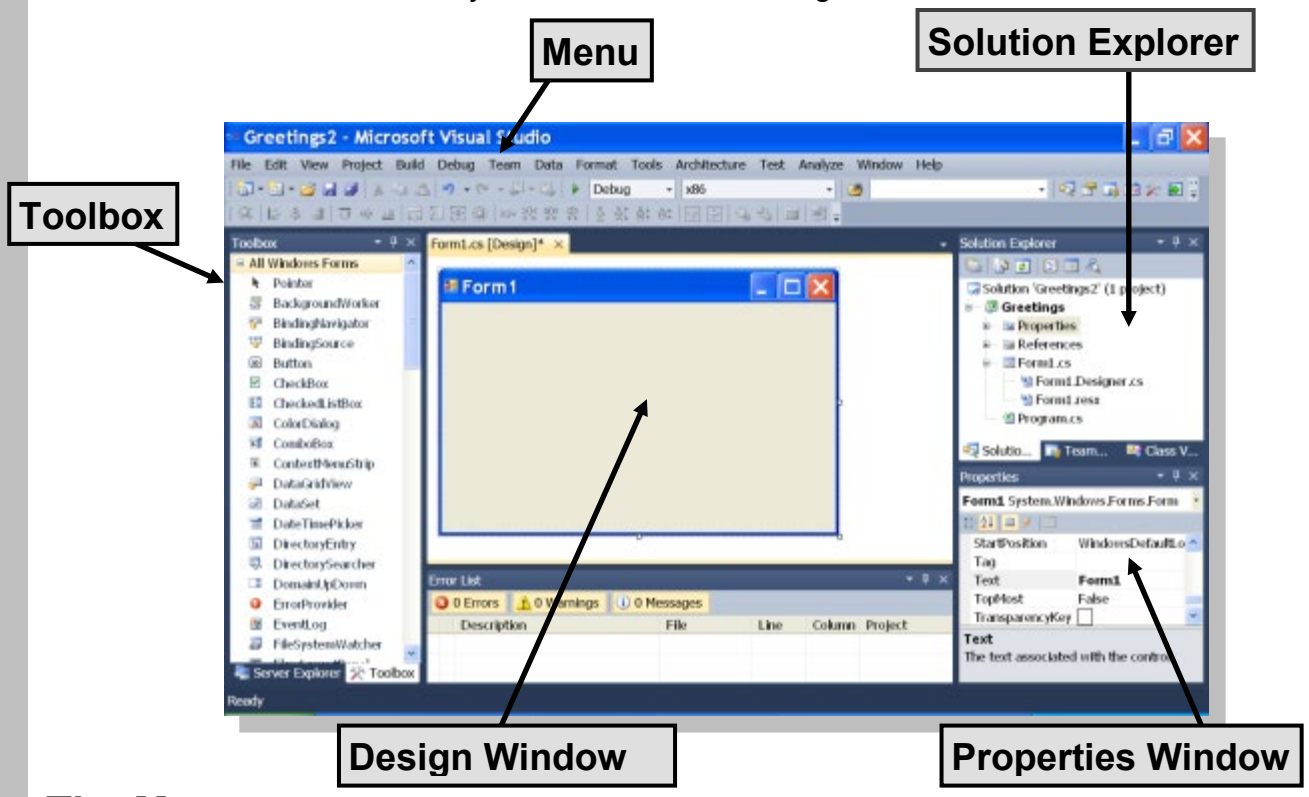
Setting the Environment

- You need a setup that is similar to the one shown on the next page
- Make sure the properties window is open .. if not, choose **View** from the menubar, followed by **Properties Window**
- You can dock the **ToolBox** by clicking here and then clicking the **AutoHide** button



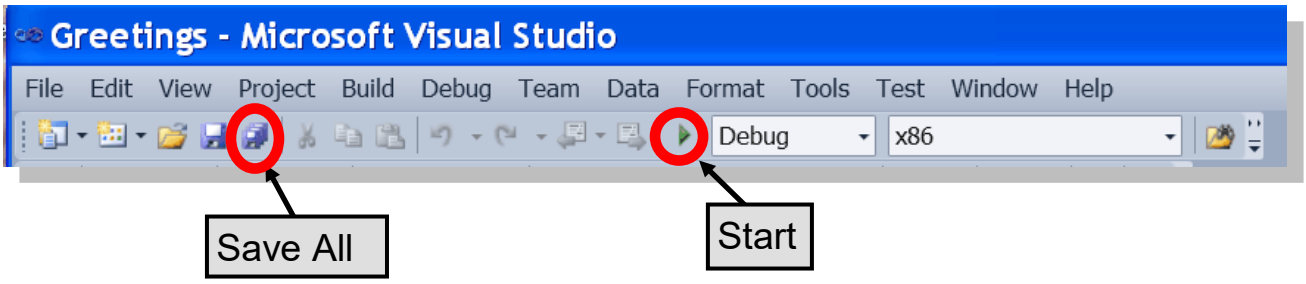
Looking at the IDE

- You should now have a layout that looks something like this:



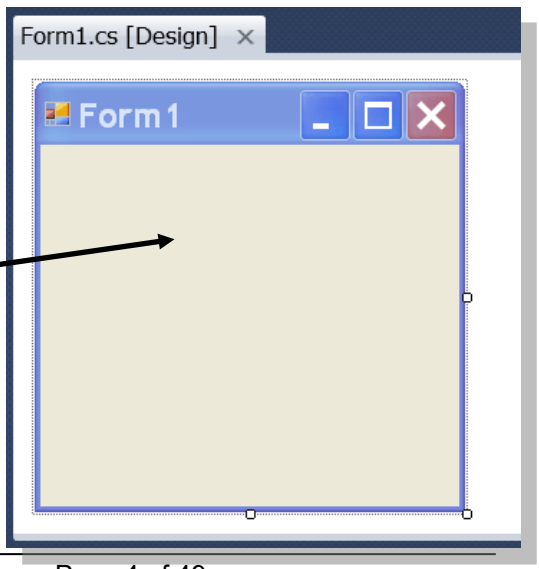
The Menu

- The menu provides many advanced features. There are two buttons that we will be regularly using, these are the **Save all** button, which saves all the files within the project and the **Start** button which executes (runs) the project.

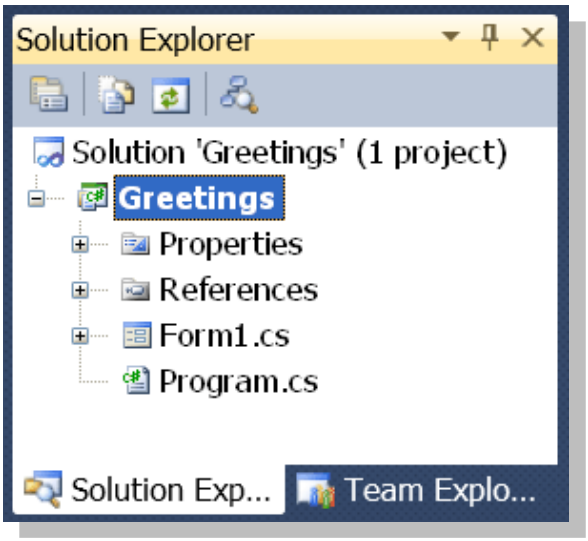


The Design Window

- It is here that interfaces are created. Within C#.NET each interface or window within the application is represented by a **form**.
- During the development of an application we place **buttons**, **text boxes** and other **controls** onto a form to create an interface.
- It is through these interfaces that the user can interact with the program.
- **Note:** You may find its a good idea to make this form bigger just drag the edges to do this.

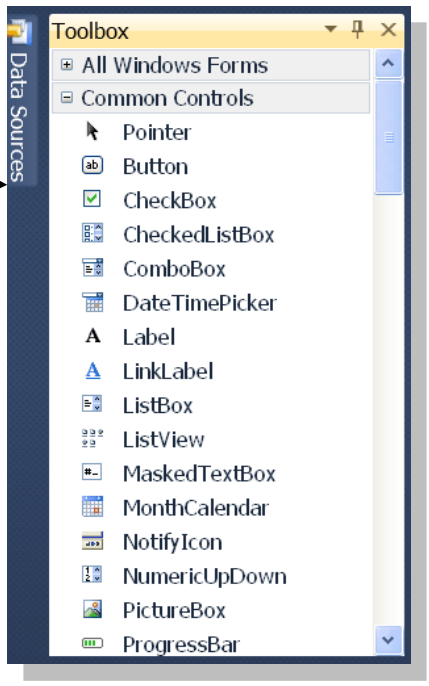


The Solution Explorer

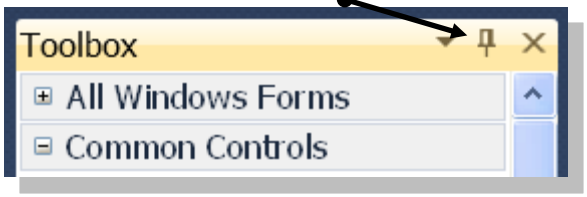


- This displays all the project files in a tree - like structure.
- The explorer can typically be used to switch between the various forms within the project at design time.
- In this project there is only one form named **Form1.cs**
- This is really the only file you need to bother with at the moment.
- If the form is not displaying at any time, double-click this form name to display it.

The Toolbox

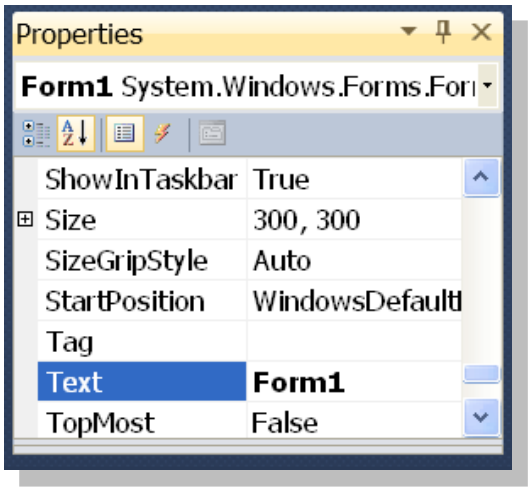


- Forms, buttons, textboxes etc are all known as **controls**.
- Apart from forms, all these controls can be found in the **Toolbox**.
- There are a number of tabs within the toolbox, however we will normally only be using controls from within the **Common Controls** tab.
- Like most of the windows, the Toolbox can be docked or undocked by clicking **Autohide**



- The Toolbox gives you a list of lots of different controls that can be placed on a form and then resized and positioned as you want.

The Properties Window



- Each form and each control in the Toolbox have a set of **properties** that affect appearance and behaviour.
- These properties can be set in the properties window.
- A control is first selected (click on it) .. then its properties are displayed in this **Properties Window**.
- If this window (or any other) is not visible, choose **View** from the menubar, followed by **Properties Window**

C# Windows Programming
Unit 1

Introduction, Calculator and Splash Screens

1.1 Task: Change Some Object Properties

- Create a new .NET project and call it Greetings
- Once created, by default, you will be in Design mode, with one object visible, a **Form**. This is your main working area and it will also become the user interface for your program.
- C# objects have many **properties** that can be changed, but most of them can be left with their default values. We shall change some properties of the **Form** object:

Task 1.1: Form Properties

First select the **Form** by clicking it once. Make sure the Properties Window is visible (or press the F4 key to reveal it)
The name of the object (**Form1**) should appear under the Properties window title bar.

1. Change the **Text** Property
   - The **Text** property is the caption that appears on the Form's blue title bar. Click once to select the Text property and type a new caption: **SuperSoft Software Co.**
   - Press the enter key and notice your new caption appears in the title bar of your form.
2. Change the **Name** Property
   - This is the internal name for the form. Select the **Name** property (scroll to the top of the Properties Window) and then type a new name for your form: **frmWelcome** (press the enter key and note a name change in the Properties Window).
3. Change the **BackColor** Property
   - Now select the BackColor property. Click on the list box ☑ and select the **Custom** tab
   - Choose a red colour.
4. Change the **FormBorderStyle** Property
   - This sets the borders of the form. Try **Fixed3D**.
5. Change the **StartPosition** Property
   - This sets the borders of the form. Try **CenterScreen**.

Running a Project

- Although the program will simply display a blank form when executed, it is good to run the application at this stage to ensure Visual Studio is configured correctly.
- On running the application Visual Studio will check for errors within any code you have created.
- To run the application select **Start Debugging** from the main menubar.
  - a new red window should appear in the centre of the screen
- To stop the program, select **Debug** followed by **Stop Debugging** from the main menubar or double-click the **Stop Debugging** button on the menu bar.

N.B. You can also start a program by pressing the **F5** key or stop it with **Shift-F5 (**or by clicking the form's close button).

Saving a Project

- Click the **Save All** button on the menubar or select **File** followed by **Save All**.
  n.b. a project has many files so it is safer to use Save All rather than just Save
- Your project will be saved to the location you chose at the start (see the introduction) overwriting any previous saves.

1.2 Task: Add Label Controls to the Form

- The various user interface elements such as textboxes, radio buttons etc. are collectively known as **Controls**.
- We will now add some label controls to the Greetings application and change some of their properties.
  First make sure you are in design mode and the Toolbox Window is visible
- **click once** on the **Label** tool in the Toolbox (the one with a large **A** on it)
- then move to the required position on the form and click again.
- A label control should now appear on the form but its **AutoSize** property may be set to **True** .. if you change this to **False** you will be able to stretch the box and fix it to any size you want.

Task 1.2: Labels

1. Add **two** label controls to the form, using the method described above.
2. Change the Label Properties using the property tables (see below)
3. Change the **Size** property of the **Form** to 600, 350
4. Run your program to see the result

Property:	Name	Text	AutoSize	Location (x, y)	Font	
					Name	FontSize
Label1:	lblGreeting	Welcome to SuperSoft	False	100, 30	Arial	20
Label2:	lblBrag	The BEST Prices	False	100, 90	Arial	16

Property:	Size	BorderSyle	TextAlign	BackColor	ForeColor
Label1:	370, 40	Fixed Single	MiddleCenter	Yellow	Blue
Label2:	310, 70	Fixed Single	MiddleCenter	Yellow	Blue

Adding a Quit Button

At the moment your program doesn't do very much. We will now try to change this.

- Buttons can have C# code attached to them so they will perform certain actions when they are clicked. We shall start by adding a button to end the program.
- Click the **Button** control in the Toolbox. Then click on the form. A button will appear on the form. Drag the button to the bottom left of the form.
- Change its **Name** property to **btnQuit** and **Text** property to **Quit**. Resize if necessary.
- Run the program and click the button .. it doesn't do anything yet!

Adding Code to the Quit Button

Every form or control has a set of **events** that it can recognize (e.g. a mouse-click event). We can attach a **function** or **method** to this button's click event. We shall now add some code to make the Quit button perform a program exit:

- In design mode, **Double-Click** the button on the form. This opens the **Code Window** for the button.
- Make sure the correct event name is displayed in the Members list at the top right of window (in this case it should be: **btnQuit_Click**)
- Look at the code window and see the namespace and partial class set up with a constructor and a code template for the **btnQuit_Click()** method.

```
namespace Greetings
{
    public partial class frmGreetings : Form
    {
        public frmGreetings()
        {
            InitializeComponent();
        }

        private void btnQuit_Click(object sender, EventArgs e)
        {

        }
    }
}
```

- Type the code to exit the application:
  Application.Exit();
  into the btnQuit_Click() method between its **{ }** brackets:

```
private void btnQuit_Click(object sender, EventArgs e)
{
        Application.Exit();
}
```

- Now **run** the program again .. clicking the button should now quit the program.
- **Note:** **Application.Exit()** may have problems running from a network .. if so, try running the program from a memory stick or another drive
  (or use **this.Close();** instead .. which closes the current form)
  You can also try **Environment.Exit(0);**

1.3 Adding More Buttons

We shall now add some buttons to change some of the properties as the program is running.

Changing a Text property

- Add another button to the form and change its **Name** property to **btnShow1**
- Change its **Text** property to **Best Computer**
- Double-click the button to get into the Code Window and add the following code to the **btnShow1_Click()** method:

  lblBrag.Text = "Supa Dupa Computer\n Selling for £339";

  This code should change the **Text** property of the **lblBrag** label when the button is clicked. Note the use of **\n** to create a newline.
- Run the program and test your new button. Make sure you understand how it works.

Changing a BackColor property

- Add more code to the **btnShow1_Click()** method to also change the **BackColor** property of **lblBrag** to a different colour. Note: many system defined colours are available:

  Color.Blue, **Color.Green**, etc. etc.

Task 1.3: Buttons

1. Make sure your **two** buttons work properly (**btnQuit** and **btnShow1**) using the method described above
2. Then add a third button with the following properties:
   - Name: **btnShow2**
   - Text: **Best Software**
3. Get this button to change the Text property of the lblBrag control to:

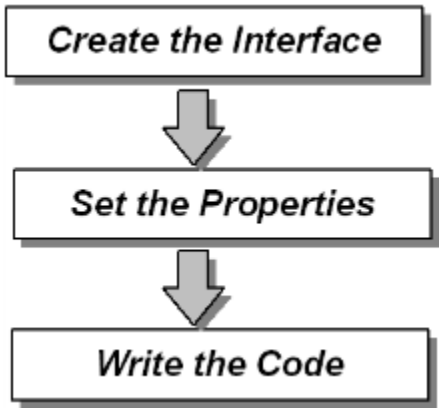
   "C# .NET for Dumbos\n A real bargain at £29.99"
   - You should have a program that looks rather like the one below
4. Now get both **btnShow1** and **btnShow2** buttons to also change the BackColor and ForeColor properties of the two labels **lblBrag** and **lblGreeting** to different and appropriate colours.
5. Run your program to check that all your buttons work correctly

Creating a Data Entry Application

Most of the work so far has involved displaying **output** onto the screen. In this section we shall consider how to **input** data.
We shall create a simple data entry form which will allow the user to enter data into text boxes.
The program will then use this information to display a message.

Creating Windows Applications

There are **three main steps** in creating a Windows application in Visual Studio



We shall be following these steps in creating our Data Entry project

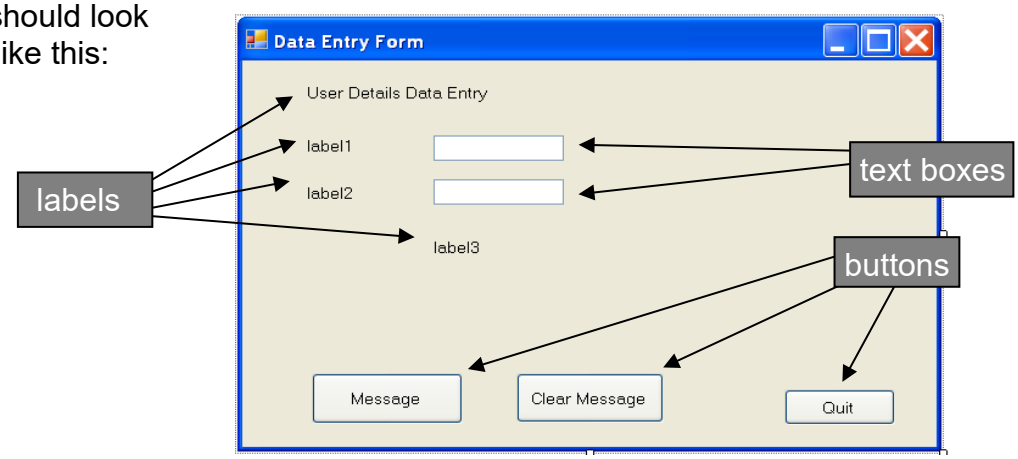
1.4 Task: Creating the Data Entry Interface

The **'interface'** is what you see in each window that opens when the program is running. The **Design Window** (or **Form Window**) is used to create the interface by placing other objects or controls in appropriate positions on the form. In Design mode, the Form is your main working area and it will become the **user interface** for the program.

First open a new C# Windows Application named **DataEntry**
Change the form **Name** to **frmDataEntry** and **Text** to **Data Entry Form**
Now start creating the interface shown on the next page by adding the following controls to the form .....

- First put a **label** at the top of the form, name it **lblMain** and set its Text property to **User Details Data Entry**.
- Add a **button** near the bottom right. Name it **btnQuit** and set its Text property to **Quit**.
- Now add three more **labels** (then name them **lblFname**, **lblSname** and **lblMessage**)
- Also add two **Text Boxes.** These will be used to enter data when the program is run. Name the text boxes **txtFname** and **txtSname**.
- Finally add two **buttons**.
  Name one of them **btnMessage** and give it a Text property **Message**. Name the other **btnClear** and give it a Text property **Clear Message**.
- Adjust the positioning of your objects so they look like the example :

- Your form should look something like this:



Setting the Object Properties

Our next step is to set the properties of the objects .. some of these have been done already, but here is a more complete list. If the item is blank, no action is required.

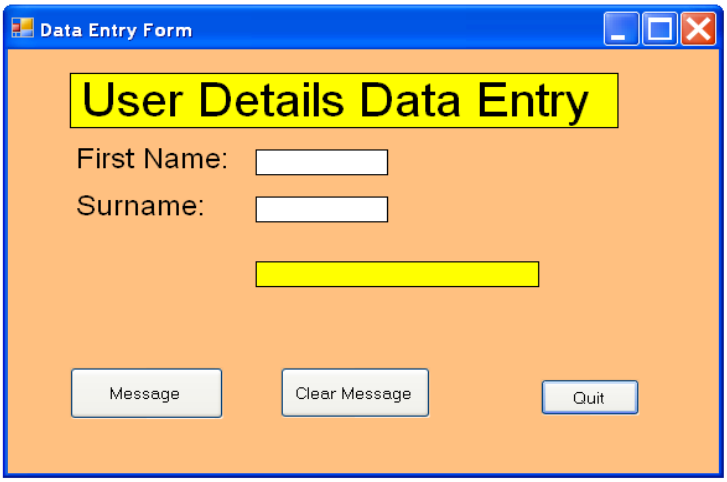
Name	Text	BackColor	BorderStyle	Location (x, y)	Font	
					Name	FontSize
lblMain	User Details Data Entry	Yellow	FixedSingle	50, 20	Arial	24
lblFname	First Name:		None	50, 80	Arial	14
lblSname	Surname:		None	50, 120	Arial	14
lblMessage	[empty]	Yellow	FixedSingle	200, 180	Arial	14
txtFname			FixedSingle	200, 85	Arial	14
txtSname			FixedSingle	200, 125	Arial	14
btnMessage	Message			50, 270		
btnClear	Clear Message			220, 270		
btnQuit	Quit			430,280		

Form Properties

Change some of the properties of the form, as follows:

Name: frmDataEntry **Text:** Data Entry Form **Size:** 580, 400
BackColor: Orange **WindowState:** Normal **FormBorderStyle:** FixedDialog
StartPosition: CenterScreen

- Note that the **Normal** WindowState means that the form is not maximised or minimised.
- the **FixedDialog** BorderStyle means the user is not able to resize the window.
- You should now have a form looking something like this:
- Try running the program with and without a fixed borderstyle (e.g. try Sizeable too)



Adding Code to Quit & Message buttons

- Now add code to the **btnQuit** button to make it end the program (see earlier if needed)
- Now add the code below to the **btnMessage** button's click event and test the program. When the Message button is clicked, the lblMessage **label** picks up any text in the two text boxes and combines them into a caption. Try it.
  Note the use of **+** to join up the text and **\n** for a newline.

```
lblMessage.Text = "Hi " + txtFname.Text + " " +
                  txtSname.Text + "\nHappy Programming" ;
```

Note: you will have to change the size of the lblMessage label to see all of the message .. change its **Size** property to 250, 70 and all should be OK.

Adding Code to the Clear Message Button

We next have to add the code below to the **Clear Message** button's click event:

```
lblMessage.Text = "" ;
```

This ensures that the Text property of the label is cleared. "" is sometimes referred to as the null or empty string.

Running the Program

You should now have a complete program to run. Try it:

- Type a forename and surname in the relevant text boxes.
- Click the Message button to reveal the message
- Clear the Message.
- Click the Quit button.

AutoSizing the Message Label

Sometimes a message is too large to fit in a label or text box. Try running the program again using a long name such as Sukhinder Pargetter .. you may find that you can only see part of the message. How can we cope with different size names? Read on:

- Select the **lblMessage** label
- Change the **AutoSize** property to **True**
- Run the application again using a long name and see the difference.

Using Functions (toUpper, toLower)

Now modify the program using a button to convert the message label into upper case using the **toUpper()** function.

- Add a button to the interface, positioned sensibly and named **btnUpper**
- Change its Text property to **Upper Case**
- Add this code to the click event of the button:

```
lblMessage.Text = lblMessage.Text.toUpper() ;
    // this converts the label text into upper case.
```

- Now add another button so that you have 2 buttons .. one that converts the message to upper case while the other converts it to lower case

Button Access Keys

Often we provide **Access Keys** (sometimes called **SpeedKeys** or **ShortCut Keys**) for buttons, so that a keyboard combination (e.g. Alt + F) can be used instead of the mouse. To provide this feature, we change the **Text** property of a button by adding an ampersand (**&**) in front of the letter to be used as a shortcut e.g...

- change the **Quit** text property to **&Quit** .. this can now be clicked using the key combination **Alt** + **Q** n.b. the Q is shown underlined to indicate this.
- Apply similar changes to convert the other buttons.
- Test the program to see if it works OK

Task 1.4: Data Entry Project

1. Complete all the tasks from 13 to 21 so that your Data Entry program works correctly .. then do the following:
2. Adding a Change Message button
   Add another button called **btnChangeMessage** which does all of the following when you click it:
   - Changes the form's BackColor to Red
     Note: to access the current form object you use the word: **this**
     so to change the form BackColor the code is :
     this.BackColor = …….. ;
   - Also changes the BackColor of both text boxes to **LightBlue**
   - Takes the Text from the lblMessage label and adds the words
     " Get on with it *Brian* " so it appears on a new line
     (using the name that was entered into **txtFname**)
3. Adding a Town box
   Add another textbox and label so the user can enter the town they live in. Change the code so the town entered here is added to the message in a suitable way.
4. Run your program to check that all your buttons work correctly

1.5 Introduction to the Calculator exercise

So far we have learned how to create a C# windows interface with various objects, set the properties of some objects and add some simple code.
We shall now develop these ideas to see how we can perform some simple calculations.
We shall develop a simple **calculator** program which will allow us to:

- enter numbers into two text boxes
- click one of four buttons labelled Add, Subtract, Multiply and Divide
- See the result displayed in a label box.

Object Oriented Design: TOE Charts

One way of starting an object-oriented design is to create a **TOE** chart.

- This is a list of the various **Tasks** to be performed, the corresponding **Object** associated with the task and appropriate **Event** to trigger it.
  (Hence **T**ask-**O**bject-**E**vent chart). The example below should make this clearer:

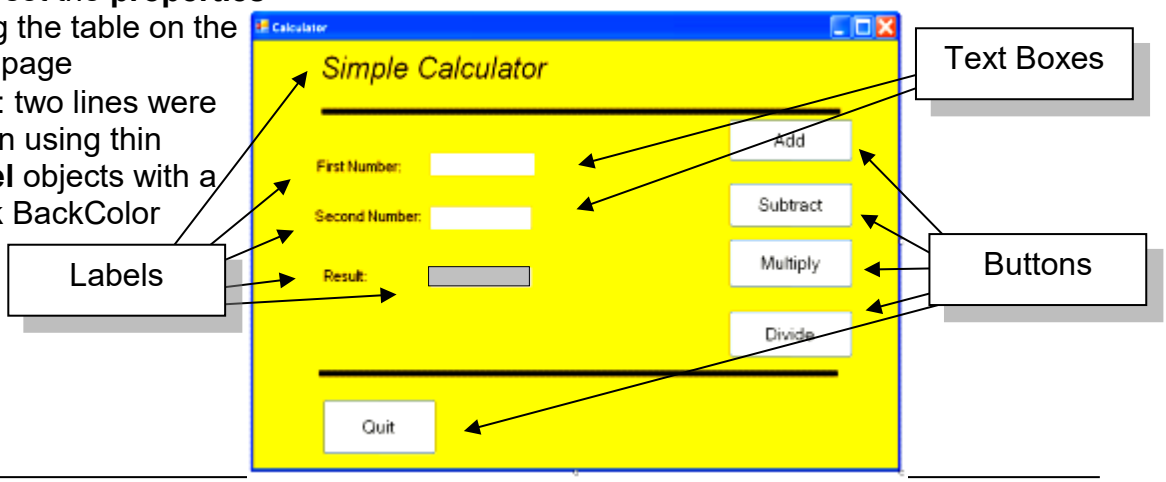
Simple Calculator Program TOE chart			
	Task	**Object**	**Event**
1	Display the Interface	frmCalc	
2	Enter First Number	txtFirstNum	
3	Enter Second Number	txtSecNum	
4	Perform Addition	btnAdd	Click
5	Perform Subtraction	btnSub	Click
6	Perform Multiplication	btnMult	Click
7	Perform Division	btnDiv	Click
8	Display Result	lblResult	
9	End Program	btnQuit	Click

- You can now see that we need 1 form (frmCalc), 2 textboxes, 1 label and 5 buttons.
- We shall also need 3 extra labels for 2, 3 and 8 and one more for a Title.
- Notice that the **click** event is being used with the buttons, but other tasks have no particular event associated with them.

N.B. Notice the **good practice** of using 3-letter prefixes (**frm**, **lbl**, **btn**, **txt**, etc.) in front of object names to clarify the type of object being used.

Calculator Project Interface Design

- Open the existing **Calculator** project and see all the required objects on the form.
- Now set the **properties** using the table on the next page
- Note: two lines were drawn using thin **Panel** objects with a black BackColor



Task: Setting the Properties

Use the following table to set some of the object properties. The object names have already been done for you.

Name	Text	Font Size	AutoSize	BackColor
frmCalc	Calculator			Orange
txtFirstNum		24		
txtSecNum		24		
lblTitle	Simple Calculator	24		
lblFirstNum	First Number:	12		
lblSecNum	Second Number:	12		
lblResultLabel	Result:	12		
lblResult	[none]	24	False	Blue
btnAdd	Add	14	False	
btnSub	Subtract	14	False	
btnMult	Multiply	14	False	
btnDiv	Divide	14	False	
btnQuit	Quit	14	False	

Adding Code to the Calculator Buttons

We now want to get our buttons to work in the correct way.

- You should be able to code the **Quit** button without any help.
- Now program the **Add** button to add together the values of the numbers in the 2 text boxes.
    - Text boxes hold **strings** and these must be converted to **numbers** before adding them together.
    - The answer can then be converted to a string and displayed using the label **lblResult**
- To do this, try the following:

```
double n1, n2, answer;                          // define 3 number variables
n1 = Convert.ToDouble(txtFirstNum.Text);        // convert the first number
n2 = Convert.ToDouble(txtSecNum.Text);          // convert second number
answer = n1 + n2;                               // add them
lblResult.Text = answer.ToString();             // put string into result label
```

- Now test the program: type a couple of numbers into the textboxes and click the **Add** button.

Task 1.5: Calculator

1. Complete the **Calculator** project so that the interface looks good
2. Add code so all buttons work as they should
3. Change the form **StartPosition** to **CenterScreen**
4. Use the **Divide** button to divide a number by zero .. you should find that the result "infinity" is displayed .. now do the following:
    - Change the code so a message "You should NOT divide by ZERO!" is displayed instead if this situation arises, and this calculation is prevented from being performed.

1.6 Adding a Splash Screen to the Project

- Suppose we want to add a new form to our project to display a startup screen when we first run our program.
- This is called a **Splash Screen**
- When we click this form it should load and display the second calculator form that we have just designed. Here goes!

Adding a New Form

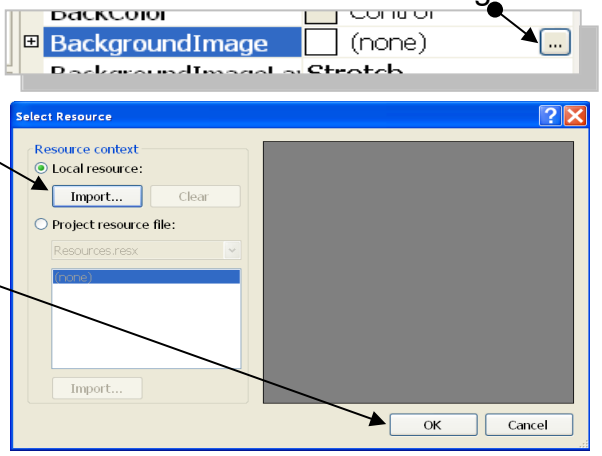
- Select **Project** from the menubar, followed by **Add Windows Form**
- Select the **Windows Form** icon and type a name for the form: **frmSplash**
- Then click the **Add** button

Designing the Form

- Change the **FormBorderStyle** property to **None**
- Add a label to the form and set its Text property to **Click to use Calculator**
- Increase the **Font Size** to 14
- Set the **StartPosition** property to **CenterScreen**

Adding a Background Image

- Select the **BackgroundImage** property and click the little button on the right
- In the **Select Resource** window, Click the **Import** button and find a suitable picture (try CalcHand.jpg, CalcMan.jpg or CalcGirl.jpg) and **Open** it.
- Then click the **OK** button
- Finally find the **BackgroundImageLayout** property and change it to **Stretch**.
- Then adjust the size of your form so your background looks good.



Adding Code to Calculator Form

We want the **frmSplash** form to be displayed before the Calculator form, **frmCalc**. There are several ways to do this. We shall use the **Load** event which takes place as a form loads .. we shall use it to show the **frmSplash** form:

- Open the calculator form (Form1) in Design mode
- Double-click on the form background (not one of the controls)
  - You should find you are in the **Form1_Load()** method
- Add the following code to this event :

```
frmSplash  SScreen = new frmSplash();   // create new SScreen form object
SScreen.Show();                         // show the splash screen
System.Threading.Thread.Sleep(2000);    // pause for 2 seconds
SScreen.Close();                        // close the splash screen
```

- Now run the project and the splash screen should show for 2 seconds before closing, when the Calculator should load up in the normal way.

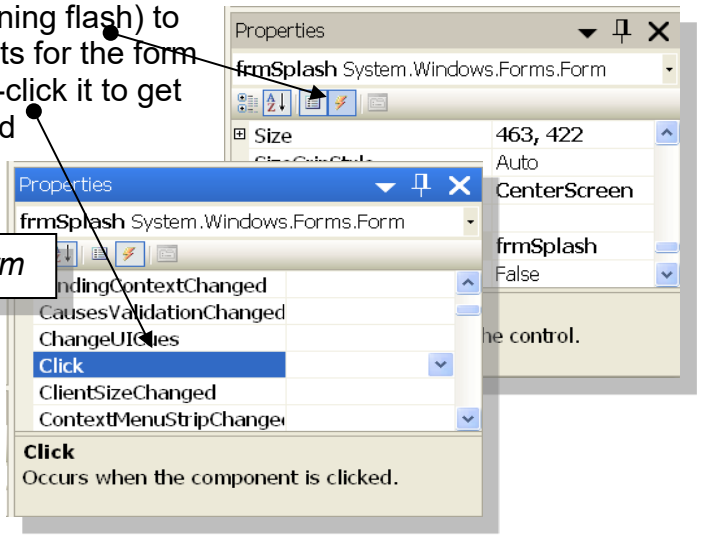
Click Splash Screen to start

- The splash screen at the moment shows for 2 seconds. How can we get it to show until we decide to click it? We need to change a few things.
- Change the code in the **Form1_Load()** method:

```
frmSplash SScreen = new frmSplash();   // create new SScreen form object
SScreen.ShowDialog();                  // show the splash screen
```

- The **ShowDialog()** method is used here instead of **Show()** .. it shows the form until you decide to close it. But how can we close the form?
- Now look at the **frmSplash** form **Design** and examine the **Properties** window
- Click on the **Events** button (lightning flash) to see a list of all the possible events for the form
- Find the **Click** event and double-click it to get to the **frmSplash_Click()** method
- Add code to close down the current form:

```
this.Close() ;   // close current form
```



- Now run the project again and your splash screen should work perfectly!

Task 1.6: Splash Screens

1. Complete the splash screen so that it works with a 2 second delay
2. Then complete the **Click to use Calculator** style of splash screen

Task 1.7: Quit Screen

1. Create a new form called **frmQuit** which has a nice friendly GoodBye message, and a **Click to Finish** message and a suitable background picture.
2. When you click the calculator's **Quit** button it should display your new form **CenterScreen** and then finish when you click it.

Task 1.8: More Calculator Buttons

1. Add another button to your calculator, **Power** that takes the first number and raises it to the power of the second number using the **Math.Pow(-- , --)** function.
   - To test it : **3** to the power of **4** is **81** (3 x 3 x 3 x 3)
     4 to the power of **3** is **64** (4 x 4 x 4)
2. Add another button called **Average** that works out the average of the 2 numbers entered in the text boxes
3. Create another button called **Clear** that clears all values entered in the text boxes.

Unit 1: Independent Study

Task 1.9: Testing the Calculator

1. You should now have a fully functioning calculator program. Before you release this you should test it thoroughly to make sure that all is working correctly.
2. Design a **Test Plan** for the program with inputs, expected outputs, actual outputs and action to be taken. You can use a spreadsheet or Word table for this.
3. In your test plan add at least 2 tests for each of the buttons on your calculator with their inputs and expected outputs.
4. Now test your program using the Test Plan that you just designed.
5. Write your actual results into the Test Plan
6. Add any actions that need to be done as a result of your testing.
7. Put the completed Test Plan into your logbook.

Task 1.10: The Calculator

- Redesign your program, by adding 4 small PictureBoxes to the main form with images representing add, subtract, multiply and divide (use the internet to find some suitable images)
- Rearrange all the items to make a visually satisfying interface
- Real calculator buttons don't normally use the words Add (etc.) .. instead they have an image or icon displayed on them .. can you design your calculator in similar fashion.
- Program the background colour of the main form to change when each type of calculation is performed (e.g. Red for Add, Blue for Subtract etc.)
- Also get the appropriate PictureBox to display as each calculation is perfomed (e.g. the Add button shows only the Add picture)
  .. the lecture notes from last week may help you with this.

Task 1.11: Calculator V 2 (more difficult)

- Try to design a calculator that looks more like the ones you normally see
- The calculator should have 10 buttons labelled 0 to 9 to make a number pad
- There are also 4 buttons for + - / and x
- There is a label to display the number as it is entered using the number pad
- Pressing the + key (for example) stores the first number and clears the display, ready for the next
- Pressing the = button stores the second number, adds them and displays the result
- Get the other buttons to work in similar fashion

Tip

You will need to use variables for the numbers that are available to many different methods .. so you must define these variables after the class definition before the methods start

C# Windows Programming
Unit 2

The Scissors Paper Stone Game (Windows version)

The SPS Game

In this unit we shall develop a Windows version of the Scissors, Paper, Stone Game. This should also teach us quite a lot about programming using the C# .Net environment.

- The game as you can see uses 2 **PictureBox** controls to display pictures for the choices made by the user and the computer
- It also uses 3 **RadioButtons** to allow the user to make a choice
- When the **Play** button is clicked, the computer makes its choice and the result is displayed using a **Label**
- Notice that we are using 3 **GroupBox** controls to group the various parts together.

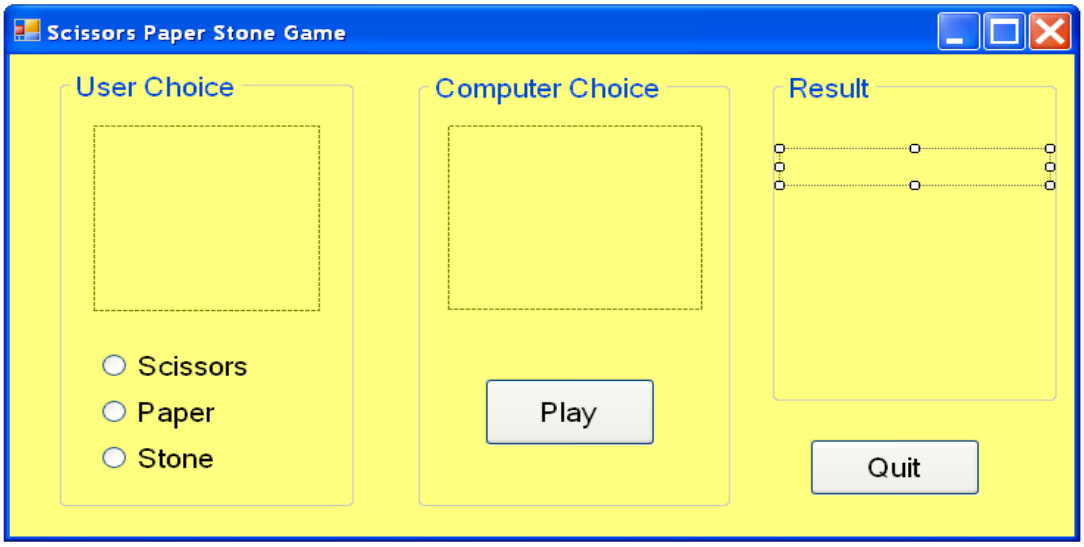
2.1 Designing the Form

- Create a New Project (Windows Application) called **SPS Windows Game**
  - Change the form's **Text** property to **Scissors Paper Stone Game**
  - Change the form **BackColor** property to something suitable
- Put 3 **GroupBox** Controls on the Form
  - change the 3 **Text** properties (as shown above) to:
    User Choice and **Computer Choice** and **Result**
  - Change the **Font** Size properties of each GroupBox to **12**

1. Add 3 **RadioButton** controls to the first GroupBox
   - change their Name properties to rbnScissors, rbnPaper and rbnStone
   - change their **Text** properties to **Scissors** and **Paper** and **Stone** as shown
2. Put 2 **PictureBox** controls into 2 of the GroupBoxes
   - change their Name properties to pbxUser and pbxComputer
   - change their **SizeMode** properties to **StretchImage** .. this should ensure that each picture fits into its PictureBox
3. Add 2 **buttons** as shown
   - change the Name properties to btnPlay and btnQuit
   - change their **Text** properties to **Play** and **Quit** as shown.
4. Add a **label** to the Result GroupBox
   - change its Name property to lblResult
   - change its **AutoSize** property to **False**

Task 2.1: SPS Game Form Design

- Complete the tasks on the previous page and adjust the sizes etc. to create a form that looks something like the one shown below



2.2 Programming the RadioButtons

- We want to get each radiobutton to put a different image into the **pbxUser** picturebox when it is checked.
- You should find 3 images that you can use on the L: drive (**Scissors.jpg**, **Paper.jpg**, **Stone.jpg**). They are in an **Images** folder in **C# Bsc Windows Progs**
  - copy these 3 images into the **Debug** folder of your project (in the **bin** folder)
  - this will make them easier for the program to locate
- Now double-click the **Scissors** button to reach the Code Window
  - You should now be in the **rbnScissors_CheckedChanged()** method
  - Add some code to load the correct image file into the **pbxUser** picturebox when the radiobutton is checked (see below)

```
private  void  rbnScissors_CheckedChanged(object sender, EventArgs e)
{
        pbxUser.Image = Image.FromFile("Scissors.jpg");

}
```

Task 2.2: SPS Game RadioButtons

1. Complete the above tasks for the Scissors radiobutton
2. Add similar code so that each radiobutton causes the appropriate image to be displayed
3. Run the program and check that each button displays the correct image

2.3 Programming the Play Button

- We shall now get the **Play** button to choose a number at random from 1 to 3 and use this to select an image for the **pbxComputer** picturebox
- We shall first need to define 2 integer variables .. one for the computer choice and one for the user choice. We shall put these in the Code Window near the top of the form so they are available throughout the form.
    - Define **userChoice** and **compChoice** as shown below:

```
namespace  SPS_Windows_Game
{
      public  partial  class Form1 : Form
      {
            int  compChoice, userChoice;
```

- Now double-click the **Play** button to enter its Click method.
    - Add code here to pick a Random number for compChoice :

```
Random  r  = new Random();    // create a new random object r
compChoice = r.Next(3) + 1;   // use r to pick a number from 1 to 3
```

- Underneath this you can now add code to put the correct image into the **pbxComputer** picturebox:

```
if (compChoice == 1)
{
      pbxComputer.Image = Image.FromFile("Scissors.jpg");
}
```

Task 2.3: SPS Game Play Button

1. Complete the above tasks for the Play button
2. Add more code so that a different image is selected for each of the 3 computer choices
3. Run the program and check that the play button works in a suitable random fashion and displays the appropriate images

2.4 Win, Lose or Draw?

- We now need to compare the user and computer choices and display the result in the label **lblResult**. First of all we need to set values for the **userChoice** variable:
  - Double-click each radio button in turn and put
    userChoice = 1; etc.
    into each method
- Now create a new method called **checkResult()**
  - In this method you should set up **lblResult** according to the values of **userChoice** and **compChoice** e.g.

```
if (compChoice == userChoice)
{
        lblResult.Text = "It's a DRAW!"
}
```

- Now you need to call this method by putting the line
  checkResult(); *// call the checkResult() method*
  at the end of the btnPlay_Click() method

Task 2.4: Game Win, Lose or Draw

1. Create the new checkResult() method as shown above
2. Complete this method to display an appropriate message for all possible combinations
   Computer WINS!!
   You WIN!!
   It's a DRAW
3. Run the program and **test** it for all possible user and computer selections

Game Rules Reminder

- Scissors beat Paper
- Stone beats Scissors
- Paper beats Stone
- Other combinations are a Draw

Task 2.5: Game ON

1. Get the program to use appropriately flashy **BackColor** and **ForeColor** settings for the label **lblResult** .. this should happen only when the **You Win!!** message is displayed!
2. Find 2 other appropriate images and display them in the 2 pictureboxes when the form first loads up (try sps1.jpg, sps2.jpg, sps3.jpg)
   - Put the images in the **Debug** folder .. where you put the others
   - N.B. if one button always loads up selected .. set **TabStop** to false
3. In case you forgot .. program the **Quit** button to work too.

Unit 2: Independent Study

Task 2.6: Game Scoring

1. Add code to define 2 more variables: **userScore** and **compScore** and initialise these to zero
2. Add 2 more labels to the Result section of your form design .. **lblUserScore** and **lblComputerScore**
3. Get the program to display both scores in the appropriate labels
   - Note that you will need to convert your numbers to strings first

   e.g. **lblUserScore.Text = userScore.ToString();**
4. Get the scores to increase properly (2 points for a Win, 1 point for a Draw)
5. Add more labels with Text values "User Score" and "Computer Score" to produce a display similar to the one shown below
6. Test your program thoroughly

Task 2.7: Splash Screen

- Create a nice Splash Screen for your game which loads up before the main game form

Task 2.8: Finish Screens

- Create 3 different Game Over screens for when a score reaches 20.
- An appropriate screen with message and picture should appear for winning, losing or drawing the game:
  - You can use **ManYesYes.gif** and **ManNoNo.gif**
  - A draw screen could show **ClappingHands.gif**
  - You can find these in the **images** folder of **BSc C# Windows Progs**

C# Windows Programming
Unit 3

MessageBoxes, Timers and other .NET controls

Some Other .NET Objects (or controls)

There are many other types of objects available to the Visual C# programmer as well as forms, buttons, text boxes, labels, etc. We shall now take a look at some of them.

3.1 Radio Buttons Revisited

Radio buttons are useful for allowing users to make choices on a form.
In this Currency Converter example we use 3 radio buttons
(note that they are mutually exclusive .. clicking one button deselects the others)

- Open the existing project called **Currency Converter**. You will see:
  - a text box named **txtUKMoney**
  - a label named **lblConverted** to hold the converted UK money
  - a **Panel** that acts as a Container for some radio buttons
  - 3 radio buttons, named **rbnDollars**, **rbnEuros** and **rbnRupees**.
  - Some other labels and a **Quit** button
- Run the project and try selecting different radio buttons.

Adding Code to the Radio Buttons

We want to get the program to take UK money from the **txtUKMoney** textbox, multiply by the rate and display the result in **lblConverted**. This needs to be done each time we check one of the radio buttons.

- Double-click the Dollars radiobutton and add some code to the **rbnDollars_CheckedChanged()** method:

```
double amount;                                  // variable for calculation
amount = Convert.ToDouble(txtUKMoney.Text) * 1.8;   // 1.8 dollars to the pound
lblConverted.Text = amount.ToString() + " Dollars";   // show the result
```

Task 3.1: Converter

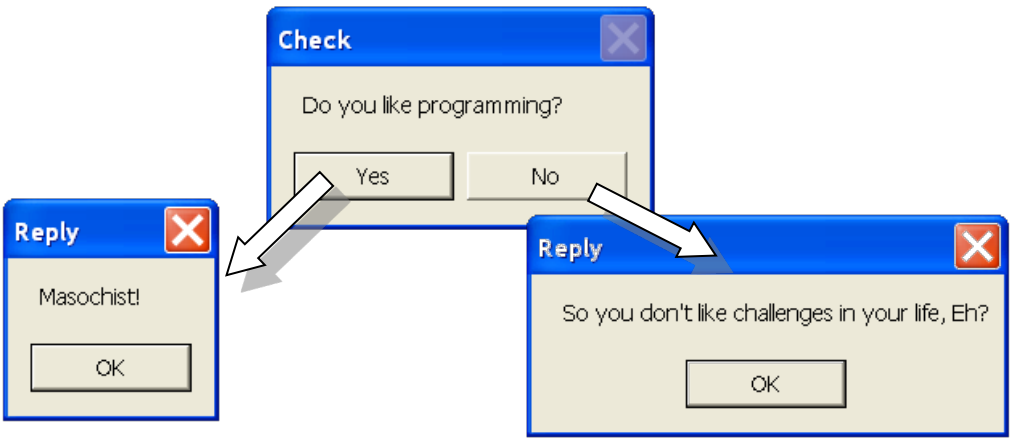
1. Complete the above task to convert UK pounds to dollars
2. Add similar code to the other radio buttons .. note that there are about 1.4 euros to the pound and about 80 rupees to the pound (revise if you wish)
3. Complete the **Converter** project so that the interface looks good and all radio buttons etc. work as they should and the form **StartPosition** is **CenterScreen**
4. Add a suitable background colour to your form
5. Improve your **Quit** button so that it shows a MessageBox which asks:
   Are You Sure?
   and gives you the chance to reply **Yes** or **No** with appropriate actions

Responding to MessageBoxes

There are different kinds of MessageBoxes. The simplest just have an **OK** button (see above) but others can have **Yes** and **No** buttons, **Yes**, **No** and **Cancel** buttons, etc. When the MessageBox has more than one button, it is possible to detect which one has been clicked. The following code below demonstrates an example of responding to choosing Yes from a MessageBox:

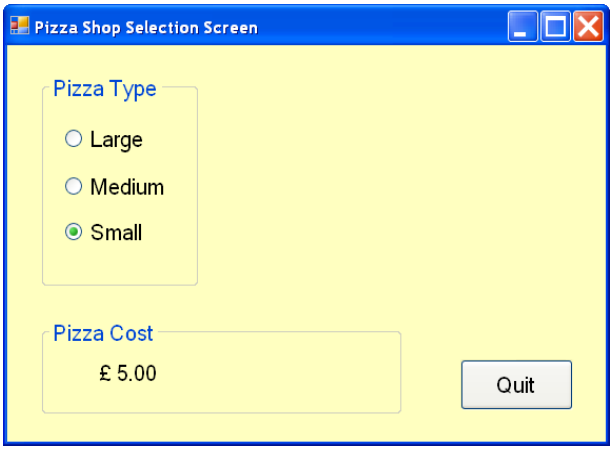
```
DialogResult response;              // define a variable for the result

response = MessageBox.Show("Do you like programming?",  "Check",
                                                    MessageBoxButtons.YesNo);
if (response == DialogResult.Yes)
{
      MessageBox.Show("Masochist!", "Reply");
}
```



3.2 Check Boxes

Check boxes are very similar to radiobuttons, **but** they allow multiple selections to be made .. and each box displays a tick when selected.



- Open the existing **PizzaShop** application.
- Run it to see how the radio buttons work.
- Also look at the code to see how the totalcost is worked out and displayed

Adding Check Boxes to the Form

- First of all add another **GroupBox** to the Form and change its Text property to **Toppings** as shown here
- Then put 4 **CheckBoxes** into the GroupBox and name them **cbx1**, **cbx2**, **cbx3** and **cbx4**
- Choose any 4 pizza toppings and change checkbox **Text** properties to display these, as shown.

Adding Code to the Check Boxes

- Each Topping costs an extra £2.50
- Double-click the first check box and add some code to the **cbx1_CheckedChanged()** method

```
if (cbx1.Checked)          // has this box been checked?
{
      // code to add 2.50 to the topping cost
       // code to display the new totalcost in the label lblCost
}
```

Task 3.2: Check Boxes

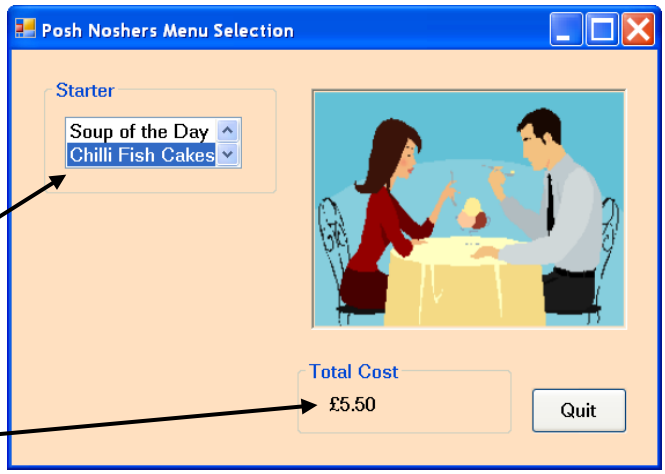
1. Complete the Pizza Shop application as described above.
2. Add code to all 4 of the CheckBoxes
3. Test the program to see that everything works correctly and that each topping adds an extra £2.50 to the cost
4. If you deselect a topping can you get the cost to reduce by £2.50?

5. **Complete a Test Plan for the PizzaShop program**

3.3 List Boxes

The list box allows users to select from a list of choices, many more than are possible with radio buttons. A list box is basically an array of strings. Scroll bars are provided for long lists.

- Open the existing Windows application called **PoshNosh**
- This has a PicureBox and a Quit button
- It also has 2 GroupBox controls: **Starter** and **Total Cost**
- There is a **ListBox** called **lstStarter** with 7 items on the list



- **Run** the program and select different items from the list .. there is a label called **lblTotal** that displays the total cost

Adding another ListBox

- First add another **GroupBox** and change its **Text** property to **Main Course**
- Add a **ListBox** control to the GroupBox, name it **lstMain** and change its **Sorted** property to **True** so the list will be in alphabetical order.
- Select the **Items** property and add 4 main course items (one per line): e.g. Steak and Chips, Fish and Chips, Vegetable Curry, Chicken Lasagne
- Now double-click the ListBox and add some code to the **lstMain_SelectedIndexChanged()** method … for example :

```
switch (lstMain.Text)                    // what is the chosen item from lstMain  ?
{
        case "Steak and Chips"    :  maincoursecost = 12.50;   break;
        case "Fish and Chips"     :  maincoursecost = 7.50;    break;
        case "Vegetable Curry"    :  maincoursecost = 8.00;    break;
        case "Chicken Lasagne"    :  maincoursecost = 7.25;    break;
}

totalcost = startercost + maincoursecost + dessertcost;        // calculate total
lblTotal.Text = "£" + totalcost.ToString("0.00");              // display total
```

Task 3.3: List Boxes

1. Complete the ListBox exercise above so that Starter and Main Course selections from the ListBoxes all work correctly .. with correct totals
2. Examine the code and make sure you understand how it all works
3. Try altering the height of a ListBox and see how this affects the display.

3.4 Combo Boxes

The drop-down Combo Box is like a cross between a ListBox and a TextBox because you can add new items and also display Text

- First add a new **GroupBox** to the previous form and change its Text property to **Dessert**
- Add a ComboBox to the GroupBox and give it a name: **cbxDessert**
- Change its Text property to **Dessert Choices**
- Now add a selection of your favourite desserts to the **Items** property.
- Run the program and see your selections revealed when you click the drop-down arrow:

- Now in design mode, double-click the combobox and add code to the **cbxDessert_SelectedIndexChanged()** method so that a dessert cost is added to the total and displayed correctly (use the previous listbox code to help you)

Task 3.4: Combo Boxes

1. Complete the **Posh Nosh** project so that all totals work correctly using two ListBoxes and a ComboBox to select the 3 courses
2. Design and use a Test Plan for the Posh Nosh project

3.5 Monkey Bash: Detecting Mouse Clicks

- Open the existing **Monkey** project and look at the form .. it has a PictureBox with an image of a cute little monkey.
- There are 3 buttons at the bottom of the form that don't do anything yet.
- Also notice a **timer** control has been added (it shows as **timer1** under the form)
- The program doesn't do much at the moment but we shall improve it to develop a 'Bash the Cute Little Monkey' game!

The timer control

- The **timer** is a control that can be set to **Tick** at a certain rate. It has a **Tick()** method that can be used to run some code each time it ticks.
- Double-click on the timer control to see the code that has already been added to its **Tick()** method. Each time the timer ticks its sets up new random values for the position of the pictureBox.

```
private  void  timer1_Tick(object sender, EventArgs e)
{
        x = rand.Next(this.Width - 100);        // pick a random x
        y = rand.Next(this.Height - 100);       // pick a random y
        pbxPicture.Left = x;                    // set picture to these new values
        pbxPicture.Top = y;
        Refresh();                              // redraw in the new position
}
```

- note: **this.Width** and **this.Height** are the width and height of the form. These are being used as a guide for the random x and y values)
- Also note that **x**, **y** and **rand** have been defined earlier in the program :

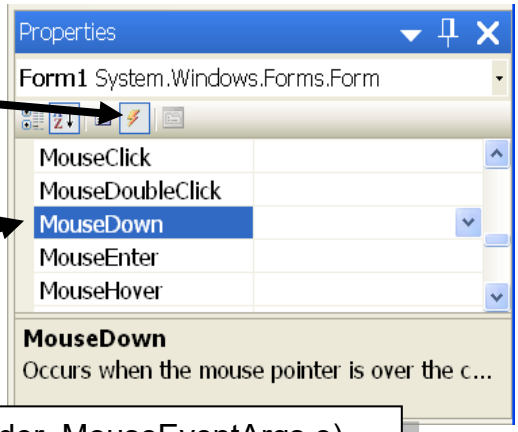
```
int x, y;
Random rand = new Random();     // create new Random object called rand
```

Task 3.5: Monkey Bash 1

1. Double-click the Start button and enter code into its click() method:
   timer1.enabled = true; *// enable the timer control*
2. Double-click the Stop button and enter code into its click() method:
   timer1.enabled = false; *// disable the timer control*
3. Now run the program and use these 2 buttons to start and stop the timer
4. Get the **Quit** button to work too
5. Now add code to display a MessageBox saying **"Ouch! You HIT Me!"** when the Monkey picture is clicked with the mouse.
6. Also get the picture to change to a frowning monkey (monkey2.gif) when it is clicked and then back again
7. Change the **Cursor** property of the form to a **hand**
8. Now play the game.
   - Also try changing the **Interval** property of **timer1** to 2000 and then to 500 ..notice the difference when playing the game and adjust your settings to find a speed that is most suitable

3.6 More Mouse Events

- There are lots of mouse events as well as the MouseClick event. For example, the **MouseDown** event can be used to find out where the mouse is on the form …
- First start a new Windows application and call it **MouseEvents**
- The default event for a form is the Load event so how do we find the MouseDown event?
  - Click on the form to select it
  - Then click the **Events** button in the Properties window.
  - This will show you a list of all the form Events .. scroll down to find the MouseDown event.



- Now double-click **MouseDown** to enter code into the code window for the MouseDown method :

```
private  void  Form1_MouseDown(object sender, MouseEventArgs e)
{
        MessageBox.Show ("The mouse X position is " + e.X);
}
```

- Run the program and press the mouse anywhere on the form to see the MessageBox

Task 3.6: Monkey Bash 2

1. Modify the previous Monkey Bash program so that you get a second message:
   You MISSED! if you click the Form instead of the monkey
   (you will need to use the form's click method for this)
2. A smiley monkey should also be displayed when this happens
3. Add a small **GroupBox** to the form with a **label** inside it.
4. Improve the game by getting the label to display the number of monkey hits
5. Display the number of misses in the GroupBox too
6. Modify the game so the picture never moves into the area of buttons or labels

3.7 MyPad (a simple version of Notepad)

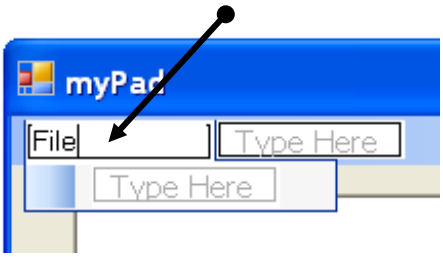
We shall now build a simple notepad clone which will use menus, RichTextboxes, common dialog boxes and also our Scroll bars. Note: You may need to do this project away from the network on a memory stick or another drive to get it to work properly.

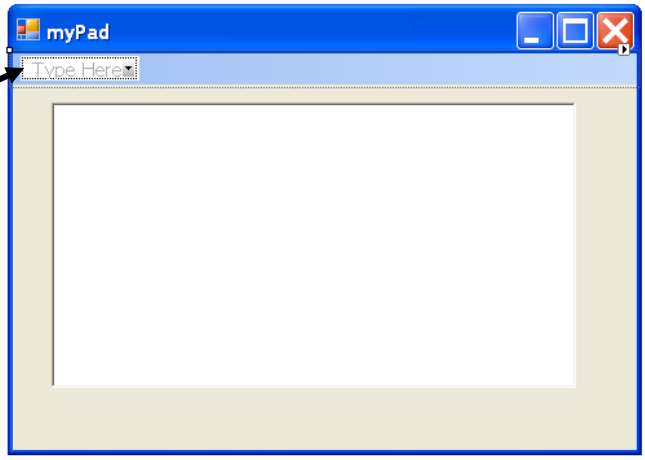
RichTextbox

- Start a new Windows project called **myPad**.
- Add a **RichTextBox** to the project from the ToolBox and call it **txtMain**
  - alter its size to fit most of the form
  - this can be used to enter lots of text which can be formatted, cut and paste etc.
  - Now add a MenuStrip to the project (see next page)

Menus

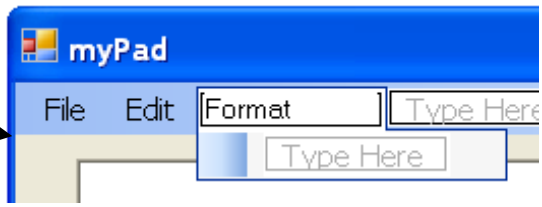
- Add a **MenuStrip** from the ToolBox
- This will appear at the top of the form (and also underneath)
- Where it says **Type Here** type in a menu item **File**



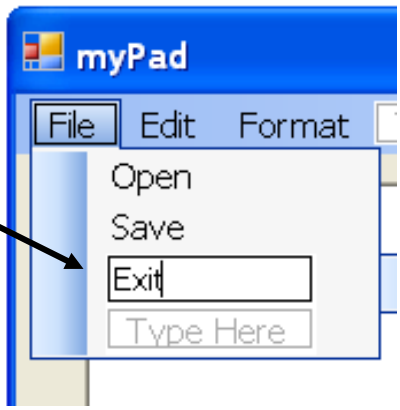


Adding Menu Items

- To the right of **File**, add 2 more items, **Edit** and **Format** (as shown here)



- Then select **File** again and add some drop-down items below File :
  Open, **Save** and **Exit**
- Then select **Edit** and add some more drop-down items below this:
  Cut, **Copy** and **Paste** and **Undo**
- Finally select **Format** and add two drop-down items **Font** and **Background Colour**



here:

Adding Shortcuts

- All the menu items can now be given shortcuts. Here is one example:
- Select **Cut** from the **Edit** menu
- Go to its **Properties** and find **ShortcutKeys**

- Select the down arrow and choose **Ctrl** and **X**
- Use a similar method to set shortcut keys for all your menu items.

Task 3.7: MyPad 1

1. Set up the myPad project with **RichTextBox** and **MenuStrip** as above
2. Run the project and check that the menu items can be selected (with no functionality as yet!) and you can type into the Text box
3. Test the shortcut keys and see that they work (no functionality yet)

3.8 Adding Functionality to the Menu

We want our menu to actually do something useful. We need to add code to the items.
Note: It may be necessary to put the MyPad folder onto a memory stick or another drive.

Programming the Edit Menu

- Select the **Cut** item from the edit menu and double-click this
- Inside the **cutToolStripMenuItem_Click()** method add some code to cut text from your RichTextBox (named txtMain). This is easy .. it is just :

```
txtMain.Cut();
```

- Now in a similar way program the **Copy** and **Paste** and **Undo** menu items
- Run the program and test you Edit Menu to see that it all works

- You should also be able to program the **Exit** item in the **File** menu

Task 3.8: MyPad 2

- Program the **Edit** menu to work correctly

3.9 The File Menu and Common Dialogs

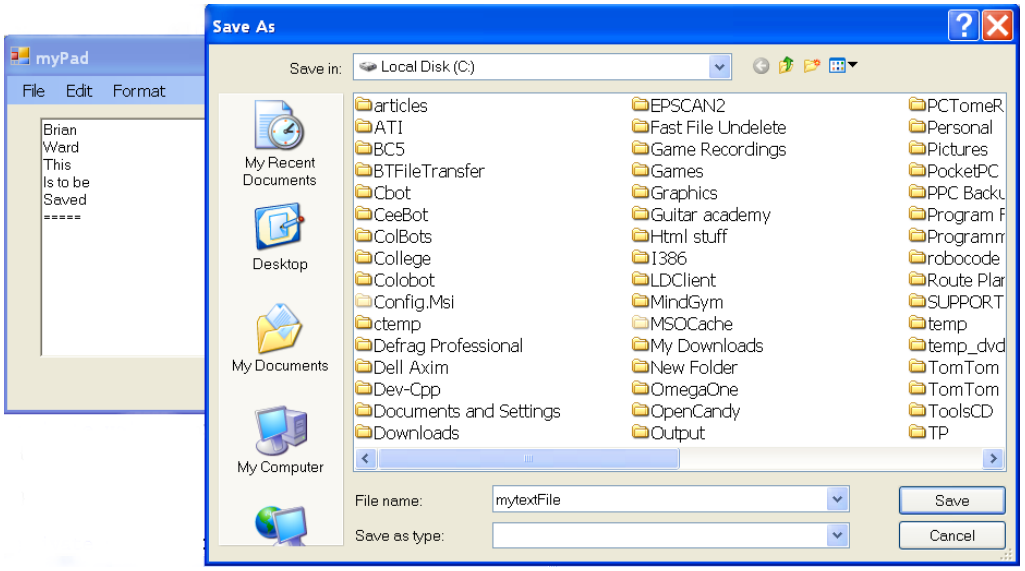
The other menu items are slightly more difficult to program. We shall be using some quite powerful controls called **Common Dialog Boxes** in this section.

Programming the Save Menu item

- Find the **SaveFileDialog** control in the Toolbox and add this to the Project. It will appear underneath the form.
- Change the name to **sfd**
- Now select **Save** from the **File** menu and double-click this
- Inside the **saveToolStripMenuItem_Click()** method add code to use the **sfd** dialog that you just added to the project:

```
sfd.ShowDialog();                    // show dialog to choose a file name
txtMain.SaveFile( sfd.FileName );    // save text using this file name
```

- Now run the project again, type a few sentences into the txtMain and then choose the **Save** option .. you should get a dialog that you will recognise:



- Choose a suitable name for your file and hit the **Save** button.

The Open Menu Item

- Select the **OpenFileDialog** control in the Toolbox and add this to the Project. Change its Name to **ofd** for short
- Program the **Open** menu item (hint: try using **txtMain.LoadFile(ofd.FileName);** and test it by opening the file you just saved.

Task 3.9: MyPad 3

- Program the **File** menu to work correctly. If necessary you may have to copy MyPad onto a memory stick or other drive and run it from there.

3.10 The Format Menu and Common Dialogs

We shall be use some more **Common Dialog Boxes** in this section.

The Font Menu Item

- Add the **FontDialog** control to the Project. Call it **fd** for short.
- Add code to the **Font** item to show the FontDialog and then select the appropriate font to use in txtMain:

```
fd.ShowDialog();
txtMain.Font = fd.Font;
```

- Test it to see if it works OK

The Background Colour Menu Item

- This time add the **ColorDialog** control to the Project. Call it **cd** for short.
- Can you see how to program the **Background Colour** item to first show the dialog and then use the **Color** selected as the **BackColor** for **txtMain** (see the code above for clues)

Foreground Colour Menu Item

- Add a third item to the **Format** menu called **Foreground Colour**
- Program this so it changes the **ForeColor** used by **txtMain** and test it

Task 3.10: MyPad 4

- Program the **Format** menu to work correctly

Unit 3: Independent Study

Task 3.11: Hit the Target !

1. Copy the whole Monkey folder and modify it so that a moving Target is displayed. You will find **Target.gif** in the normal Images folder on the L: drive
2. Make the Size of the PictureBox equal to 100, 100 (so its centre will be 50,50)
3. Use labels to display a score and message that depend on your distance from the target centre (see note below)
4. Note: **e.X** and **e.Y** can be used here in the formula for a right-angled triangle
   - distance = square root of ($(\mathbf{e.X} - 50)^2 + (\mathbf{e.Y} - 50)^2$) .. if the target centre is 50, 50
   - This gives us the mouse distance from the centre
   - Math.Pow() and Math.Sqrt() can be used in the formula

Task 3.12: More Monkeys

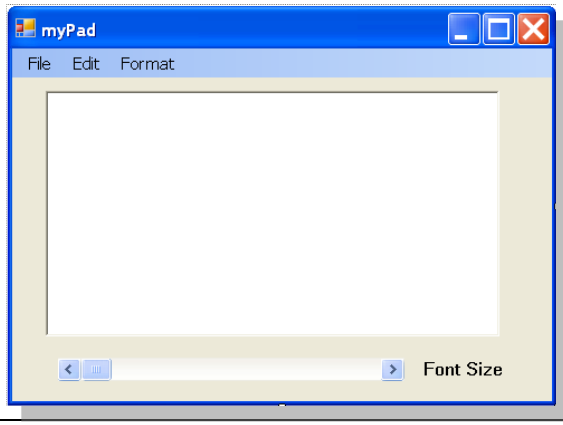
2. Add 4 more timers and Monkeys to your Monkey project
3. Set different intervals for the timers and create an interesting variation of the game

Task 3.13: MyPad Scrollbar

- Continue the myPad project, and add a scrollbar which changes the font size of selected text, as below

Adding a Scroll Bar

Add a horizontal ScrollBar. Also add a **Font Size** label as shown here.



- Change the **Minimum** property of Scrollbar to **8** and **Maximum** to **36**
- Now add code to change the Font Size of the txtMain font
  e.g. this example changes the size to 20:

```
Font  f = new  Font(txtMain.Font.Name, 20, FontStyle.Regular);  // change size to 20
txtMain.SelectionFont = f;                                      // use this font in text box
```

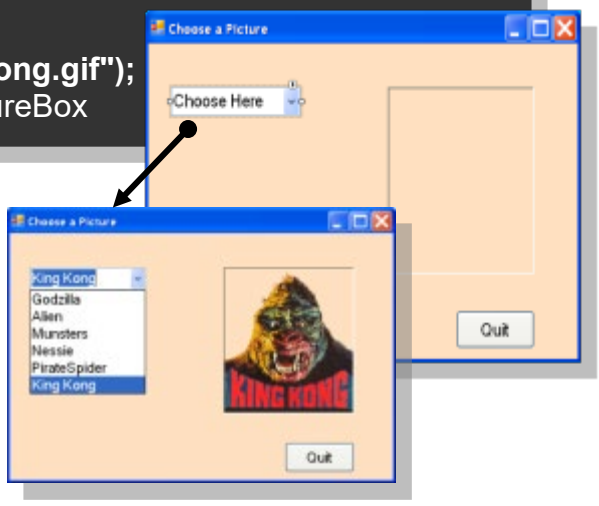
- Modify this code so it uses the value generated by the Scroll bar as the size

Task 3.14: Choose a Picture

1. Create a **new** Windows application and add to the form:
   a. a ComboBox (name: cbxChoice)
   b. a PictureBox (name: pbxChoice)
   c. a Quit button (name: btnQuit)
2. Look at the **monsters** folder in the **images** folder and choose at least 6 monster images .. copy them to your project **Debug** folder (in **bin**)
3. Look at the combobox **Items** property and add some item names .. the names of your selected monster pictures. Also change the **Text** property.
4. Now add code so that when a particular Text is chosen a corresponding image is displayed in the PictureBox

Note:

pbxChoice.Image = Image.FromFile("kong.gif");
puts an image into the **pbxChoice** PictureBox



C# Windows Programming
Unit 4

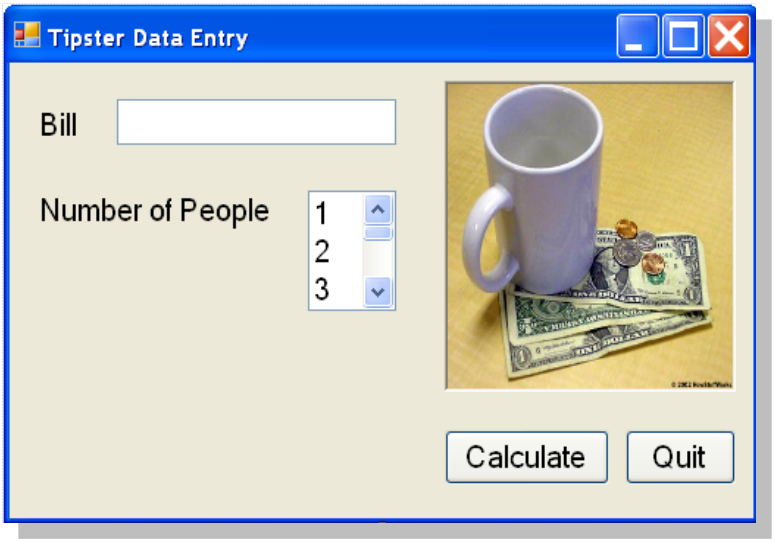
Multi-Form Projects

Introduction

- Our Windows projects so far have used just one form to deliver most of the action.
- When we added forms to a project, this was mostly for cosmetic reasons.
  - for example we used a Splash Screen to introduce our Calculator project.
- In this section we shall try to make better use of forms and this will require us to be able to pass information from one form to another.
- We shall create a simple project to allow us to calculate payments in a restaurant

4.1 Starting the Tipster Project

- Start a new Windows project called **Tipster**
  This will first allow you to enter a bill total and work out the cost per person.
- Design the main data entry form so that it looks something like the one shown here, with a textbox, listbox, picturebox, 2 buttons and some labels.
- The listbox should allow you to select a number from 1 to 10

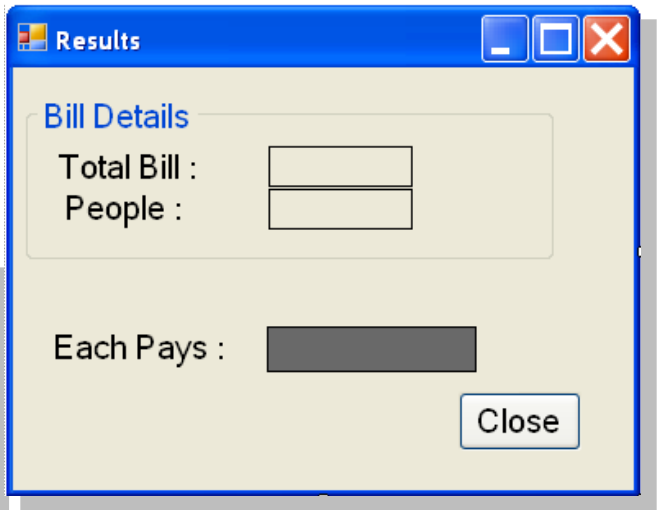


Task 4.1: Tipster 1

- Create the Data Entry Form for the Tipster project as shown above

4.2 Adding a Form to Display Results

- Add a new Results form to your Project
- Design this form with labels, a button and a GroupBox so that it looks something like the one shown here.
- At the moment there will be no numbers displayed



Task 4.2: Tipster 2

- Create the Results Form for the Tipster project as shown here
- Get this to show up when the **Calculate** is clicked and close when the **Close** button is clicked
- For help see the next page note

How to Show a new Form (also see Unit 1)

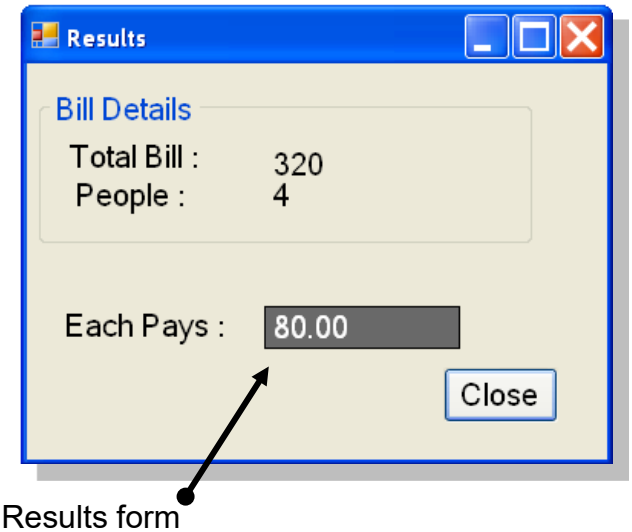
- First you need to create a new form Object using the form's Class name e.g. if the form is named frmResults then this could be done like this:
  frmResults newForm = new frmResults();
- Then you can show your new form object like this:
  newForm.Show();
- This code is added to a button's Click method so it runs when the button is clicked
- Note: The current form can be closed by using **this.Close();**

4.3 Moving Data between Forms

Now we want to be able to move data from our main form to the display form .. see the example here.

Important Note

- By default when you add objects to a form they are set to **Private**
- This means that they are not available to other forms.
- To make them available, we need to set their **Modifiers** property to **Public**



- Select one of the data labels on the Results form
- In the **Properties** window, find the **Modifiers** property and change this to **Public**
- Repeat this for the two other data labels.
- Now program the **Calculate** button to also copy the Bill amount from the textbox to the label on the new form. For example:
  newForm.lblTotal.Text = txtBill.Text;
- Add more code to also copy the people number from the listbox to the new form.
- Finally add code to display the amount each person is to pay .. this is done by dividing the total bill by the number of people. Note that you will have to convert the Text to a number (double) before calculating. Here is some information from chapter 2 .. use this to help you add the necessary code :

```
double  n1, n2, answer;                    // define 3 number variables
n1 = Convert.ToDouble(txtFirstNum.Text);   // convert the first number
n2 = Convert.ToDouble(txtSecNum.Text);     // convert second number
answer = n1/n2;                            // divide them
lblResult.Text = answer.ToString();        // put string into result label
```

Task 4.3: Tipster 3

- Code the **Calculate** button so all Results are displayed correctly on the form
- Program the **Quit** and **Close** buttons too

4.4 Using try/catch .. to trap errors

It is possible that someone might type something daft like 'smsm' by mistake for the bill amount. We can trap errors or exceptions using the very useful try .. catch .. instruction.

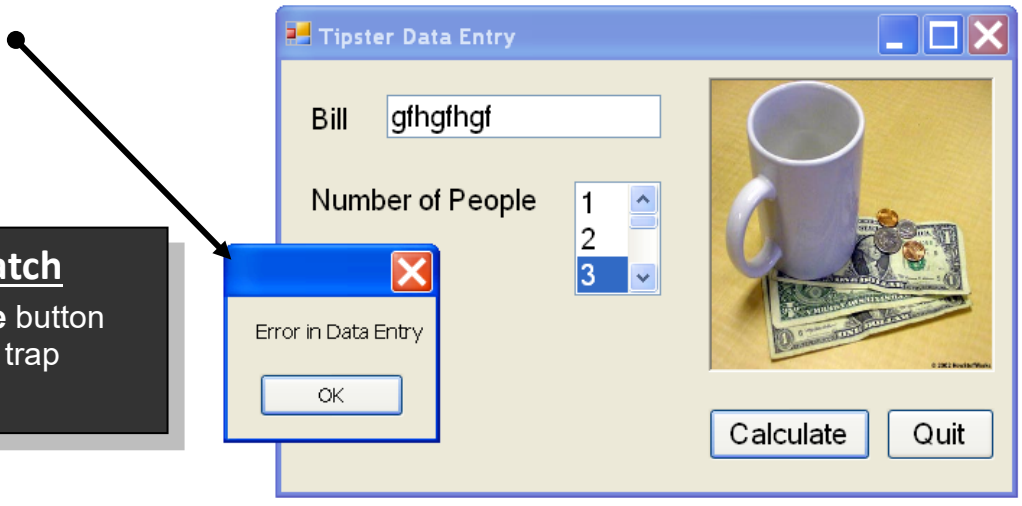
- The **try .. catch ..** instruction has many different versions but we shall just consider the simplest one here.
- The basic structure is this:

```
try
{
        // put instructions here that you want to check
}
catch
{
        // display an error message here if an exception occurs
}
```

- Here is an example:

```
try
{
        double n1, n2, answer;
        n1 = Convert.ToDouble(txtBill.Text);
           // the rest of the calculations also go here
}
catch
{
        MessageBox.Show("Error in Data Entry");
}
```

- The **catch** block is only done if there is an error in the **try** block
- Here is the possible result:



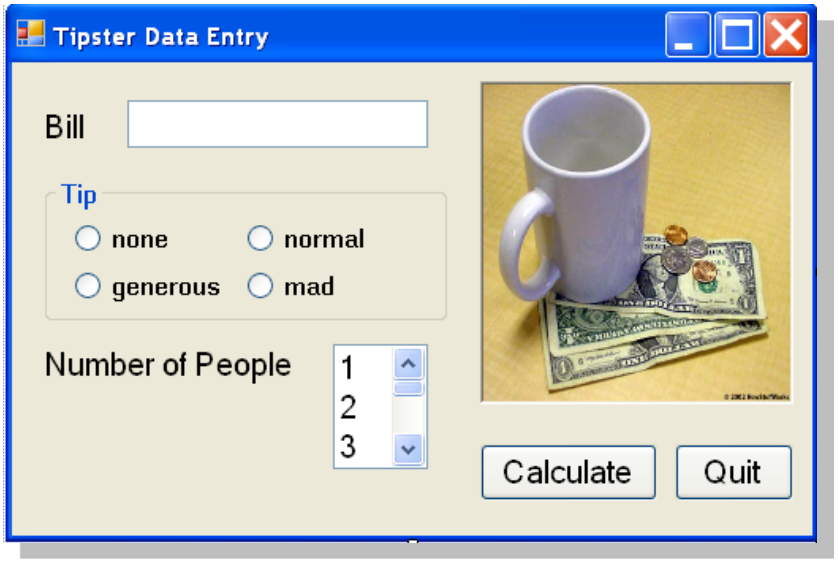
Task 4.4: Using try/catch

- Program the **Calculate** button using **try** and **catch** to trap errors like this

4.5 Tipping

Now add a GroupBox and 4 radiobuttons as shown to allow the user to select a tip to be added to the Bill.

- None is just that .. 0 (stingy or what?)
- Normal is 10% of the bill
- Generous is 15% of the bill
- Mad is 20% of the bill



Task 4.5: Tipping

- Complete the form to allow tipping as shown above
- Add code so that the appropriate tip is added to the bill (you may want to look at chapter 5 for help)
- Add another **Bill + Tip** label to the Results form and ensure that this new total is now used to calculate the amount per person

Task 4.6: Test Plan

- Design a Test Plan for your Tipster project and use it to test your final product

Unit 4: Independent Study

Bucks New University is hosting a conference called "Education in the 21st Century". Teachers from three local schools have been invited to attend.

- Attendees can have lunch and/or dinner
- Each school is charged £50 per person to attend the conference and £10 for lunch and £15 for dinner
- You have been asked to design a prototype program to collect data from each teacher as they arrive at reception and produce bills depending on choices made.

Program Specification

There are 3 forms in the project. Your task is to design the main data entry form, the bill form and the totals form. Then link these together.

- All forms are well designed with pictures, good layout and choice of fonts and colours

4.7: The Data Entry Form

Design a form with the following functionality:

1. The attendee should be able to put their name into a textbox
2. They should be able to choose their school from a dropdown list
3. The various dining options should be available via radiobuttons or checkboxes
4. A **Quit** button on the form quits the program
5. A **See Bill** button allows the user to see their Bill (see 6.8)
6. A **Totals** button allows university staff to see the totals (see 6.9)

4.8: The Bill Form

This form should have a suitable BackgroundImage and the following functionality:

1. A button returns back to the main Data Entry form .. this can be done by simply closing the form
2. The name of the teacher and school name from the main form are displayed in a heading
3. The choices are also displayed together with a Total Bill.
4. An **Accept Bill** button asks the teacher if they are happy with their choices and if so, various totals and counts are updated (see 6.9)

4.9: The Totals Form

This form should have the following functionality:

1. There is a Back button that closes the form
2. The total number of teachers attending the conference so far is displayed
3. The individual school totals are displayed (see hints)
4. The total bills for each school are displayed

Some Hints and Reminders

1. Most of the programming will need to be done on the main **DataEntry** form.
2. You will need to define a number of variables to hold the various **totals** and bills, etc. You should do this near the beginning of the form code, after the form header.

e.g.

```
namespace Project
{
    public partial class frmDataEntry : Form
    {
            int totalDelegates = 0;
```

3. By default when you add objects like textboxes etc. to a form they are set to **Private**
4. This means that their contents are not available to other forms.
5. To make them available, you must to set the **Modifiers** property to **Public**

Some Extra Useful C# .NET Stuff

Some Display Instructions

```
_____.BackColor = Color.Blue;          // set a background colour
_____.ForeColor = Color.Yellow;        // set a foreground colour
_____.Text = "Hello " + name ;         // set the Text property
```

Delay for a time

```
System.Threading.Thread.Sleep (1000);
// gives a delay of 1 second (1000 milliseconds)
```

The Math.Pow() function

Use the Math.Pow() function to return the power of a number: e.g.

```
cube = Math.Pow(number, 3);
```

The Math.Abs() function

This function returns the value of a number, ignoring any + or - sign e.g.

```
value = Math.Abs(number);
```

Converting Text into upper case

```
lblName.Text = lblName.Text.ToUpper() ;
// converts a label's Text into upper case (e.g. yes becomes YES)
```

Random Number Generation

- First we must create a new object from the Random class .. e.g.

```
Random  r = new  Random();          // creates a new object called r
```

- Then you can use **r.Next()** to pick the next number e.g.

```
n = r.Next(6) + 1;          // puts either 1,2,3,4,5 or 6 into n
```

Output of Decimal Places

If you want to display a double type of variable (num) to 2 decimal places:in a label:

```
lblAnswer.Text = "The answer is " + num.ToString("0.00") ;
```

Number Conversion

e.g. Convert text from a textbox to the right type using Convert e.g.

```
num = Convert.ToDouble(txtAmount.Text);    or
num = Convert.ToInt32(txtAmount.Text);
```

Closing Down

```
Application.Exit();
```

Graphics Summary (after first getting a Graphics object g)

Clearing the Screen

```
g.Clear (BackColor);          // clear using the current backcolor
```

Rectangles and Squares

```
g.DrawRectangle (myPen, x, y, w, h);      or   g.FillRectangle(myBrush, x, y, w, h);
```

Ellipses and Circles

```
g.DrawEllipse (myPen, x, y, w, h);        or   g.FillEllipse(myBrush, x, y, w, h);
```

Polygons

```
g.DrawPolygon (myPen, shape);             or   g.FillRectangle(myBrush, shape);
```

Lines

```
g.DrawLine (myPen, x, y, w, h);
```

Text

```
g.DrawString (s, myFont, myBrush, x, y);
```

Pens and Brushes (for drawing and filling graphics)

```
Pen  myPen = new Pen(Color.Green, 5);          // create a new green pen, size 5
Brush  myBrush = new SolidBrush(Color.Red);    // create a new red brush
```

RGB Colours (up to 256 values each for Red Green and Blue combinations)

```
this.BackColor  = Color.FromArgb(100, 50, 10);      // set a RGB background for this form
```

Appendix B:
Assessment of CO453 Application Programming

1. The module is assessed by coursework that consists of a series of directed study exercises and programming projects that must be recorded in a logbook.

2. The logbook must be an e-book and should contain your designs, algorithms, test plans, source code and results of your work.

3. The directed study includes **independent study tasks** and **programming projects**.

4. The **classwork** component of the directed study is assessed each week in your practical sessions. You **MUST** be observed doing the classwork in the computer laboratories during these timetabled sessions. You must record your classwork in a logbook and this will be presented for inspection at designated times. Your attendance and achievement will be recorded weekly.

5. The **independent** component of the directed study is your own unaided work. This work must be recorded in your logbook. The **independent** directed study is assessed when logbooks are submitted

6. The **programming projects** are your own unaided work and must be recorded in your logbook. They are assessed at various times during the timetabled practical session.

7. The assessed directed study is contained in several directed study packs.
The weighting of the assessment to the final grade will be in the order of the following:

100% C# .NET Windows, Independent Study and C# .NET Windows Project –

15%; C# .NET Windows, Independent Study 60%; and C# .NET Windows Project 15%; contribution.)

Log Books

For the module you should submit (unless otherwise instructed):

- Clearly numbered Task headings
- Simple description of task(s)
- Sample Screen Shots
- Fully commented Source code of relevant sections using highlighter pen to show added code where necessary
- Comments on problems encountered etc.

Grade related criteria for Programming - CO453

A	Where the student has demonstrated clear evidence of an excellent understanding of the theories and principles together with a high degree of analytical accuracy, good design skills, implementing fully tested solutions that show reliability, maintainability, readability and minimal complexity and correct form of presentation skills. *To acquire the knowledge and skills to demonstrate the above the student will normally be expected to attend the lecture and practical sessions and attempt at least 85% of directed study for each week.*
B	Where the student has demonstrated clear evidence of a good understanding of the theories and principles together with a good analytical ability, good design skills, implementing solutions that show reliability, maintainability, readability and minimal complexity and correct form of presentation skills. *To acquire the knowledge and skills to demonstrate the above the student will normally be expected to attend the lecture and practical sessions and attempt at least 75% the directed study for each week.*
C	Where the student has demonstrated a reasonable understanding of the theories and principles together with a reasonable analytical ability, design skills, implementing solutions that appreciate the need for reliability, maintainability, readability and minimal complexity and reasonable presentation skills. *To acquire the knowledge and skills to demonstrate the above the student will normally be expected to attend the lecture and practical sessions and attempt at least 66% of the directed study for each week.*
D	Where the student has demonstrated an understanding of the theories and principles of analysis, design, implementation and presentation skills. *To acquire the knowledge and skills to demonstrate the above the student will normally be expected to attend the lecture and practical sessions and attempt at least 50% of the directed study for each week.*
E	Where the student has made a genuine attempt to acquire the knowledge and skills but requires further application and study to demonstrate an understanding of the theories and principles of analysis, design, implementation and presentation skills. *In order to demonstrate a genuine attempt the student will normally be expected to attend the lecture and practical sessions and attempt at least 40% of the directed study.*
F	Where the student has clearly not acquired sufficient knowledge and skills and not attempted or coped with the directed study with any degree of competence regarding theories, principles, analysis, design, and implementation and presentation skills or where the student has NOT attended for assessment or where the student has copied work from an alternative source.

Module Name and code	Application Programming CO453
Staff: Richard Jones, Richard Mather, Carlo Lusuardi & Nick Day	

Learning Outcomes:

- Analyse a simple requirement in a structured manner
- Design, document, implement and test reliable, maintainable programs as solutions to simple problems
- Use structured techniques of design and implementation and good documentation practice.
- Use software development tools.

Teach WK	Uni WK	LECTURE/TUTORIAL	PRACTICAL
1	18	**C# (Console) 4 Methods and Parameters**	C# Directed Study Unit 4
2	19	**C# (Console) 5 Arrays**	C# Directed Study Unit 5
3	20	**C# (Console) The Project**	C# Directed Study Project Unit
4	21	**Workshop**	
5	22	**Windows C#1 Introduction & Splash Screen**	C# Directed Study Unit 1
6	23	**Windows C#2 SPS Game**	C# Directed Study Unit 2
7	24	**Windows C#3 Other .NET Controls**	C# Directed Study Unit 3
8	25	**Windows C#4 Multiform projects**	C# Directed Study Unit 4
	26-28	**Spring Recess – (Easter)**	
10	29	**Workshop**	
11	30	**Windows C#5 Animation**	C# Directed Study Unit 5
12	31	**Windows C#6 Graphics**	C# Directed Study Unit 6
13	32	**Windows C#7 Web Programming**	C# Directed Study Unit 7
14	33	**Windows C#8 The .Net Project**	C# Directed Study Unit 8
15	34	**Workshop**	

Course Texts:

Comprehensive Course Notes are provided

Bradley & Millspaugh, ***Programming in C#***, 2010, pub: McGraw Hill

Deitel & Deitel, ***Visual C# 2010 How to Program***, 2011, pub: Pearson