Week 1: Practical

Introduction to Moodle and Visual Studio .NET

This practical will introduce you to techniques in designing Visual Studio applications and documenting projects. You will also learn how to graphically draw lines on a form using C#. Read the entire practical before you begin creating your application.

Reading

Before starting this practical, you should have read Chapters 1 and 2 of Gaddis.

Getting Started

In your documents folder create a new folder called **101**. You should save all of your applications for this course in this folder. In your **101** folder create another folder called **Week 1** which will hold your work for the practical for Week 1.

Each practical has set exercises to complete and to get full marks you must complete all exercises to the satisfaction of the demonstrator. Everyone must attempt Exercise 1 of each practical. Many practicals will have options for Exercise 2 and you must complete one of the options. Chose an option that interests you or is relevant to the degree that you are doing. You may complete more than one option but you can only have one option marked. The textbook also has many exercises that you can attempt for more practice. The more practice you do the better you will become at creating applications in C# .NET.

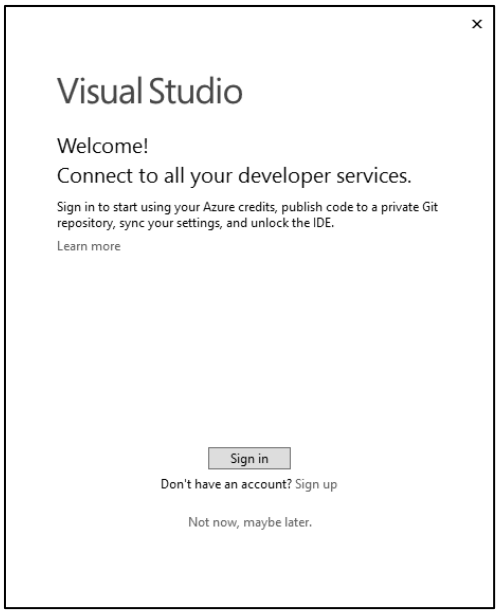
Course Library Folder

Some of the practicals will require you to copy some files into your account before you can complete the practical. All files required for practicals can be found in the COMPX101 course library folder. To get to the library folder, double click on **My Computer** on your desktop, or right-click on the Start (Windows) icon and select **File Explorer**. You should then see an icon named **Library (L:)**, double click on this icon and then double click on the folder called **COMPX101**. All files for the practicals will be inside this folder. DO NOT copy the entire COMPX101 library folder into your

account, just copy the files inside COMPX101 to your account as required.

Starting Visual Studio

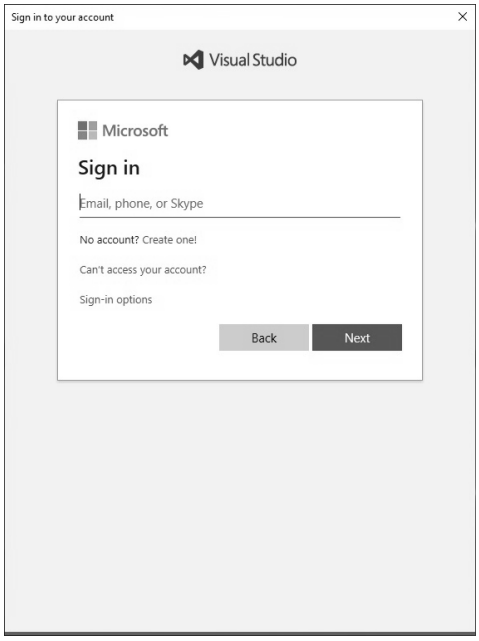
From the Start (Windows) menu, select **Visual Studio 2019** (the program, not the folder).



You may see a window which looks similar to this:

You will then need to sign in using your university credentials otherwise Visual Studio will stop working after 30 days. Click the **Sign In** button.

You will then see:

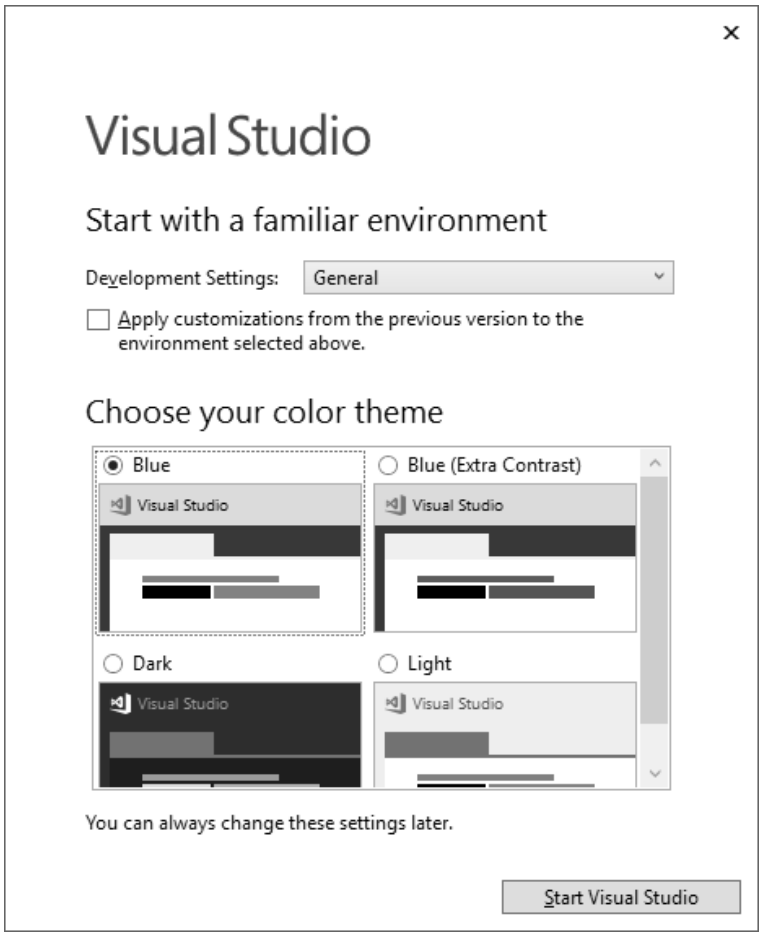


Type in your university email address:

<moodle username>@students.waikato.ac.nz

You will then be taken to a university login page so type in your university password then click ok and it will register Visual Studio to your university account. You can do this if you also install Visual Studio on your own computer.

You may then see the following screen:



Change the **Development Settings** option to **Visual C#** and then choose a colour theme of your choice and then click on the **Start Visual Studio** button.

Using Visual Studio

It is vital that you watch the Introduction to Visual Studio recording in the Panopto section on Moodle to show you how the lab works and on how to use Visual Studio. Even if you have used Visual Studio before it is highly recommended you watch the video as it explains how it works in our lab.

Exercise 1

Create a folder called **Week1** in your **101** folder. Copy the **Week1-Ex1** folder from the COMPX101 library folder into your Week1 folder. Inside the folder you will find a **Week1-Ex1.sln** file, and a folder with the code and other files that make up the Visual Studio project.

Open the project in Visual Studio either by using **File > Open** and selecting the **Week1-Ex1.sln** file, or by double clicking on the **Week1-Ex1.sln**.

Note: Do not try to open the program by opening the .cs code file(s). Although it will load the code into Visual Studio, it will not load the program properly and will not work.

Run the program to see what it does. Now look at the code and determine which lines of code cause which effect when the program is run.

a) Alter the code by replacing the word "world" with your name. Test your change.

b) Alter the code so that the Smaller button makes both the Width and the Height of the form smaller, in the same way that the Bigger button works. Test your program. What happens if you repeat pressing the Smaller button?

c) Select the Smaller button, and look at its properties in the designer window (normally bottom right). Find the **Anchor** property and change it so the button is anchored to all four edges. What does this change do?

Week 1 Practical: REVIEW PAGE Name: _____

Questions: Complete the following questions. (Note: this must be done before you seek a verification).

1. In the compx101 Moodle forum, when is Quiz 1 due (i.e. when does it stop you taking it)?

2. How difficult was it for you to find the **Week1-Ex1.sln** file when Windows hides the important file extensions by default? How annoying is this?

3. What happened when you repeated pressing the Smaller button? Did the form shrink to nothing?

4. What effect did changing the anchor property have?

Verification: To assess your competency of this material your demonstrator will verify that you have:

1. Answered the set questions.
2. Created the required applications.
3. Chosen your two lab times in Moodle.

Week 2: Practical

An Introduction to Visual Studio and C#

This practical will introduce you to techniques in designing Visual Studio applications and documenting projects. You will also learn how to graphically draw lines on a form using C#. Read the entire practical before you begin creating your application.

Reading

Before starting this practical, you should have read Chapters 1 and 2 of Gaddis. You should also have read Appendix D: Introduction to Algorithms and the pseudo-code guide at:

`http://users.csc.calpoly.edu/~jdalbey/SWE/pdl_std.html`

Getting Started

Create a paper prototype for your user interface for Exercise 2 and show this to the demonstrator when you get this practical verified. The user interface should be designed using the interface design practice and guidelines you have seen so far on the course and in the textbook.

Exercise 1

In this exercise you will learn how to draw shapes in a picturebox.

a) Create a new C# .NET solution and save it in your **Week 2** folder. Add a picturebox control to the form and give it the name **`pictureBoxDisplay`**. Create a button at the bottom of the form and give it the name **`buttonDrawLine`** and make the text in the button read **`Draw Line`**. Change the title of the form to be **`Drawing Application`**. Also set the **`StartPosition`** property of the form to **`CenterScreen`**. This means that whenever the applications runs the form is always in the centre of the screen. You should do this for all applications you create unless you have a specific reason not to.

Hint: To change the title of the form, change the **`Text`** property of the form in the designer window.

Create a click event procedure for the button (by double clicking on the button in the design window) and inside the curly braces type in the following code:

```
Graphics paper = pictureBoxDisplay.CreateGraphics();
Pen pen1 = new Pen(Color.Blue, 5);

paper.DrawLine(pen1, 10, 10, 100, 100);
```

Now run the application and when you click on the button you should see a line appear in the picturebox. Try changing the thickness of the pen with different colours. Can you draw multiple lines in the picturebox? Try doing this. Can you create pens of different colours and draw lines with them? Explore, have fun!

b) Create an **`Exit`** button next to the **`Draw Line`** button which closes the application. Give the button an appropriate name.

The statement to exit from the application is:

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

c) Create a **`Draw Square`** button and draw a square with each side of the square drawn in a different colour.

Note: If the form is minimised or another window is placed on top of the form then the drawing will disappear. Do not worry about trying to fix this, as it is not important. When getting the exercise verified just keep the drawing window open, or click the Draw Line button again.

Exercise 2

a) Create a new C# .NET solution and name the project **`MouseMoveGraphics`** and save it in your **Week 2** folder. Make the form reasonably large. Add a picturebox control to the form and give it the name **`pictureBoxDisplay`**. Make the picturebox fill up all of the form.

b) You will now need to create a custom event for the picturebox. Follow the instructions below and if you have problems then ask the demonstrator for help.

Click on the picturebox and in the **Properties** window click on the *lightning bolt*. This will switch the window to show the events you can associate with a control instead of the properties. Scroll down until you find the **MouseMove** event and double click on it. It will then generate the MouseMove event handler for the picturebox. This is how you can associate different events to controls from the default ones which are generated when you double click on a control.

The **MouseMove** event is triggered whenever you move the mouse pointer while it is over the picturebox. What you will be doing is drawing lines as you move the mouse.

c) In the **MouseMove** event, create a Graphics object connected to the picturebox. Also create a pen called **pen1** and give it a colour and a size of 2.

What you need to do is draw a line from position 0,0 to where the mouse pointer is. The **e** parameter variable stores the x and y position of the mouse inside its properties. Type in the following:

```
paper.DrawLine(pen1, 0, 0, e.X, e.Y);
```

This draws a line from 0,0 to the x and y position of the mouse. Run the application and see what happens when you move the mouse pointer around the picturebox. What happens if you move the mouse pointer outside the picturebox?

d) Change the code so that it draws from the centre of the picturebox to the mouse pointer location. Then run the application.

Exercise 3

Create a new C# .NET solution and save it in your **Week 2** folder.

For this exercise, you only need to create the user interface. You do not need to make the application work. You will do that in Week 3's practical.

Nilesh has started a new business, "Hard As Concrete", which lays concrete driveways for new homes. Nilesh remembered you were a genius programmer and wants you to build an application which allows him to type in the length (e.g. 5.3) and width (e.g. 2.5) of the driveway in metres. It will then calculate and display (in separate textboxes) the volume of concrete required (the depth of every driveway is always 0.5m), the number of kilograms of cement required, the number of bags of concrete required and the total cost of the cement. One kg of cement will make 1.5m^3 of concrete and the cost of a bag of 2kg cement is $15.50.

Your application must have **3** buttons: **`Calculate Concrete Cost`**, **`Clear`** and **`Exit`**.

The **`Clear`** button will clear the input text boxes and all output textboxes and set the focus to the first input text box. The **`Exit`** button will end the application.

a) Produce a paper prototype of the user interface. Think about what controls you will need and the layout of the controls on the form. Think carefully which controls should be read-only. Get your demonstrator to check the paper prototype before continuing.

b) Now build the user interface and add your name and id number as comments at the top of your code. You don't need to write any code yet, you will finish this application in Week 3.

Paper Prototype:

Week 2 Practical: REVIEW PAGE Name: _____

Questions: Complete the following questions. (Note: this must be done before you seek a verification).

1. What is the purpose of paper prototyping the user interface of an application?

2. In exercise 1, what happens if you move the mouse pointer outside the picturebox?

3. If you wanted to display the cost of an order that is calculated by an application, which control would you use?

4. What is an object? List the objects you used in exercise 1.

Verification: To assess your competency of this material your demonstrator will verify that you have:

1. Drawn the paper prototype for exercise 2.
2. Answered the set questions.
3. Created the required applications.
4. Used appropriate names for your controls.

Week 3: Practical

Using Objects, Variables and Constants

This practical will introduce you to using variables and objects in C# to store values while an application is running. You will also learn about the scope of variables and how to create and use constants. Read the entire practical before you begin creating your application.

Reading

Before starting this practical, you should have read Chapter 3 of Gaddis.

You are not required to produce a paper prototype for this practical.

Exercise 1

a) Create a folder called **Week 3** in your **101** folder. Copy the **`MouseMoveGraphics`** project from Week 2 into your Week 3 folder and then open the project.

b) Add a **`Set Colour`** button to your form. Now add the **`ColorDialog`** control to your project from the **`Dialogs`** section in the toolbox and change the name to **`colorDialog1`**. This control allows the user to visually select a colour which is stored in the **`Color`** property of the dialog control. In the click event method of the **`Set Colour`** button type in the following to display the dialog window to the user:

```
colorDialog1.ShowDialog();
```

c) Modify the **`MouseMove`** event method so that the lines are drawn in the colour chosen in the dialog control. Run the application and try changing the colour several times.

Hint: When creating the pen, remove the colour value and replace it with the Color property from the dialog control.

Exercise 2

a) Create a folder called **Week 3** in your **101** folder. Create a new C# .NET solution and save it in your **Week 3** folder.

Add a **`Draw Triangle`** and **`Draw Square`** button to the form.

In the click event method of the **`Draw Triangle`** button ***declare three point objects, corner1, corner2 and corner3*** with different coordinates. Using the **`DrawLine`** method draw lines between the corners to form a triangle on the form.

b) Create an **`Erase`** button that will erase any graphics in the picturebox. In the click event procedure of the button type the following:

```
pictureBoxDisplay.Refresh();
```

Note: This command will only clear the graphics that have been drawn on the form. It does not clear the contents of controls like text boxes and labels.

c) In the click event method of the **`Draw Square`** button, create an x variable set to 50, a y variable set to 80 and a size variable set to 60. Create a graphics object connected to the picturebox and a pen object and then use the **`DrawRectangle`** method to draw a rectangle using the variables you declared previously. Use the size variable for the width and height of the rectangle.

Exercise 3

a) Watch the recording on using the Visual Studio debugger tool that is posted in the Week 3 section on Moodle.

b) Copy your project folder for **Week 2 Exercise 3** into your **Week 3** folder and then open it in Visual Studio. You will now complete the code for the application.

Add the following constants:

```
//The depth of every driveway.
const double DRIVEWAY_DEPTH = 0.5;
//Amount of concrete created from 1 kg of cement
const double CONCRETE_PER_KG = 1.5;
//Weight of a bag of cement
const double BAG_WEIGHT = 2.0;
//Cost of a bag of cement
const decimal BAG_COST = 15.5m;
```

c) Write the code for the **`Clear`** button which will clear the input text boxes and all output values and set the focus to the first text box. Write the code for the **`Exit`** button.

d) Write the code for the **`Calculate Concrete Cost`** button using the pseudo-code given below (also use a Try..Catch structure in the code to deal with errors):

```
TRY
   Declare Variables
   GET the length of the driveway
   GET the width of the driveway
   CALCULATE the volume of concrete required
     ( = length * width * depth)
   CALCULATE the number of kilograms of cement required
     ( = concrete volume / concrete from 1 kg of cement)
   CALCULATE the number of bags of cement (rounded up)
     ( = kilograms required / weight of a bag of cement
   CALCULATE the total cost of the cement
     ( = number of bags of cement * cost per bag)
   DISPLAY the volume of concrete required (to 3dp)
   DISPLAY the number of kilograms of cement required (to 3dp)
   Display the number of bags of cement required
   DISPLAY the total cost of the bags formatted as currency
CATCH
   Display error message
   Clear all textboxes
   Set focus to first input textbox
ENDTRY
```

Hint: *int x = (int)Math.Ceiling(y) gives the value of y rounded up to the nearest whole number.*

Week 3 Practical: REVIEW PAGE Name: _____

Questions: Complete the following questions. (Note: this must be done before you seek a verification).

1. What is a variable and what is it used for?

2. How do you create a variable that can be accessed by any method in the form's class? Where should this variable be declared?

3. What is a **constant** and why would you use one?

4. What does the Parse method do? Give an example of how to use it.

Verification: To assess your competency of this material your demonstrator will verify that you have:

1. Created the required applications.
2. Answered the set questions.
3. Can use the debugger to set a breakpoint, step through the code and display values of variables using the mouse.

Week 4: Practical

Using Selection Structures

This practical will introduce you to using selection structures to alter the flow of control in your code. You will learn two different methods of selection and the use of Boolean operators. Read the entire practical before you begin creating your application.

Reading

Before starting this practical, you should have read Chapter 4 of Gaddis. You should also read section 5.8 from Chapter 5 on generating random numbers.

Exercise 1

a) Create a folder called **Week 4** in your **101** folder. Copy the **`MouseMoveGraphics`** project from Week 3 into your Week 4 folder and then open the project. At the moment you cannot control when the lines are being drawn. So you will now change the code so that lines are only drawn when the mouse button is pressed. The pseudo-code is:

```
IF the button pressed is the left button THEN
  Draw the line
ENDIF
```

In the **`MouseMove`** event for the picturebox type in the following:

```
if (e.Button == MouseButtons.Left)
{

}
```

This checks the button property of the e parameter variable which stores a value for the button that was pressed. It checks this against the value for the left mouse button. Move the **`DrawLine`** method call in between the curly braces. Now the line is only drawn if the left button is pressed and the mouse is moved. Test the application.

b) Now let's try to draw lines from random x and y position to the mouse location. First you must create the Random number generator object at **class scope level** at the beginning of your form class. Type in:

```
Random rand = new Random();
```

This creates the object called rand which you can use to generate random numbers. You only create the object once at the start of the application which is why it is created at class scope level.

If you wanted to generate a random number between 0 and the width of the picturebox (inclusive) you would type in `rand.Next(pictureBoxDisplay.Width+1)`. We can use this when we draw the line for the start x and y position of the line. Comment out the **`DrawLine`** method you have in the code and then type in:

```
paper.DrawLine(pen1, rand.Next(pictureBoxDisplay.Width+1),
rand.Next(pictureBoxDisplay.Height+1), e.X, e.Y);
```

Test the application. You need to add 1 to the width and height because the Next method always gives a value between 0 and the max value – 1.

c) Lets try drawing circles instead. Comment out the second DrawLine method call and type in:

```
paper.DrawEllipse(pen1, e.X, e.Y, 50, 50);
```

e.X and e.Y specify the x and y position of the top left hand corner of the circle. 50, 50 represent the width and height of the circle. Run the application and see what happens. Experiment with changing the width and height values and what happens if the width and height are different. **Set the width and height back to 50 before continuing**.

d) Lets make filled in circles now. Change the colour of the pen to **`Black`** when it is declared. Now also declare a Solid Brush object (you can pick a different colour if you want):

```
SolidBrush br = new SolidBrush(Color.Orange);
```

Inside the if statement but before the **`DrawEllipse`** method call, type in:

```
paper.FillEllipse(br, e.X, e.Y, 50, 50);
```

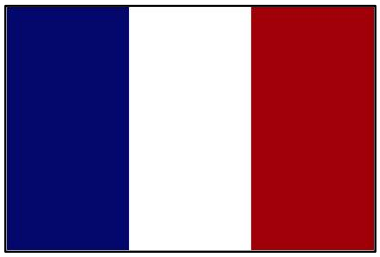
The FillEllipse method uses the brush to fill in the circle with the colour of the brush. Test the application. Test what happens when you put the FillEllipse method call after the DrawEllipse method call, then put it back to the original way.

Test what happens when you use e.Y for the **x** position and e.X for the **y** position of the circles. Then change it back before continuing.

Exercise 2

a) Create a new C# .NET solution and save it in your Week 4 folder.

You will create an application which draws the French flag to fill the entire picturebox.



The first bar of the flag is blue, the second bar is white and the third bar is red.

b) Add a large picture box. Also add a **`Draw Flag`**, **`Clear`** and **`Exit`** button. Add a constant to store the number of bars in the flag which is set to 3.

c) Create the click event method for the Exit button and write the code to close the application. Create a click event method for the **`Clear`** button which clears any graphics in the picturebox.

h) Create a click event method for the **`Draw Flag`** button using the pseudo-code below. ***Use x and y variables set to 0 for the position of the first bar of the flag and then shift the x variable across to draw the other bars.***

```
Declare variables
Calculate the width of a bar
   ( = width of picturebox / number of bars)
Draw blue bar at current x and y position
Shift x variable to the right by the bar width
Draw white bar at current x and y position
Shift x variable to the right by the bar width
Draw red bar at current x and y position
```

Exercise 3

You only need to complete 1 option from the following to get your practical marked. You can do more than one if you have time and want to practice.

Option A: **Coin game.** Create a new C# .NET solution and save it in your **Week 4** folder.

You will create a simple coin tossing game where the player can bet a certain amount of money against coin tosses and keep playing until they run out of money.

The application will have a textbox to display the player's current balance and a textbox to allow the user to enter the amount to bet. It will need to show the values of the two coins after rolling. You could just display each coin value in a text box or use picture boxes and display a picture or even draw graphics the picturebox to show heads or tails for a coin. Add two radio buttons to the form, one for choosing Heads and one for choosing Tails. It will also need a Toss Coins and Exit buttons.

The Toss Coins button will get the player's bet. It will then check to make sure that the player's bet is less than or equal to their current balance, if it isn't then display an error message. If the bet amount is ok then it will generate a random number between 1 and 2 for each of the two coins and display the values of the coins (1 means head and 2 means tail).

If the user chooses heads and both coin values are 1 then the player wins and the value of their bet is added to the balance. If the player chooses tails and both coin values are 2 then the player wins and the value of their bet is added to the balance. Otherwise they lose and their bet amount gets subtracted from their balance. An appropriate message should be displayed whether they win or lose each time they toss the coins. The balance should be re-displayed in the textbox. The bet amount should be cleared and the bet textbox should be given the focus. Use a Try..Catch structure to deal with any errors.

a) Write the pseudo-code for the **`Toss Coins`** button click event method and get a demonstrator to check it before going on to the next step.

b) Now create the user interface for the application.

c) You want the balance to be shown as soon as the application is started. To do this select the form in the Design Tab. Click on the lightning bolt in the properties window to see the events for the form. Double click on the **Shown** event to create it's click event method. This event is triggered when the form is first shown so any setup code for an application should go in here.

Create a class scope variable to store the balance and set it to a value of your choosing. In the Shown event method write the code put the balance into the textbox.

d) Create the Toss Coins button click event method and implement the code using your pseudo-code.

e) Test the application with different inputs to make sure the results are what you expect.

Option B: **Simple Logic Simulator.** Create a new C# .NET solution and save it in your **Week 4** folder.

You are to create an application to simulate an And gate and an Or gate. You will need two **checkboxes** for the two inputs going into your gate, input A and input B. You could use a label with a fixed border style, back colour set to White, with no text in it to represent a lamp or use a picturebox and display different images for the state of the lamp. You will need another **checkbox** to represent if the Power is on or off.

If the lamp is off the back colour of the label should be white and if it is on the colour should be yellow. Alternatively you could use a picturebox and display a picture when it is off and another picture when it is on.

When the Power button is clicked on you need to visually show the state of the power like the input buttons.

Also when the power is turned on, the application will check to see which type of gate is being simulated and turn the lamp on or off as appropriate.

a) Write the pseudo-code for the Power button and then get a demonstrator to check your pseudo-code before going onto the next step.

b) Now create the application and for the Power button's click event method code use your pseudo-code.

c) Test the application with different inputs to make sure the results are what you expect.

Optional: Extend the application so that the simulation runs if the power is on and one of the input checkboxes is changed.

Option C: **Morra Game.** Create a new C# .NET solution and save it in your **Week 4** folder.

Morra is a hand based game dating back to the ancient Greek and Roman civilisations because they didn't have tv's and computers back then. In this version of the game each player shows one hand displaying the number of fingers extended. Each player also simultaneously shouts out a guess as to the number of fingers extended by the other player. If a player correctly guesses the number of fingers extended by the other player and the other player guesses wrongly, then that player wins and the total number of fingers extended by both players is added to the player's points. If both players guess correctly or both players guess incorrectly then the game is a tie and no points are added to either player.

You will create an application to play the game, human against computer.

For the player you will need a read only textbox to display the number of points for the player. You will also need a normal textbox for the player to enter the number of fingers they want to extend and their guess as to how many fingers the computer will extend. You may want to have a picture box that displays the players fingers graphically using an image or drawing the graphics yourself.

For the computer you will need a read only textbox to display the number of points for the computer. You will need a read only textbox to display the computers guess as to how many fingers the player will extend. To display the number of fingers the computer will extend you can use a read only textbox or use a picturebox that displays the computers fingers graphically using an image or drawing the graphics yourself.

Hint: There will be a recording posted in Week 4's section on Moodle that shows you how to load images into a project and then display them in a picturebox through the code.

You will need a Play button that will play a round of the game and each time the Play button is clicked another round of the game is played and the points are updated accordingly.

You will need **class scope** variables for the points for each player.

In the Play button you will need to get the players number of fingers and their guess. Then generate the computers number of fingers and it's guess as random numbers then display those to the user. Then use an if statement to figure out who wins or if it is a draw and then update the points for each player and display them in the textboxes.

a) Write the pseudo-code for the Play button and then get a demonstrator to check your pseudo-code before going on to the next step.

b) Now create the user interface for the application.

c) You want the points to be shown as soon as the application is started. To do this select the form in the Design Tab. Click on the lightning bolt in the properties window to see the events for the form. Double click on the **Shown** event to create the click event method. This event is triggered when the form is first shown so any setup code for an application should go in here. Write the code put the point values for each player into the textboxes which should be 0.

d) Write the code for the Play button based on your pseudo-code. Test the application with different inputs to make sure the results are what you expect.

Option D: **Lunar Lander Game.** Copy the Lunar Lander template folder from the COMPX101 library drive to your **Week 4** folder or download it from the Datafiles folder on Moodle and unzip it.

You will create a simple version of the lunar lander game where your lander is falling to the ground and must land completely on the landing pad otherwise your lander will crash and the astronauts inside will die.

For this project you will draw the graphics directly on the form rather than in a picturebox as it makes it easier to handle the keyboard events. A Start button is provided which you will hide and un-hide as required.

In the code some class scope variables are provided for you, read the comments to see what they will be used for.

a) Go to the designer window and select the Form. Then in the Form properties find the **KeyPreview** property and set it to **true**. This has to be done so that the keyboard events will trigger.

b) Create a class scope constant for the amount of fuel the lander will have and set it to 100.

c) Add a **Timer** control to the form, you can rename it if you want to or leave it as timer1. Leave the interval property at 100 so the timer ticks every 100 milliseconds. Whenever the timer ticks it will execute the code in it's tick event (which you will create later).

d) Add the **lander.jpg** file that is in the project folder as a resource to your C# project. See the video on panopto which will show you how to do this.

e) Create the click event method for the **Start** button. In the method set the visible property of the start button to false to hide it. Create a new point object for the position of the lander, it's x position should be in the middle of the form (this.Width will give you the width of the form) and the y position should be 0 (the top of the form).

Then set the initial velocity of the ship to 0. Create a new rectangle object for the landing pad, it's x position should be a random number between 0 and the width of the form - 60 (so the pad is not

cut off from the right hand edge), it's y position should the height of the form – 50, a width of 60 and a height of 50.

Hint: Use the rectangle variable already declared at class scope and just create a new object for it.

Create a new rectangle object for the fuel gauge, it's x and y position should be 0, 0, it's width should be set to the fuel constant (the width determines how many units of fuel the lander has) and a height of 10. Use the rectangle variable already declared for you

Then start the timer by calling it's Start method and give the form the focus using **this.Focus()**. The "this" object refers to the current form. By doing this all keyboard events will be directed at the form which is what we want.

f) In the form designer window, double click on the timer to create the tick event. Create a graphics object connected to the form using **this.CreateGraphics()** and create three solid brushes, one in the colour **GreenYellow** for the landing pad, one in **Orange** to indicate lots of fuel remaining and one in **Red** to indicate low fuel. Then clear the picturebox with the colour black using the Clear method of the graphics object.

Now we will apply gravity to the lander, add 1 to the velocity of the lander. Each time the timer ticks the velocity will get faster.

Change the y position of the lander by adding the velocity to it and then draw the lander using the **DrawImage** method of the graphics object at it's current x and y position. See the video on panopto if you can't figure out how to do this. Also draw the landing pad as well.

Now run and test the game and you should see the lander fall down and drop past the bottom of the picturebox. Close the application and come back to the code.

Now here is the description of the rest of the code for the tick event. If the width of the fuel gauge is less than 20 then draw it in red otherwise draw it in orange. If the lander sprite is completely in between the left and right edges of the landing pad and the bottom edge of the sprite is greater than or equal to the top edge of the pad then stop the timer (can you figure out how?), display a

landed message and set the visible property of the button to true. Otherwise if the bottom edge of the lander is greater than the height of the form then stop the timer, display a crashed message and set the visible property of the button to true.

Write the pseudo-code for all of the steps in the timer tick event.

Now write the rest of the code for the tick event. When working out if the lander has landed successfully or not, try to solve the problem but as a demo for help if you get stuck.

g) Now you just need to add the keyboard event to make the lander move left and right and apply thrust. Add a **KeyDown** event for the form (ask a demo if you can't figure out how to do this). The 'e' parameter variable which is created automatically stored the value of a key press in the KeyCode Property. If you typed in:

```
if (e.KeyCode == Keys.Up)
```

then the condition will be true if the user pressed the up arrow key. Here is the pseudo-code for the event method:

```
IF key pressed is up arrow key THEN
   IF fuel > 0 THEN
      Subtract 5 from velocity
      Subtract 5 from fuel
   ENDIF
ELSE IF key pressed is left arrow key THEN
   Subtract 2 from x position of the lander
ELSE IF key pressed is right arrow key THEN
   Add 2 to the x position of the lander
ENDIF
```

Run and test the game, if the lander seems sunken into the landing pad that is fine, don't worry about it.

Pseudo-code:

Week 4: Practical REVIEW PAGE Name: _____

Questions: Complete the following questions. (Note: this must be done before you seek a verification).

1. Write an If statement equivalent to the following english expression:

   A is equal to B or A is greater than C and B is equal to D

2 Given the following values: A=3, B=3, C=2, D=1, is the above expression true or false? What is the problem here in terms of precedence?

3. State the benefits of writing comments in program code.

Verification: To assess your competency of this material your demonstrator will verify that you have:

1. Created the required applications.
2. Answered the set questions.

Week 5: Practical

Using Repetition Structures

This practical will introduce you to repetition structures to repeat (or loop) statements many times. Read the entire practical before you begin creating your application.

Reading

Before starting this practical, you should have read sections 5.1 up to and including section 5.5 of Chapter 5 of Gaddis.

Exercise 1

a) Copy the **`MouseMoveGraphics`** project from Week 4 into your Week 5 folder and then open the project Create a variable called **`size`** in the **`MouseMove`** event method of the picturebox. Inside the if statement before you draw anything store a random value between 0 and 50 in the size variable. Then use the size variable for the width and height for each method call. This means a new circle size is generated when the mouse moves. Test the application.

b) Having all the circles in the same colour is a bit bland. You will now generate a new colour for the SolidBrush before drawing the circles. You will use the FromArgb method to generate a new colour value. The method is passed 3 values (a value between 0 and 255) to represent the Red value, the Green value and the Blue value of a colour.

Before drawing the circles type in:

```
br.Color = Color.FromArgb(?, ?, ?);
```

You need to replace each '?' with a different random value between 0 and 255 (inclusive). Do not use the same random value for Red, Green and Blue as this will give you a gray colour.

Run and test the application.

c) You will now extend the code so that when the mouse is pressed it will draw a random number of circles around the mouse pointer with their positions slightly different from each other.

The pseudo-code is:

```
IF left mouse button is pressed THEN
   Generate a random number of circles between 5 and 10
   Set counter variable to 1
   WHILE counter <= number of circles to draw
      Generate a random number between 2 and 10 for
         the size of the circle
      Store xPos = mouse X + random number between
         -10 and 10
      Store yPos = mouse Y + random number between
         -10 and 10
      Draw ellipse at xPos and yPos position
      Add 1 to counter variable
   ENDWHILE
ENDIF
```

Modify the **MouseMove** event for the picturebox so that it matches the pseudo-code above. Run and test your application. Congratulations! You have now created your very own custom airbrush tool that you have complete control over. Try producing some art using the tool. Be careful as there is no undo tool if you make a mistake.

Press the **PrintScrn** button on the keyboard to copy your screen and then open MS Paint and paste the screen into Paint and save it as a jpg file. Post the image in the *Artwork* thread in the Paper Discussion forum on Moodle and the best one will get a **peanut slab**.

Exercise 2

You only need to complete 1 option from the following to get your practical marked. You can do more than one if you have time and want to practice.

Option A: **Term Deposit Calculator.** Create a new C# .NET solution and save it in your **Week5** folder.

Nilesh wants an application that will display the amount of interest generated over a number of years on a term investment.

Nilesh will enter the initial deposit for the term investment, the interest rate and the number of years of the term investment using textboxes. It will have a Calculate button that will display a table in the Console/Output window showing the interest generated each year and the current deposit value. You can assume the interest is only paid at the end of the year and is not compounded monthly. The application should make sure that the deposit value and interest rate are above 0 and that the number of years is 1 or more.

For instance, if the user types in 10000 for the deposit, 5.5 for the interest rate and 10 for the years, then the following is displayed:

```
Year 1: interest = $550.00 and deposit = $10,550.00
Year 2: interest = $580.25 and deposit = $11,130.25
Year 3: interest = $612.16 and deposit = $11,742.41
Year 4: interest = $645.83 and deposit = $12,388.25
Year 5: interest = $681.35 and deposit = $13,069.60
Year 6: interest = $718.83 and deposit = $13,788.43
Year 7: interest = $758.36 and deposit = $14,546.79
Year 8: interest = $800.07 and deposit = $15,346.87
Year 9: interest = $844.08 and deposit = $16,190.94
Year 10: interest = $890.50 and deposit = $17,081.44
```

All output is displayed in the Output Window using the **`Console.WriteLine`** method

a) Complete the pseudo-code below for the **`Calculate`** button and then get a demonstrator to check your pseudo-code before going on to the next step.

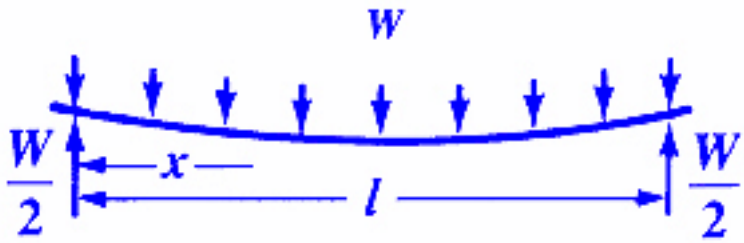
b) Now create the application and write the code for the **`Calculate`** button using your pseudo-code.

c) Test the application with different inputs to make sure the results are what you expect.

d) Document your code appropriately and add your name and id number as comments at the top of your code.

Option B: **Beam Deflection.** Create a new C# .NET solution and save it in your **Week5** folder.

This problem deals with calculating the beam deflection on a beam supported at both ends with a uniformly distributed load W.



More information can be found at:

www.engineersedge.com/beam_beanding/beam_bending1.htm

Nomenclature for beam deflection and stress calculation equations:

W = load (lb)
E = Modulus of elasticity (lb/sq-in)
I = Moment of inertia (in4)
x = distance from datum point (in)
l = beam length (in)
y = deflection (in)

The deflection of the beam (y) in inches, at any point in the beam, is given by:

$$y = \frac{Wx(l-x)}{24EIl}\left[l^2 + x(l-x)\right]$$

Allow the user to enter values for W, *E*, *I*, *l* and x. The user will be able to click on a calculate button which will display the beam deflection at the point given by input for x.

Try with the following values:

```
W = 2500 lb
E = 30000000 psi
I = 258 in⁴
l = 300 in
x = 60
```

your answer (y) should be 0.067 in.

The application will calculate and displays the deflection at every 5 inches of the beam, i.e. at x = 0, x = 5, x = 10, etc. Use the Console/Output window to display the x value and the corresponding deflection (y) value.

a) Write the pseudo-code for the **`Calculate`** button which will display the deflection at 5 inch intervals along the length of the beam and then get a demonstrator to check your pseudo-code before going on to the next step.

b) Now create the application and test that you get the correct answer using the example values with x at 60. Then write a loop that starts x at 0 and moves x every 5 inches along the beam, displaying the deflection every time.

c) Test the application with different inputs to make sure the results are what you expect.

Option C: **Drawing Stars.** Create a new C# .NET solution and save it in your **Week5** folder.

Do you know how to draw a star? You draw 5 lines, rotating 2*360/5 (=144) degrees to the right after each line.

Write a program that draws a five-sided star shape in a large picture box. Allow the user to enter the size of the star by specifying the length of each side. The click event method for your **`Draw`** button should contain a loop that draws five sides, where each side starts at point (x_i,y_i) and goes to point (x_{i+1},y_{i+1}), and the points are calculated as follows:

```
//this gives us 144 degrees, but in radians
angle =  4.0 * PI / 5.0
```

$x_{i+1} = x_i + size*cos(angle*i)$
$y_{i+1} = y_i + size*sin(angle*i)$

Hint 1: Use the appropriate methods of the **`Math`** object to get the sin and cos values.

Hint 2: Recurrence Equations, like the definition of (x_{i+1},y_{i+1}) above, define the i+1'th value of the sequence in terms of the previous, i'th, value. When we write a loop to generate a sequence of these values, we need to remember the i'th value from one iteration of the loop to the next, so we have to declare the x and y variables outside the scope of the loop body. So your code should look something like this pseudo-code (where i is the number of iterations we've done):

```
//declare the x and y variables or Point objects for
//the i'th point and the i+1'th point,
//initialise them to the start position (x0,y0)

while (…)   // or use a for loop
{
   //note that x and y contain the (xi,yi) values at
   //this point, calculate the xi+1,yi+1 values into
   //two temporary variables.

   //draw the line from (xi,yi) to (xi+1,yi+1)
   //copy the xi+1,yi+1 values into x and y, ready for
   //the next iteration
}
```

a) Now create the application using the pseudo-code given above.

b) Test the application with different inputs to make sure the results are what you expect.

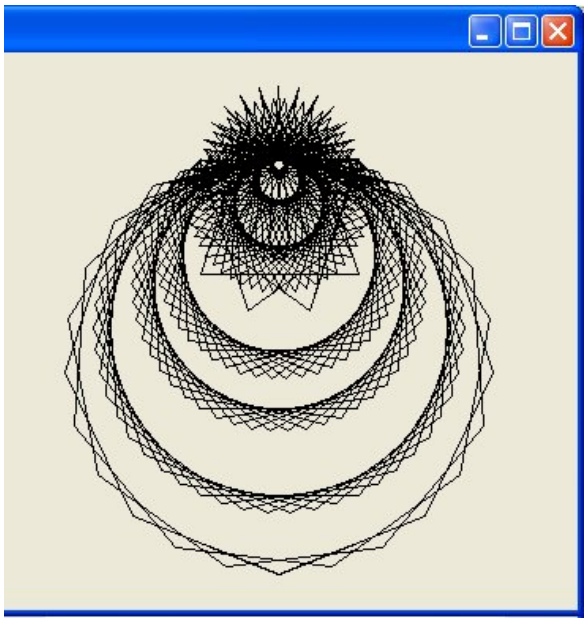
Crazy Stars: Let's generalize the above program so that we can draw star-like shapes with many different sides, and even "spirograms" (a popular drawing tool for children).

Change your program so that the user can enter the number of sides and the desired number of rotations, as well as the size of each line, then the Draw button will draw a star with that number of sides.

```
//in radians
angle = rotations * 2.0 * PI / sides
```

$x_{i+1} = x_i + size*cos(angle*i)$
$y_{i+1} = y_i + size*sin(angle*i)$

If you set Sides = 5 and Rotations = 2, you should get a nice traditional star like your program from Part 1. Experiment with Sides = 50 and different values of Rotations to get pictures like the ones in the screenshot below. Values of Rotations that are not factors of 50 work best.



c) Write the pseudo-code for drawing Crazy Stars and then get a demonstrator to check your pseudo-code before going on to the next step.

d) Now extend the application using the pseudo-code.

e) Test the application with different inputs to make sure the results are what you expect.

Pseudo-code:

Week 5: Practical REVIEW PAGE Name: _____

Questions: Complete the following questions. (Note: this must be done before you seek a verification).

1. What are the advantages of using repetition when writing program code?

2. Write some code which will display a launch counter to the Output window using the **`Console.WriteLine`** method, counting down from 10 to 1 and ending in "Blastoff!".

Verification: To assess your competency of this material your demonstrator will verify that you have:

1. Created the required applications.
2. Answered the set questions.

Week 6: Practical

Algorithmic Problem Solving

This practical will require you to use all the programming concepts taught so far to solve algorithmic problems. Read the entire practical before you begin creating your application.

Exercise 1

To begin, create a **Week6** folder in your **101** folder. Copy the **Week6-Ex1** folder from the COMPX101 library folder to your **Week6** folder or download it from Moodle.

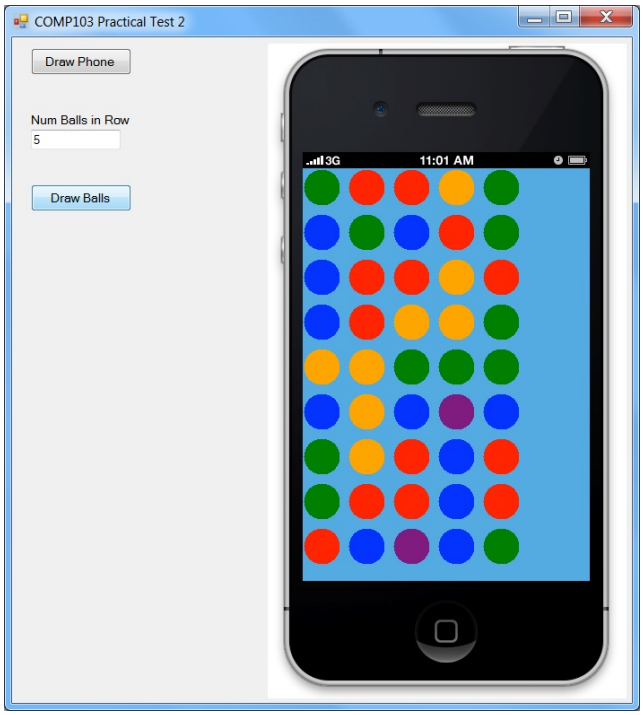
Lotto is looking at developing a new mobile app, and they have asked you to help develop the prototype. You are to build a program to draw a grid of lotto balls on a smart-phone display. The program will ask the user to type in the number of lotto balls in a row. The program will have a **`Draw Phone`** button that will draw the phone. The program will also have a **`Draw Balls`** button that will draw a grid of lotto balls on the phone display. The grid will have the user specified number of lotto balls in a row, and as many rows (and gaps) as will fit in the display height. We will use a coloured ellipse to represent a lotto ball.

The program should check if the number of lotto balls in a row that the user has entered will fit in the display width, i.e. (`BALL_SIZE` + `GAP_SIZE`) * number of balls in a row is less than `DISPLAY_WIDTH`. If it does, then the program should calculate how many rows of balls can be displayed down the display height i.e. number of rows = `DISPLAY_HEIGHT` / (`BALL_SIZE` + `GAP_SIZE`). Then it should draw the grid of balls in the smart-phone display, starting at the top left of the display area, as specified by the constants i.e. the first ball starts at position `DISPLAY_LEFT, DISPLAY_TOP` and is of `BALL_SIZE` and each following ball in that row should be to the right, separated by a gap of `GAP_SIZE`.

If the number of columns entered by the user is too large, then no balls should be drawn, and a suitable error message should be displayed. Any other errors should be handled gracefully.

A form is provided, which already has a picturebox and a **`Draw Phone`** button. You need to add buttons for **`Draw Balls`**, **`Clear`** and **`Exit`**, and a textbox for the user to enter the number of lotto balls in a row.

Example Screenshot



Tasks

1. Write the pseudo-code for drawing the rows of 5 balls in the picturebox, based on the users input of the number of lotto balls in a row. Then get the demonstrator to check your pseudo-code before continuing.

2. Create the user interface to match the specifications above.

3. Create the click event method for the **`Exit`** button, which exits the form.

4. Create the click event method for the **`Clear`** button, which clears the picturebox, clears the contents of the textbox and gives it the focus.

5. Create the click event method for the **`Draw Balls`** button, which will implement your algorithm from task 1 to draw the balls.

Hint: *Concentrate on just drawing a row of balls, at the top left corner. Then when this is working, add in the gaps, then move the row to start at the top left of the display area, then work on drawing multiple rows.*

Hint: *Constants have been provided for you so use them where appropriate.*

6. Lotto has 5 colours of balls: Blue, Orange, Green, Red, and Purple. Change the code to fill the balls with a random lotto colour. That is, for each ball, generate a random number from 1 to 40. If the number is 1 to 9, then colour the ball Blue, if the number is 10 to 19 then colour the ball Orange; 20 to 29 is Green, 30 to 39 is Red and 40 is Purple.

Exercise 2

You must complete Task A and one other task (feel free to do all of the tasks for more practice).

Task A

Copy the **Week6-Ex2A** folder from the COMPX101 library folder to your **Week6** folder or download it from Moodle.

Nilesh has designed a new board game called Smack! He wants to create a computer version of this game and since he is lazy he wants you to create an application which will draw the board for his game.

The user should be able to specify the number of rows to draw for the board. *The board will always have 10 columns in each row.* The number of rows of the board should be between 5 and 10 inclusive. The first and last column on each row are to be light pink, columns 5 and 6 are to be light blue and all other columns are to be light green. See the screenshot provided in the template folder.

A form and picturebox is provided for you to work on. You will need to add buttons for **`Draw Board`**, **`Clear`** and **`Exit`**, and a textbox for the user to enter the number of rows to draw.

Hint: *Constants have been provided for you so use them where appropriate.*

a) Write the pseudo-code for drawing the Smack! Board and get a demonstrator to check it. The application will get the number of rows and then it checks to see if the value entered is between 5 and 10 inclusive. If the value is valid, the Smack! board is then drawn based on the user's input. If the value isn't valid, an appropriate error message is displayed. The application should calculate the width and height of a square based on the user's input and the size of the drawing area. Use the Try..Catch structure to deal with errors gracefully.

```
square width = width of picturebox / num columns
square height = height of picturebox / num rows
```

b) Create the user interface to match the specifications above.

c) Create the click event method for the **`Exit`** button, which exits the form.

d) Create the click event method for the **`Clear`** button, which clears the picturebox, clears the textbox and gives the textbox the focus.

e) Create the click event method for the **`Draw Board`** button based on your pseudo-code in Task 1.

Hint: *Try drawing one row of squares in a single colour first, then get it drawing multiple rows and then get it drawing the columns in the correct colours.*

Task B

You will create a **console application** which will allow the user to enter the number of stars to draw in a single row. It will then draw that number of stars with the first star replaced with a dash in the console window. The number also represents the number of rows to draw and the dash moves to the next position in each row. If any errors occur then an error message is displayed and the application will end. For instance, if the user enters 5, the following pattern is drawn in the console window:

```
-****
*-***
**-**
***-*
****-
```

a) Write the pseudo-code to draw the pattern and get a demonstrator to check it.

b) Create the console application to draw the correct pattern based on the user's input.

Task C

You will create a console application which will allow the user to enter the number of stars to draw in a single row. The number also represents the number of rows to draw. Every third row is drawn using the '=' character rather than a star. If any error occurs then an error message is displayed and the application will end. For instance, if the user enters 10, the following pattern is drawn in the console window:

```
**********
**********
==========
**********
**********
==========
**********
**********
==========
**********
```

a) Write the pseudo-code to draw the pattern and get a demonstrator to check it.

b) Create the console application to draw the correct pattern based on the user's input.

Pseudo-code for Ex 1:

```
Declare variables
Set up the paper to draw on the picturebox
TRY




CATCH
   DISPLAY an error message
   Clear number of rows textbox
   Give focus to the number of rows textbox
ENDTRY
```

Pseudo-code for two tasks from Ex 2:

Week 6: Practical REVIEW PAGE Name: _____

Questions: Complete the following questions. (Note: this must be done before you seek a verification).

1. Write the pseudo-code that will add up all the even numbers between 1 and 100 inclusive.

2. Write the C# code based on your pseudo-code for question 1.

Verification: To assess your competency of this material your demonstrator will verify that you have:

1. Created the required applications.
2. Answered the set questions.

Week 7: Practical

Methods and Menus

This practical will introduce you to using methods to make your code more efficient and how to use Menus in your application. Read the entire practical before you begin creating your application.

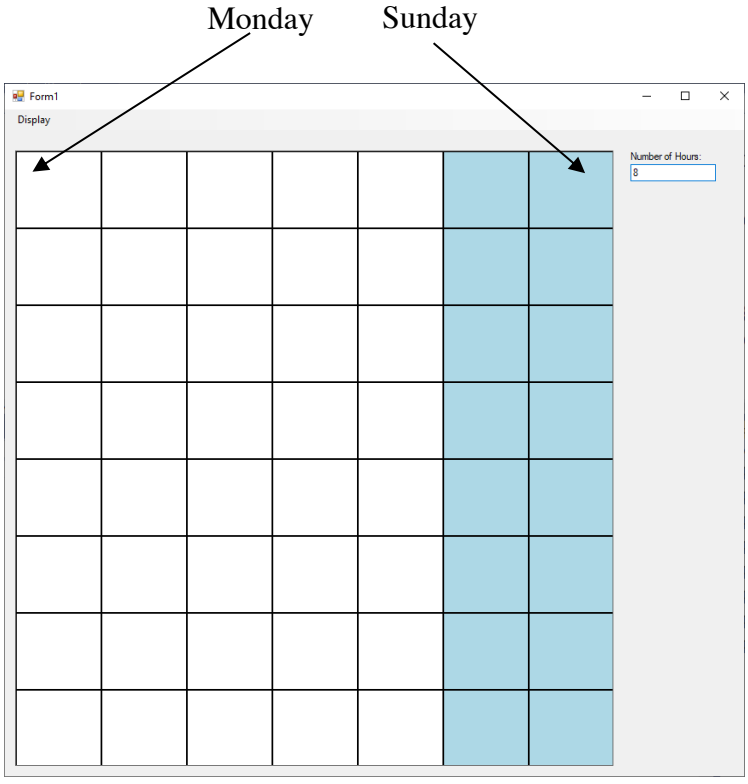
Reading

Before starting this practical, you should have read Chapter 6 of Gaddis. You should also have read the section on Menus in Appendix B of Gaddis (pages 739 – 745). Also watch the Panopto recording on creating menus for a more visual guide.

Exercise 1

Create a folder called **Week7** in your **101** folder. Copy the **Week7-Ex1** folder from the COMPX101 library folder to your **Week7** folder. Then open the solution.

You are helping to write a simple appointment planner application to display a weekly planner.



For this exercise you will create a grid of squares representing an hour for each day. Monday is day 1 and Sunday is day 7. The exercise will **not** involve interacting with the planner at all.

The user should be able to specify how many hours per day to display (i.e. the number of rows). It will check if the value entered is between 1 and 24 inclusive. If the value is valid then the planner is drawn based on the users input with weekday appointments drawn in White and weekend appointments drawn in Light Blue using methods. If the value isn't valid, an appropriate error message is displayed. The application should calculate the width and height of an appointment based on the user's input and the size of the drawing area. Use the Try..Catch structure to deal with errors gracefully.

```
appt width = width of picturebox / number of days
appt height = height of picturebox / number of hours
```

A form and picturebox are provided for you to work on. You will need to add a menu called **`Display`** with menu options inside it for **`Draw Planner`**, **`Clear`** and **`Exit`**, and a textbox for the user to enter the number of hours. Also give the menu options appropriate shortcut keys.

Hint: *Constants have been provided for you so use them where appropriate.*

a) Create the click event method for the **`Exit`** menu option, which closes the application.

b) Create the click event method for the **`Clear`** menu option, which clears the picturebox, clears the textbox and gives the textbox the focus.

c) Create the click event method for the **`Draw Planner`** menu option and create a Graphics object linked to the picturebox.

d) Create a method called **`DrawSquare`** which is passed a graphics object, the x and y position of the square, the width and height of the square and the colour of the square. It will then draw a filled in square in the appropriate colour using the values passed to the method. Also draw a black outline around the square.

Test your method by calling the **DrawSquare** method to draw a square at position 0,0 in the weekday colour in the **Draw Planner** event method. Then comment out the method call once you know the method is working.

e) Create a method called **DrawRow** which is passed a Graphics object, the current y position and the width and height of a square.

The method will draw a single row of squares using the pseudo-code given below:

```
Method DrawRow(paper, y pos, square width, square height)
   Declare x variable
   FOR each day/column to draw
      IF day >= Saturday THEN
         Call DrawSquare with weekend colour
      ELSE
         Call DrawSquare with weekday colour
      ENDIF
      Shift x to the right by width of square
   ENDFOR
ENDMETHOD
```

Test your method by changing the **Draw Planner** event method calculate the width and height of a square and store the values in appropriate variables. Then draw a single row of the planner at y position 0.

f) Now extend the **Draw Planner** event method to draw the entire planner using the pseudo-code given on the next page:

```
Declare variables
TRY
   GET the number of hours
   IF number of hours is valid THEN
      FOR each hour to draw
         Call DrawRow
         Shift y down by height of square
      ENDFOR
   ELSE
      Display error message
      Clear the board size textbox
      Give the board size textbox the focus
   ENDIF
CATCH
   Display error message
   Clear the board size textbox
   Give the board size textbox the focus
ENDIF
```

Optional Extra

g) Write a method called **`GetColour`** which is passed the current day number and the method returns back the correct colour.

Modify the **`DrawRow`** method to use the **`GetColour`** method to determine the colour of a square.

Exercise 2

Option A: **ATM Simulator**. Create a new C# .NET project and save it in your Week7 folder.

You are to create an application that will simulate a simple ATM machine which allows the user to deposit and withdraw money. It will have a text box to allow the user to enter an amount and a label to display the current balance in the user's account. It will also have **`Deposit`**, **`Withdraw`** and **`Exit`** buttons.

a) Read the entire exercise and then design the interface. You may like to create a paper prototype but it is not worth any marks.

b) Declare a class variable to store the current balance of the user. The reason that it is a class variable, is that it needs to be accessed by many methods.

c) Write the pseudo-code for a method called **`CheckDeposit`** which is passed the deposit amount and returns back a boolean value which indicates whether the amount given is a valid amount.

If the deposit amount is less than $20 or greater than $200 then display an appropriate error message in a message window and **`false`** is returned back from the function. Otherwise **`true`** is returned back from the method.

d) Create the method using your pseudo-code in part c.

e) Write the pseudo-code for the **`Deposit`** button which gets the deposit amount and then calls the **`CheckDeposit`** function passing the deposit amount and storing the boolean value returned back.

If **`true`** is returned back then the deposit amount is a valid amount and should be added to the user's current balance. If **`false`** is returned back then nothing else needs to be done.

Finally the current balance is updated, the user input is cleared and the focus is given back to the control used to get the amount.

f) Create the click event method for the **`Deposit`** button using your pseudo-code in part e).

g) Write a method called **`CheckWithdrawal`** which is passed the withdrawal amount and returns back a boolean value which indicates whether the amount given is a valid amount.

If the withdrawal amount is less than $20 or is greater than the current balance then an appropriate error message is displayed in a message window and **`false`** is returned back from the method. Otherwise **`true`** is returned back from the method.

h) Create a click event method for the **`Withdraw`** button which gets the withdrawal amount and then calls the **`CheckWithdrawal`** function passing the withdrawal amount and storing the boolean value returned back.

If **`true`** is returned back then the withdrawal amount is a valid amount and should be subtracted from the user's current balance. If **`false`** is returned back then nothing else needs to be done.

Finally the current balance is displayed, the user input is cleared and the focus is given back to the control used to get the amount.

i) Create a click event method for the **`Exit`** button which will end the application.

j) Document your code appropriately and add your name and id number as comments at the top of your code.

Option B: **Beam Deflection.** Create a new C# .NET **Console Application** and save it in your Week7 folder. If you don't know how to do this then ask a demonstrator.

You will create and use methods to simplify the calculation of the beam deflection to give you practice in creating and using methods. You may need to review the problem description given in Week 6's practical.

You will notice the formula for the beam deflection can be broken down into 3 main parts:

$$y = \frac{Wx(l-x)}{24EIl}\left[l^2 + x(l-x)\right]$$

```
Part1 = Wx(l - x)
Part2 = 24EIl
Part3 = l² + x(l - x)
deflection (y) = (Part 1 / Part 2) * Part 3
```

The application will allow the user to run multiple experiments until they wish to exit. Each time the user wants to run an experiment they will need to get the load, elasticity, inertia and length values from the user using the console window. It will then call a method to calculate the deflection at increments of 5 along the length of the beam.

The following menu should be displayed to the user:

```
1) Run experiment
2) Exit
Please enter your selection:
```

Below is the incomplete pseudo-code for the Main method:

```
Call Menu returning option
WHILE option is not 2
  IF option is run experiment THEN
      Your pseudo-code goes here
  ELSE
      Display error message
  ENDIF
  Call Menu returning option
ENDWHILE
Call Pause
```

a) Write the complete pseudo-code for the Main method adding in the extra steps required for the full simulation. Get a demonstrator to check your pseudo-code before going on to the next step.

b) Write a method called **`Pause`** which just asks the user to press the return key and waits for them to do this.

c) Create a method to calculate the answer to Part 1, which is passed values for the load (W), distance from datum point (x) and beam length (l) and returns back the answer.

d) Create a method to calculate the answer to Part 2, which is passed values for the elasticity (E), moment of inertia (I) and beam length (l) and returns back the answer.

e) Create a method to calculate the answer to Part 3, which is passed values for beam length (l) and distance from datum point (x) and returns the answer back.

f) Create a method to calculate the beam deflection and return the deflection amount back. This method would call the previous methods to work out the answer. Pass the relevant values to this method to enable it to work out the beam deflection.

Note: You should not write to the console window from within this method.

g) Write the code for the **`Main`** method using your pseudo-code from part a).

h) Document your code appropriately and add your name and id number as comments at the top of your code.

Pseudo-code:

Week 7: Practical REVIEW PAGE Name: _____

Questions: Complete the following questions. (Note: this must be done before you seek a verification).

1. Explain the advantages to using methods in code?

2. Are there any disadvantages to using methods in code?

Verification: To assess your competency of this material your demonstrator will verify that you have:

1. Created the required applications.
2. Answered the set questions.
3. **Also that you can step into a method and step through the method using the debugger.**

Week 8: Practical

Text Files

This practical will introduce you to text files and how to manipulate information in them. Read the entire practical before you begin creating your application.

Reading

Before starting this practical, you should have read sections 5.6 and 5.7 on text files in Chapter 5 of Gaddis.

Exercise 1

Create a folder called **Week8** in your **101** folder. Copy the **Week8-Ex1** folder from the COMPX101 library folder to your **Week8** folder. Then open the solution.

You are going to create an application which will interpret a subset of the Logo programming language and perform the actions in a Logo program. Logo is commonly associated with the term "Turtle Graphics" which used a robot turtle which had a pen attached to the tail of the turtle. The turtle could turn left and right and could move one step at a time. If the turtle's tail was down when performing a step command, then a line would be drawn. By combining all of these actions into a logo program, complex graphics could be drawn.

Your application will read a logo program and perform the actions inside the program. Each line of the logo program only contains 1 logo command.

Methods for controlling the turtle in the application have already been provided for you.

Logo Commands

Left: turn the turtle's direction 90 degrees to the left
Right: turn the turtle's direction 90 degrees to the right
Tail: if the tail is up, put it down and vice versa
Step: Move the turtle 1 step in the current direction

a) Create a **`File`** menu with the following menu items:

```
Open Logo Program…
Clear
-
Exit
```

b) Write the click event method for the **`Exit`** menu item which will exit from the application. Write the click event method for the **`Clear`** menu item which will clear the picturebox.

c) Write the code for the **`Open Logo Program`** menu item which will set the filter for the dialog control and then ask for the filename of the logo program and check the user clicks on the Ok button. It will then read each logo command from the file and perform the command until the end of the file is reached. If a command is invalid then the command is written to the console window with an error message. The pseudo-code is given below:

```
Declare variables
Set filter for dialog control
IF user selects file and clicks on OK button THEN
   Open the selected file
   WHILE not the end of file
      Read line from the file
      IF command is Left THEN
         Call Left method
      ELSE IF command is Right THEN
         Call Right method
      ELSE IF command is Tail THEN
         Call Tail method
      ELSE IF command is Step THEN
         Call Step method
      ELSE
         Write error message to console window
      ENDIF
   ENDWHILE
   Close the file
ENDIF
```

h) Document your code appropriately and add your name and id number as comments at the top of your code.

Exercise 2

Option A: **Transaction Processing.** Create a new C# .NET solution and save it in your **Week8** folder.

Copy the files, **small_weapons.txt** and **large_weapons.txt** from the **COMPX101** library folder into your **Week8** folder or download them from Moodle. For testing purposes you should use the small file. Use the large file when you think the application works correctly. To see what is in the files just open them in Wordpad or Notepad, don't use Excel.

Nilesh is currently enjoying the action RPG game Torchlight 2 which is an awesome game and totally blows Auction House Simulator 3, oh sorry, that should be Diablo 3, out of the water. He has got a file containing info of some of the unique weapons in the game.

The transaction file contains the following information:

Weapon Name (string)
Weapon Type (string)
Damage (int)
Weapon Speed (double)
.
.
.

To tell if one weapon is better than another you need to know the Damage Per Second (DPS) the weapon does, since weapons have a different attack speed. DPS is calculated by taking the damage value and dividing it by the attack speed.

You will write an application that will allow Nilesh to load the file and display the weapon info and DPS of every weapon in the input file.

When the user chooses to open a weapon file, they are required to choose the file using the open file dialog window. It will then read in the 4 values about a particular weapon and then display the 4 values to the console window on one line neatly padded along with the DPS of the weapon. This is repeated until the end of the file.

Once the file has been processed, display a message to the user and also say how many weapons were processed. The application just needs two buttons, a **`ProcessFile`** button and an **`Exit`** button.

a) Write the pseudo-code for processing a weapon file. Get a demonstrator to check your pseudo-code before going on to the next step.

c) Write a method called **`CalculateDPS`** which is passed the damage value and speed value and returns back the DPS value as a double.

d) Write the code for opening a scan file and processing it based on your pseudo-code in a). Use the **`CalculateDPS`** method to calculate the DPS of each weapon.

e) Modify your code from d) so that the weapon information written to the console window is also written to a text file. The user should be able to specify the name of the file. Add a header to the beginning of the file which has column headers for each column. At the end of the text file display the total number of weapons in the file.

f) Document your code appropriately and add your name and id number as comments at the top of your code.

Note: Weapon information used in the files has been adapted from the information found at:

`http://torchlight.wikia.com/wiki/Great_Weapons_(T2)`

Torchlight 2 Website:

`http://www.torchlight2game.com/`

Grim Dawn Website (an ARPG to keep and eye on):

`http://www.grimdawn.com/`

Option B: **Gravitational Forces.** Create a new C# .NET **console application** and save it in your **Week8** folder.

Open the file, **planets.txt** from the **COMPX101** library folder in Wordpad or Visual Studio. You should be able to see the structure of the file, close it when you are finished.

Note: This exercise has been adapted from the following document: `http://hurri.kean.edu/dept/murphy/1100dir/homework-labs/Astronomy-lab-planets.doc`

The surface gravity of a planet depends on both its mass and its size. We can calculate the value of the gravitational acceleration, g_p, for the planet from Newton's Law of Universal Gravitation as:

$$g_p = G\ M_p / R^2$$

Where M_p is the mass of the planet, and R its radius. Hence, the gravitational acceleration of the earth is:

$$g_p = G\ M_p / R^2$$

$$g_p = (6.67 \times 10^{-8}\ \text{dyne cm}^2/\text{g}^2)(5.98 \times 10^{27}\ \text{g})/(6.372 \times 10^8\ \text{cm})^2$$
$$g_p = 980\ \text{cm/s}^2$$

We would like to get an idea of the gravitational acceleration (gravity) of other planets. To simplify our calculations let us express the gravity of the other planets in terms of the gravity of the earth, hence the gravity of the earth will be 1. Therefore, if the gravity we calculate for another planet is 3.5, that means the planet has a gravitational acceleration 3.5 times that of the earth (i.e. if you weigh 100 kg on the earth you will weigh 350 kg on this planet).

We can simplify the calculation of gravity on other planets if we express the mass of the planet in earth masses, and the radius of the planet in earth radii. For example, what is the gravity on the moon in earth gravity units? The mass of the moon is 7.35×10^{25} g, and the radius of the moon is 1.74×10^8 cm.

The mass of the moon in earth masses is: $M_m = 7.35 \times 10^{25} / 5.98 \times 10^{27} = 0.012$ and the radius of the moon in earth radii is $R_m = 1.74 \times 10^8 / 6.372 \times 10^8 = 0.27$.

Hence the gravitational acceleration of the moon in earth gravity units is

$g_m = 1\ (0.012) / (0.27)^2 = 0.165$

Our calculation shows that the acceleration of gravity on the Moon is 0.165 or 16.5 % of that on earth. If you weigh 100 kg on the earth then you would only weigh 16.5 kg on the Moon.

You are to write an application which will read the mass (in earth masses) and radius (in earth radii) of a planet from a file and display the information with the gravitational acceleration (in earth gravity) to the console window. This is repeated until the end of file is reached.

The **`planets.txt`** file contains the planet name, mass (M_e) and Radius (R_e) on separate lines. For example:

```
Venus
0.81
0.95
Mars
0.11
0.53
...
```

a) Write the pseudo-code for the Main method. Get a demonstrator to check your pseudo-code before going on to the next step.

b) Create a method called **`CalcGravity`** which is passed a mass value and a radius value and returns back the gravitational acceleration for that planet.

c) Write the code for the Main method based on your pseudo-code in a). Use the method in b) to calculate the gravity value.

Note: Enter **`g:\COMPX101\planets.txt`** as the path and name of the input file.

d) Modify your code from c) so that the planet information written to the console window is also written to a text file. The user should be able to specify the path and name of the file using the console window.

Option C: **Graphing Application.** Create a new C# .NET solution and save it in your **Week8** folder.

Copy the files, **small_drink_transactions.txt** and **drink_transactions.txt** from the **COMPX101** library folder into your **Week 7** folder or download them from Moodle. For testing purposes you should use the small transactions file. Use the main transactions file when you think the application works correctly. To see what is in the files just open them in Wordpad or Visual Studio.

Bluebird vending operates drink vending machines which sell a variety of bottles of drink. Each time a drink is sold the vending machine adds the drink sold to a transaction file stored in the internal memory. The vending machine is re-filled each week and the transaction file is downloaded.

The manager wants you to write an application which will process the transaction file to sum up the number of bottles of each type of drink sold and then display the number sold as a bar graph in a picturebox. The X-axis denotes the type of drink and the Y-axis denotes the number sold. There are only 4 types of drink sold, coke, pepsi, sprite, 7up.

Each line in the transaction file contains the name of the drink sold, for example:

```
pepsi
coke
pepsi
sprite
pepsi
7up
...
```

The application should allow the user to open the transaction file to process, clear the graph, and exit from the application. You can use a menu or buttons to allow the user to perform these tasks.

a) Write the pseudo-code for opening a transaction file. Get a demonstrator to check your pseudo-code before going on to the next step.

b) Create the event methods which will clear the graph and exit from the application.

c) Create a method called **`CalcBarHeight`** which is passed the number of drinks sold and returns back the height of the bar in the graph. You will need to use the height of the picturebox and will need to scale the height appropriately.

d) Write the code for opening a transaction file based on your pseudo-code from a). Use the method from c) to calculate how high each bar of the graph should be.

You are not required to label your graph with text to get the practical marked but if you want to try to add text to label your graph you will need to use the **`DrawString`** method of the Graphics object. See if you can figure out how to do this if you have time.

Pseudo-code:

Week 8: Practical REVIEW PAGE Name: _____

Questions: Complete the following questions. (Note: this must be done before you seek a verification).

1. What does the term 'appending to a file' mean?

2. Why is using the Try…Catch structure in your code important?

Verification: To assess your competency of this material your demonstrator will verify that you have:

1. Created the required applications.
2. Answered the set questions.

Week 9: Practical

CSV File Processing and Arrays

To introduce you to the concept of arrays. You will learn how to declare arrays, how to manipulate 1 element of an array and how to manipulate the whole array. You will also learn how to process a Comma Separated Value (CSV) file. Read the entire practical before you begin creating your application.

Reading

Before starting this practical, you should have read sections 7.1 up to and including 7.8 and sections 8.1 and 8.2 of Gaddis.

Exercise 1

Create a folder called **Week9** in your **101** folder. Copy the **Week9-Ex1** folder from the COMPX101 library folder into your Week9 folder. Open the project in Visual Studio.

Tim recently bought a fitbit pedometer that keeps track of how much exercise he does. It outputs the data to a CSV file in the following format: **Date, Calories Burned, Steps, Distance, Minutes Sedentary (Inactive), Minutes Lightly Active, Minutes Fairly Active, Minutes Very Active, Activity Calories.**

A data files has been provided, **`fitbit-export.csv`** which contain valid data, and **`bad-fitbit.csv`** which is used to test task 5.

Here are the first few lines of the file **`fitbit-export.csv`**:

```
01-03-2014,2766,5589,4.24,1165,164,111,0,1110
02-03-2014,2511,3706,2.81,1275,93,72,0,683
03-03-2014,2831,7886,5.99,1228,87,106,19,1060
```

You are to write a program that will read in a CSV file of fitbit data, process this data and display the data using graphs and a listbox.

Some constants and methods have already been provided for you. Read the comments for the methods to see what they do.

You must create a **File** menu with the following items:

```
Open File…
Clear
-
Exit
```

a) The pictureboxes and listbox have been provided for you (*the second picturebox will be used in Week 10s practical*). Create the **File** menu as specified above also assigning appropriate shortcut keys.

b) Write the click event method for the **Exit** menu item which will exit from the application. Write the click event method for the **Clear** menu item which will clear the listbox and clear the first picturebox.

c) As practice, write the pseudo-code for the **Open File…** menu item as described in tasks e) to j) below.

d) Create any extra constants that could be used in the application.

e) Create the click event method for the **Open File…** menu item and then write the C# code as described below:

The Open File menu item will ask for the name of the input file and check that the user clicks on the OK button of the dialog control. It will then open the file and read in the file until the end of file is reached. For each line it reads from the file, the program will split it and then extract the values from the string array into separate variables. It will then display the information in the listbox lined up in columns.

Note: **You DO NOT need to store the data into arrays. Just extract the values into separate variables.**

Hint: *Use the* `PadRight` *or* `PadLeft` *methods to line up the text.*

f) Modify your **Open File…** menu event method so that it checks for bad data in the file, such as the correct number of elements (commas) on a line, or the numbers parsing correctly. If it doesn't have the correct number of elements, or catches an exception then write the raw line to the console window. Test this using the **bad-fitbit.csv** data file.

g) Extend your **`Open File...`** event so that while reading in the data, it adds up all the **steps** values, so that at the end of the file it can display the number of steps taken that month in a MessageBox.

h) Write a method called **`CalculateStepsPerMetre`** that accepts an integer, which is the number of steps, and a double, which is the distance walked in kilometres, and returns a double, which is the number of steps Tim walks per metre, i.e. number of steps / (distance * 1000).

i) Extend your **`Open File...`** event so that it calls your **`CalculateStepsPerMetre`** method after it has successfully converted the data. Alter the display from task e) to also display the steps/m value (to 3 decimal places) as an extra column after the active calories.

j) Extend your **`Open File...`** event so that it displays the **distance** walked in the top picturebox as a bar graph, by drawing a bar for each day using the **`DrawBar`** method provided. The X position of the bar is shifted across by `BAR_WIDTH` after each bar is drawn, i.e. `X += BAR_WIDTH`, and the height of the bar is the **distance** walked * `SCALE_FACTOR`.

Start off with the Y position = 0, so all the bars hang down, then change the Y position so all bars 'rise' from the bottom of the picturebox. See the example screenshot in the template folder.

Exercise 2

Option A: **Student Grades.** Copy the **Week9-Ex2OptionA** folder from the COMPX101 library folder into your Week9 folder.

The files **small_marks.csv** and **large_marks.csv** are provided in the template folder for testing.

You are to write an application which reads information about student exam marks. The file contains the student id number followed by the exam mark in csv format.

Open the files in Wordpad to see the structure of the files. The file small_marks.csv just contains a few lines to make it easier to test your application as you are creating it.

The user should be able to Load a marks file and display the id number and mark for each student in a listbox, graph the marks in a picturebox, and generate a report showing the student id, mark and letter grade for each student.

The application should have buttons or menu items to Load Mark File, Graph Marks, Clear Graph, Generate Report and Exit.

The maximum number of students in a marks file is 100.

a) The code for opening a file and loading the data into the two arrays has been provided. Study the code and make sure you understand how it works.

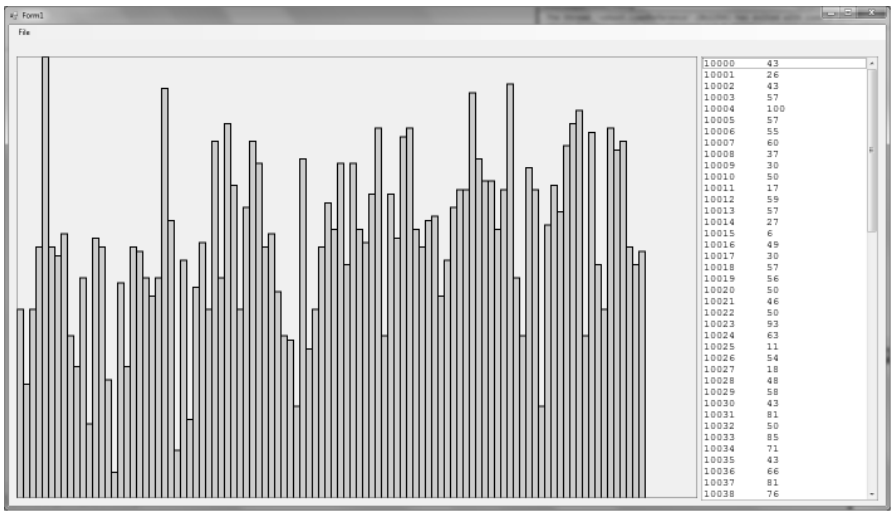
b) Write the code for the **`Clear Graph`** button or menu item which clears the picturebox and the **`Exit`** button.

c) Write the code for loading a marks file using your pseudo-code from a). Use the Try/Catch structure to deal with any errors which may occur.

d) Write a method called **`CalculateBarHeight`** which is passed a mark and returns the height of the bar for the graph as an integer.

Hint: Bar Height = height of picturebox * mark / 100

e) Write the pseudo-code for the Graph marks task which will go through each mark that has been stored in the arrays and draws it's bar on the bar graph. All bars should start at the bottom of the picturebox. See the picture below



f) Write the code for the **`Graph Marks`** button or menu item based on your pseudo-code.

g) Write a method called **`CalcLetterGrade`** which is passed a mark out of 100 and returns back the correct letter grade based on the table below:

```
A      80-100
B      65-79
C      50-64
D      35-49
E      0-34
```

h) Write the code for the **`Generate Report`** button or menu item which will write to a textfile the id, mark and letter grade of each student. The report should have a header for each column and a footer at the end stating the number of students in the report.

i) Document your code appropriately, with comment headers for each method, and add your name and id number as comments at the top of your code.

Option B: **Waikato Weather**. Create a new C# .NET **console application** and save it in your **Week9** folder.

The files **weatherData.csv, smallWeatherData.csv** and **hugeWeatherData.csv** are in the **COMPX101** library folder or download them from Moodle if you are working from home.

The weatherData.csv file contains the maximum and minimum temperature values recorded each day from 1/1/2009 to 30/4/2009 for the Waikato region. The small weather data file contains a few lines of data you can use for testing purposes. The huge weather data file contains data for all of 2008.

You are to create a console application to do some simple processing on the data and find out the highest and lowest recorded temperature values during that period. The application will display the following menu:

```
1) Load File
2) Display Data
3) Find Highest Temperature
4) Find Lowest Temperature
5) Generate Report
6) Exit
Please select option:
```

a) Create a method called **`Menu`** which displays the above menu and returns back the option entered.

b) The **`LoadFile`** method has been created for you. Study the code carefully and make sure you understand how it works.

c) Modify the main method to display the menu and then repeat until the user wants to exit. Inside the loop display the menu again. Test that you can enter multiple options (which do nothing at this stage) and that you can exit correctly.

d) Create a method called **`DisplayData`** which will display the high and low value on one line to the console window. It should do this for all valid index positions.

Hint: Use your index variable to tell the program when to stop repeating.

e) Write the pseudo-code that will find the highest temperature value in the data by using a loop to iterate through the data.

f) Write a method called **`FindHighestTemperature`** which is not passed any values and returns back the highest temperature recorded. Write the code for the method using your pseudo-code.

g) Write a method called **`FindLowestTemperature`** which is not passed any values and returns back the lowest temperature recorded. It will need to loop through the low temperature array keeping track of the lowest value found and then return the lowest value found when the loop is finished.

h) Modify your main method so that the methods above are called when required and the high and low values are displayed to the user in the console window.

i) Create a method called **`CalcAverageTemp`** which is passed a high and a low temperature value and returns back the average temperature.

j) Create a method called **`GenerateReport`** which has the following algorithm:

```
Get path and filename of report file
Open report file
Write headers to file
FOR each valid temperature value
 Call CalcMeanTemp with high and low temp returning
 mean temp
 Write high, low and mean temp to file
ENDFOR
Call FindHighestTemperature returning highest temp
Call FindLowestTemperature returning lowest temp
Write the highest temperature to file
Write the lowest temperature to file
Close report file
```

The information should be neatly formatted with the data lined up in columns (hint: use the **`PadRight`** or **`PadLeft`** methods) and the temperature values should be displayed to 1 decimal place.

Option C: **The Enigma Machine.** Create a new C# .NET solution and save it in your **Week9** folder.

"The Enigma machine was an advanced electro-mechanical cipher machine developed in Germany after World War 1. The Enigma machine was used by all branches of the German military as their main device for secure wireless communications until the end of World War 2. Several types of the Enigma machine were developed before and during World War 2, each more complex and harder to code break than its predecessors. The most complex Enigma type was used by the German Navy. In addition to the complexity of the Enigma machine itself, its operating procedures became increasingly complex, as the German military wanted to make Enigma communications harder to code break.

Various intelligence evidence during World War 2 led the German military to make several investigations about the possibility that The Allies can read Enigma messages. The German intelligence and communications experts concluded that Enigma was still secure from allied code breakers. They were wrong.

The basic operating procedure of the Enigma machine was simple. To send an encrypted message, the operator set the Enigma's electric and mechanical settings (the plug wirings and the rotor wheels) to a predefined initial combination known to him and to the receiving operator. Then he typed the free text message on the Enigma's keyboard. For each typed letter, a different letter was lit in the upper board. The operator wrote down each lit letter, so that when he finished typing the original message on the Enigma, he had a meaningless stream of letters, which was the Enigma-encrypted message.

He then transmitted the encrypted message with a standard Morse code radio transmitter. The receiving operator wrote the received encrypted message, set his Enigma machine to the same pre-defined combination, and then typed the message at the machine's keyboard. Typing the encrypted message on his Enigma machine with the same combination of settings deciphered it, so that the operator read the original free text message by the letters lit in the upper board as he typed." (Noy, U. *Enigma Machine*. Retrieved from http://www.2worldwar2.com/enigma.htm)

If you wish to learn more about the Enigma Machine and the allied code breakers at Bletchley Park just do a Google search on Enigma Machine and you will find tons of reference material.

For this practical you will implement an application which will simulate a simple 3 rotor enigma machine. Copy the files, **rotors.csv**, **SmallEncrypted.txt** and **LongEncrypted.txt** into your **Week10** folder or download them from Moodle.

Your Enigma application will have 3 buttons, **`Load Rotors`**, **`Decode File`** and **`Encrypt File`**.

a) Create 3 class scope arrays of type string to store each of the three rotors (large, middle and small). At each position of the array will be a single letter stored as a string. The arrays should just be empty as they will be created later when you split the CSV lines.

b) Create a click event method for your **`Load Rotors`** button which will read each of the three lines in the **`rotors.txt`** file. After reading each line, use the Split method to split the line into an array of strings using your arrays you created in b). **The first line in the file is for the large rotor, the second line is for the middle rotor and the last line is for the small rotor**.

c) Create a click event method for your **`Decode File`** button which will ask the user for the name of the file to decode and the name of the file to save the decoded message in using the Dialog controls.

Open the file to decode and read the text from the file using a single **`ReadLine`** statement into a string variable. Now you will need to loop from the beginning of the string (position 0) to the end of the string. At each character in the string you will eventually need to decode the character, but for now just display the current character in the string to the Output window to test whether you are progressing through the string properly.

You will need to use the **`Substring`** method of a string variable to get 1 character at the current position in the string. Check to see that the text in the output window matches the encrypted text in the file.

Hint: Use the **`Console.Write`** method to write a string to the output window but stay on the current line.

d) To decode an encrypted letter, find the letter in the large rotor and then store the letter at the same position on the middle rotor in a temporary variable. Then find the letter in the temporary variable on the large rotor and then store the letter in the same position on the small rotor. This letter is the decoded letter and should be returned back from the method. Write the pseudo-code to decode an encrypted letter and get a demonstrator to check it before going on to the next step.

e) Create a method called **`DecodeLetter`** which is passed a letter as a string and returns back the decoded letter as a string. Use your algorithm in part d).

f) Now modify your click event method for the **`Decode File`** button so that you decode the current letter from your encrypted

string and then display the decoded letter to the output window. You then need to shuffle the rotors by 1 position clockwise.

g) Create a method called **`ShuffleRotors`** which will first shuffle the position of all the letters on the small rotor by 1 in a clockwise direction. If the space gets to the end of the array then at the next shuffle the space goes back to the beginning of the array and then the middle rotor is also shuffled 1 position clockwise. The middle rotor is only shuffled when the small rotor has made a full rotation.

h) Now after displaying the decoded letter in the output window, call your **`ShuffleRotors`** method. Test your application with the **`SmallEncrypt.txt`** file and if it works try the **`LongEncrypt.txt`** file . You should now see the decoded message appear in the output window.

i) Now write the decoded letter to the file the user has specified for the decoded message. Your message should now appear in the decoded file, check that it works by opening your file in Notepad or Wordpad.

j) Create a method called **`EncryptLetter`** which is passed a letter as a string and returns back the encrypted letter as a string. To encrypt a letter find the letter on the small rotor and then store the letter at the same position on the large rotor in a temporary variable. Then find the letter in the temporary variable on the middle rotor and then store the letter in the same position on the large rotor. This letter is the encrypted letter and should be returned back from the method.

k) Now create the click event method for the **`Encrypt File`** method which gets the name of the file to encrypt and the name of the file to store the encrypted message. It then reads in the complete text from the message file and encrypts each letter and writes he encrypted letters to the encrypted file. To test it is working properly you should be able to decode the message using your **`Decode File`** button.

Try creating some encrypted messages and pass them to your friends and see if they can decode them.

Pseudo-code:

Week 9: Practical REVIEW PAGE Name: _____

Questions: Complete the following questions. (Note: this must be done before you seek a verification).

1. What are the benefits of using arrays?

2. If you wanted to store the name and age of 100 people what variables would you declare to store this information? Explain your answer.

Verification: To assess your competency of this material your demonstrator will verify that you have:

1. Created the required applications.
2. Answered the set questions.

Week 10: Practical

Lists

This practical will introduce you to using lists instead of arrays. Read the entire practical before you begin creating your application.

Reading

Before starting this practical, you should have read section 7.9 of Gaddis.

Exercise 1

Create a folder called **Week10** in your **101** folder. Copy the **Week9-Ex1** folder from your Week9 folder into your Week10 folder. Open the project in Visual Studio.

You will be extending the FitBit application to use lists to store the data and then graph the data differently.

a) Extend the file menu to add another menu item as shown below:

```
Open File…
Graph Days
Clear
-
Exit
```

b) Write suitable (class scope) lists to store the data from the fitbit file.

c) Extend the **Open File** menu event method so that after adding the information to the listbox it should store all the information into the appropriate lists.

d) Tim thinks there is a weekly cycle to his activity, so would like to see the data grouped so he sees all Mondays data together, and all Tuesdays data together, etc. This involves displaying the same graph from the first picturebox in the second picturebox, but with the order the data is drawn changed. Start at day 0 and draw every 7th day data after that (i.e. 0, 7, 14, 21, 28) then start at day 1 and draw every 7th data after that (i.e. 1, 8, 15, 22, 29), then start at day 2, and so on until you start from day 6.

Create the **`Graph Days`** menu event and write the code so that it draws the distances in the bottom picturebox in this new order.

Hint: *Use for loops that increment by 7 each loop to draw each day's distance. Put this in a loop that loops the start day from 0 to 6.*

e) To make seeing the days easier, extend task d) to use a different colour to represent each day of the week. See the example screenshot in the test folder.

Hint: *Use an array of colours so you can use the day loop counter to set a different colour for each day. I suggest you use the colours of the rainbow.*

Exercise 2

Create a new C# .Net solution and save it in your Week10 folder.

You will be creating a simple web browser application which will allow the user to book mark a webpage and also save and load book mark files.

The application will need to have a web browser control, a textbox to type in URL's and Back, Forward and Go buttons. Also add a textbox at the bottom of the form for the status. Make the **`Go`** button as the **`Accept`** button of the form. So when the user presses the Enter key it will activate the Go button.

Hint: Select the form in the design tab and then find the Accept Button property and set it to your go button event method.

The application will also need the following **File** menu with appropriate shortcut keys for each item:

```
New Bookmark File
Load Bookmark File...
Save Bookmark File...
Add Bookmark
-
Exit
```

a) Add the **WebBrowser** control to the form. Change the **Dock** property to **None** and then resize the browser control to what you want. Add the rest of your controls based on the description above.

Also add a listbox to the form to allow you to display bookmarked websites in the listbox.

b) Create click event methods for the **Back**, **Forward** and **Go** buttons and add the following code:

```
Back button method:       webBrowser1.GoBack();
Forward button method:    webBrowser1.GoForward();
Go button method:         webBrowser1.Navigate(textBoxURL.Text);
```

Note: You might have a different name for your WebBrowser control and URL textbox.

c) You should also add an event method for the "**Navigating**" event (not Navigated) of the WebBrowser control, to do this select the WebBrowser control and then in the Properties window click on the Lightning Bolt to change to events. Then double click on the Navigating event to create the event method in the code.

In the event method display the message "Loading…" into the status textbox. Whenever the browser control is navigating to a page it will execute this event method. This will inform the user when a page is loading. Ask the demo for help to create the event method if you have problems.

d) Add an event method for the **`DocumentCompleted`** event of the WebBrowser control, this event is executed when the web page has finished loading.

The event method should clear the status textbox and then sets the URL textbox to **`webBrowser1.Document.Url.ToString()`** to display the URL navigated that has been navigated to. It then sets the titlebar of the window to **`webBrowser1.DocumentTitle`** which is the title of the webpage that is displayed.

Now test your program. You should be able to type URLs into the text box, then click GO or press the enter key to visit that web site and navigate to any links inside the webpage. Test the back and forward buttons too.

c) Create a class scope list to store bookmarks for webpages. The list will store the URL's of any pages that have been bookmarked.

d) Create a method called **`UpdateListbox`** which clears the current contents of the listbox and goes through each bookmark in the bookmark list and displays the URL to the listbox until all the URL's are displayed. Then whenever you need to update the listbox you only need to call this method.

e) Create a method called **`Initialise`** which will clear the bookmarks list, clears the contents of the listbox and also clears the URL textbox.

f) Create the menu event method for the **`New Bookmark File`** menu item that will call the Initialise method to set the application back to its initial state.

g) Create the menu event for the **`Add Bookmark`** menu item that will add the current URL to the bookmark list and update the listbox. It should check that the URL textbox is not empty before adding the bookmark and updating the listbox.

h) Create a **`SelectedIndexChanged`** event method for the listbox control. Write the code that will allow the user to click on a URL in the list box and then the URL is retrieved from the bookmark list and displayed in the URL textbox and then navigate to that URL.

i) Create the menu event method for the **`Save Bookmark File...`** menu item which asks the user for the name of the file to save the information to using the SaveFileDialog control. It then goes through each URL in the bookmark list and writes each bookmark to the file on separate lines. Open the file in Notepad to see if you have written the information correctly.

j) Write the pseudo-code for the **`Open Bookmark File...`** menu item which asks the user for the file they want to open using the OpenFileDialog control. It should open the file and then initialise the application by calling the Initialise method.

It should then read each line from the file, storing the each URL into the bookmark list. After the file has been processed update the listbox to show all the bookmarks. Use the Try/Catch structure to deal with any errors which may occur. Get the demonstrator to check your pseudo-code before continuing.

k) Write the code for the **`Open Bookmark File...`** menu item based on your pseudo-code in task j).

l) Add a **`Delete`** button which removes the bookmark that has been selected in the listbox and then updates the listbox.

Week 10: Practical REVIEW PAGE Name: _____

Questions: Complete the following questions. (Note: this must be done before you seek a verification).

1. What are the advantages of using a list to store information rather than an array?

2. If you needed to store the name, age and weight of a class of fitness students, explain how this information would be stored.

Verification: To assess your competency of this material your demonstrator will verify that you have:

1. Created the required applications.
2. Answered the set questions.

Weeks 11 and 12: Practical

Problem Solving

This practical will involve using all the programming concepts in the paper to solve problems.

Reading

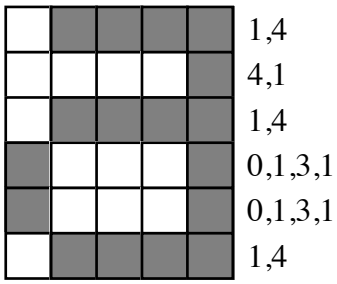
Before starting this practical, you might want to revise all of the chapters of Gaddis that we have covered so far in this paper.

Exercise

You only need to complete 1 option from the following to get your practical marked. You can do more than one if you have time and want to practice. For your option, you need to produce pseudo-code first.

Option A: **Run length coding.**

Possibly the simplest way to represent a black and white image is to associate black with 1 and white with 0. We then transform an image into "runs" of white and black pixels. For example,



Note that black pixels in the first column are represented as zero white pixels. This is to preserve the property: number of white, number of black, number of white, number of black, and so on. This method works well under the assumption that there will be large blocks of black (or white) pixels in an image — which happens to be true in most documents. This method of representation is typical in facsimile machines (they typically use 200 pixels per inch). In a fax machine we would read an image, encode it as a sequence of run-lengths, and then transmit the run-lengths to another fax machine which would decode the run-lengths back into the original image.

Write a command line program or a GUI program to allow the user to display an image encoded using run length coding. In the **RunLength** folder are some CSV files that contain images encoded using run length coding, i.e. each line of the file contains a list of numbers giving how many white, then black, then white, etc. The first line of the file is the image size i.e. *<width>,<height>*

For the command line program, use spaces and '#' characters to display the image. For the GUI program, use black or white 10x10 rectangles for each pixel.

Option B: Text Analysis.

For a long time, there has been much debate over whether Shakespeare really wrote some of his plays, or whether they may have been written by someone else, like Francis Bacon. We want to write a program that helps us do simple statistical analysis of some text to see if it is more similar to Bacon's writing style or to Shakespeare's style. We want to calculate the frequency of certain words, and the total number of different words that were used in a document (the vocabulary size).

There are several statistics we want to calculate so that we can perform a text analysis of the document. Write a command line program or a GUI program to allow the user to choose a filename and then analyse the text.

1. Count the number of lines

You will need to read each line in the file, count the number of lines, then display the result.

Example: **G:\COMPX101\Text\bacon1.txt** should have 50 lines.

2. Count the number of words in the document

To calculate this, you should split each line into an array of words (*Hint: use the Split method of the string class*), and sum up the number of words on each line to obtain the total number of words in the file. Be careful not to count empty strings as words. What characters separate words?

Example: **G:\COMPX101\Text\bacon1.txt** should have 635 words. Note that you may get different value, depending on what separators you used.

3. Count the number of times a given word appears

As you process the lines of the document, count the number of times that the word entered by the user appears. You should convert all words to lowercase before comparing them. What about punctuation? Display the number of times that it appears in the whole document and its percentage frequency (of all words counted in task 2).

Example: **bacon1.txt** should have 16 occurrences of "love", which is 2.52%. Again, results may vary depending on your choices of seperators.

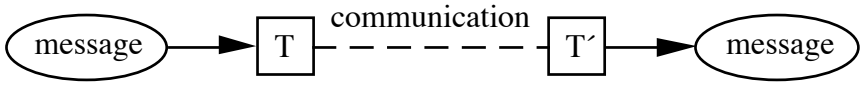
4. Count the size of the vocabulary

We also want to count the number of *distinct* words that the author used. That is, the number of different words that appear in the document. The number of distinct words divided by the total number of words gives us a measure of the richness of the writer's vocabulary. Hint: Use a **List<string>** called **distinct**, and add each word into this list, if it is not already there.

Example: **bacon1.txt** has a vocabulary size of 305 distinct words.

Option C: Statistical Attack.

The encryption process has three elements: a confidential message M that is to be stored or transmitted; a transformation T that converts M into an encrypted form; and another transformation T′ that restores the encryption back to the original message M.



One of the most simple transformations is to substitute each character of the message with another character. Using 27 letter English (i.e. the 26 letters of the alphabet plus the `SPACE` character, which we will represent with a '·') we could substitute each

character of the message with the *k*-th letter beyond it. When we reach the end of the alphabet, we add the `SPACE` character and start counting again from the letter `A`. By appending one `A` to the beginning of the message prior to its encoding, the receiver can deduce the value of k.

Consider the following secret message encoded using 27 letter English and the k-th substitution method described above.

```
EQIIXDQIDEXDXLIDGVMGOIXDKEQIDSRDWEXYVHEB
```

We know that the first character `E` corresponds to the letter `A` giving an offset of k=4. That is, `E` is the fourth letter after `A` in the alphabet. This allows us to construct the translation table below (Note that the `SPACE` character '·' translates to `W`, and the letter `D` translates to a `SPACE` '·').

```
code:  EFGHIJKLMNOPQRSTUVWXYZ·ABCD
text:  ABCDEFGHIJKLMNOPQRSTUVWXYZ·
```

Since the first character of the encoded sequence is not part of the original message, we can begin decoding at the first `Q`. We look `Q` up in the top row of the table and find by looking at the bottom row that it translates to the letter `M`. Next in the coded message are two `I`'s which translate into two `E`'s. After that there is an `X`, which translates to the letter T, and the `D` after that becomes a space '·'. Continuing with this algorithm gradually reveals the secret message.

```
QIIXDQIDEXDXLIDGVMGOIXDKEQIDSRDWEXYVHEB
MEET·ME·AT·THE·...
```

Encryption presumes that someone might try and read the message we are attempting to keep secret. A ciphered message invites attempts to decipher it. Such an attempt is called an "attack" on the encryption method.

The k-th substitution method is particularly vulnerable to attack as it takes no time at all to simply try all of the 26 possible offset values for k (note that k = 27 would not encrypt the message). We could make things a little tougher for the attacker by using a random substitution for each letter instead of a fixed offset. Such a method would allow us to replace 'A' with any of the other 26

characters, then replace 'B' with any of the remaining 25 characters, and so on. This might force the attacker to try each of the 26!=4033 million trillion permutations before stumbling upon the correct substitution. (Note: The actual number of permutations available is $1.0888 * 10^{28}$, but some of these would not encrypt the message very well, like replacing every letter with itself, or by uzing a zimilar letter az itz zubztitute).

Unfortunately, all basic substitution methods are easily attacked through a very simple statistical analysis. In English, the `SPACE` character is by far the most frequent in plain text. The next most frequent characters are `E`, `T`, `A`, `O`, and so on down to `J`, `Q` and `Z`. By substituting a `SPACE` in for the most frequent symbol in the coded message, `E` for the next most frequent, and so on, the attacker can get very close to deciphering the message on the very first attempt. The attacker can then use his or her knowledge of the language to make educated guesses about unknown characters (e.g. new ?ealand, cr?ptograph?). Using tricks such as these allow the attacker to greatly reduce the number of permutations that must be tried before the correct one can be found.

Assume that the characters below are listed in descending order according to their observed frequency in English text (i.e. SPACE '·' is the most frequent character, then `E`, then `S`, and so forth down to the least frequent character `X`).

```
· E S T O I A U B N M D H C G L R Y F P K V W Q J Z X
```

Decode the following message using the statistical attack described above (in message.txt).

```
DMBACHICDFDHDFEJAKBDMELAESABJNELFJOAGAKBCCGO

BAFCACFKTPBADEAHCBAIHDAFDAFCAGPCEAGAVBQRABGC

RANELBADEAIQBGUA
```

Remember, the frequency table is just a guide for English in general, and may not reflect the frequencies of these letters in this particular message. You may find that a little common sense will help you to make guesses that will save you a lot of work.

Write a command line program or a GUI program to allow the user to use a statistical attack to decode a message stored in a text

file. It should first count how many of each letter there are, then use the frequency table above to guess what each letter is substituted for. Then it should output a possible decoding of the message using this substitution. Some encoded message files are in the folder called **Code** for you to practice on.

Weeks 11&12: Practical REVIEW PAGE Name: _____

Questions: Complete the following questions. (Note: this must be done before you seek a verification).

1. What was it about your problem that made it hard to solve?

2 Do you think pseudo-code helped your solving of this problem?

3. What was the most important thing you learned in COMPX101.

Verification: To assess your competency of this material your demonstrator will verify that you have:

1. Written pseudo-code to help solve your problem in Exercise 2.
2. Created the required application(s).
3. Answered the set questions.