## EXERCISE 1

The tasks in the first part of this practical are step by step instructions as to how an read and write to files in C# and to use the string class and its operations

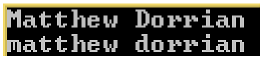
### TASK 1: String Operations

The String class provides a number of methods that can be used to create new strings from existing ones as the String class is Immutable (The contents of the string object cannot be changed after the object is created).

Key Methods include:

- **ToLower()** – Convert a string to all lowercase

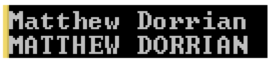
```
string name = "Matthew Dorrian";
Console.WriteLine(name);
string lowerCaseName = name.ToLower();
Console.WriteLine(lowerCaseName);
```



```
Matthew Dorrian
matthew dorrian
```

- **ToUpper()** – Convert a string to all uppercase

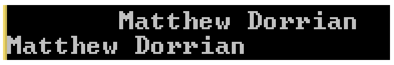
```
//ToUpperCase
string name = "Matthew Dorrian";
Console.WriteLine(name);
string upperCaseName = name.ToUpper();
Console.WriteLine(upperCaseName);
```



```
Matthew Dorrian
MATTHEW DORRIAN
```

- **Trim()** – Remove leading and trailing whitespace from a string

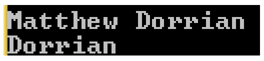
```
string spaces = "        Matthew Dorrian        ";
Console.WriteLine(spaces);
string trimmedName = spaces.Trim();
Console.WriteLine(trimmedName);
```



```
        Matthew Dorrian
Matthew Dorrian
```

- **Substring(int start)** – Gets the substring from the specified start position and the end of the string

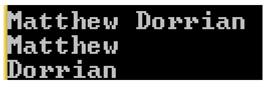
```
//Substring
string name = "Matthew Dorrian";
Console.WriteLine(name);
string surname = name.Substring(7);
Console.WriteLine(surname);
```



```
Matthew Dorrian
Dorrian
```

- **Spilt(char delimiter)** – Separate strings into a string array using the delimiter

```
//Spilt
string name = "Matthew Dorrian";
Console.WriteLine(name);
string[] spiltName = name.Split(' ');
foreach (var item in spiltName)
{
    Console.WriteLine(item);
}
```



```
Matthew Dorrian
Matthew
Dorrian
```

Now try these each of these operations on a string of your own.

## TASK 2: Using FileStreams to write to a file with a Console Application

**Step 1:** Open the Visual Studio File menu, and then select New (or press `Ctrl+Shift+n`). Then click on Project, select C# Console Application and name it `e.g. Prac08Task1`.

**Step 2:** We want to allow the user to enter text saved to a text file. To do this, open the Program.cs and prompt the user to enter in text using the Console as shown below:

```
//Allow user to enter in text to save to the file
Console.WriteLine("Enter text to be saved to file:");
string text = Console.ReadLine();
```

**Step 3:** Next we will create a string variable which will be the location where the text file will be saved to and the name of the file:

```
//location to save the file too
string location = "C:\\Users\\user\\Desktop\\streamtest.txt";
```

**Step 4:** We will create a FileStream object which will deal with reading and writing to a file. Once the FileStream is created we say specify the location of the file and the creation mode:

```
//Create filestream object
FileStream wFile;

//Create the filestream specifying the location and the file creation mode
wFile = new FileStream(location, FileMode.Create);
```

**Step 5:** Now that we have the FileStream created and is set up to write the file to the specified location, we will change the text inputted from the user and convert the data to a byte array to be used by the FileStream:

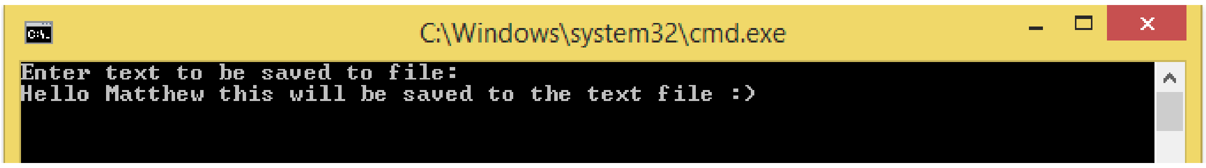
```
//Create a byte array
byte[] byteData;

//Create byte array using the text
byteData = Encoding.ASCII.GetBytes(text);
```

**Step 6:** Now it's time to create the file and to do this, we need to write the data to the FileStream:

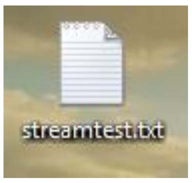
```
//write the data to the filestream which will be saved to the location
wFile.Write(byteData, 0, byteData.Length);

//Close the filestream
wFile.Close();
```

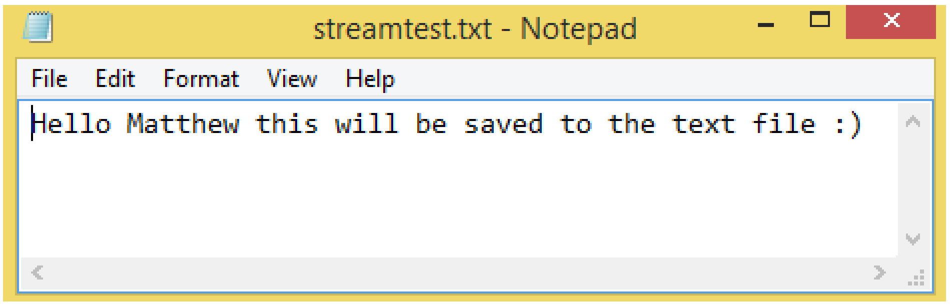
*Remember to Close the filestream as this releases the file to be accessed by other processes/programs

Example program is shown below:



```
Enter text to be saved to file:
Hello Matthew this will be saved to the text file :)
```





```
Hello Matthew this will be saved to the text file :)
```

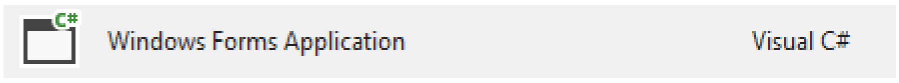
Unfortunately when dealing with files, exceptions can regularly occur e.g. access denied for the location. Therefore to be safe and to perform error handling, use a try/catch block with the FileReaders to ensure the program doesn't crash if an exception occurs:

```
//try catch method to ensure error handling e.g. permission issue with location to write to
try
{
    //location to save the file too
    string location = "C:\\Users\\user\\Desktop\\streamtest.txt";

    //Create filestream object
    FileStream wFile;

    //Create the filestream specifying the location and the file creation mode
    wFile = new FileStream(location, FileMode.Create);

    //Create a byte array
    byte[] byteData;

    //Create byte array using the text
    byteData = Encoding.ASCII.GetBytes(text);

    //write the data to the filestream which will be saved to the location
    wFile.Write(byteData, 0, byteData.Length);

    //Close the filestream
    wFile.Close();
}
//Catch any IOExceptions
catch (IOException ex)
{
    //Print error to console
    Console.WriteLine(ex.ToString());
}
```

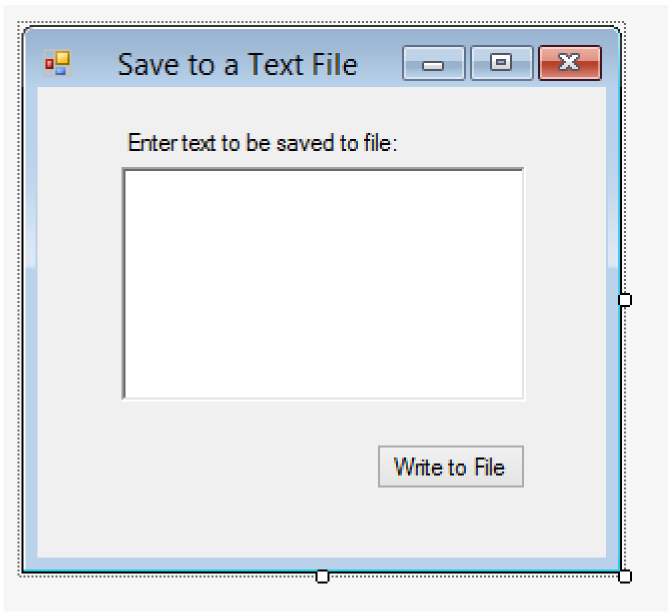
As you can see the catch will catch any IOExceptions and write the exception to the console

## Task 3: Using FileStreams to write to a file with a GUI

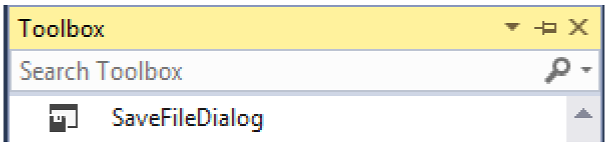
**Step 1:** Open the Visual Studio File menu, and then select New (or press `Ctrl+Shift+n`). Then click on Project, select C# Windows Forms Application and name it `e.g. Prac08Task2`.



**Step 2:** Create the following GUI using the toolbox to add elements to the form e.g. Button, RichTextBox. You can reposition any element on the screen by clicking and dragging the element to the desired location. Remember to use the Text and Name properties to correctly name your buttons and change the title of your window:



**Step 3:** Add a `SaveFileDialog` component using the toolbox and drag and place on the form. This is a non-visible component:



**Step 4:** Now that the UI has been set up, it is time to look at the code behind. We will first set up the properties of the `saveFileDialog` using the constructor for the form and set the dialog to save the file as text file (.txt) as default:

```
public Form1()
{
    InitializeComponent();
    saveFileDialog1.Filter = @"Text Files | *.txt";
    saveFileDialog1.DefaultExt = "txt";
}
```

**Step 5:** Double click on the button in design view which will automatically generate a method which will be called when the button is clicked. This method will make the saveDialog appear:

```
//Open the dialog
1 reference
private void WriteBtn_Click(object sender, EventArgs e)
{
    saveFileDialog1.ShowDialog();
}
```

**Step 6:** Next we need to create a method that will be called when the file location is selected and file name entered using the file dialog button and the Ok button is selected. To do this, navigate to the designer view and double click on the saveFileDialog which will be shown on the designer screen below the form:

saveFileDialog1

**Step 6:** This will automatically generate a method which will be called when the OK button is clicked. This method is where the text from the rich text box will be written to the file specified using the save dialog. First assign the filename from the saveDialog to a string variable to be used in the text file:

```
//location and file Name to save the file too
string fileName = saveFileDialog1.FileName;
```

**Step 7:** We will create a FileStream object which will deal with reading and writing to a file. Once the FileStream is created we say specify the location of the file and the creation mode:

```
//Create filestream object
FileStream wFile;

//Create the filestream specifying the location and the file creation mode
wFile = new FileStream(fileName, FileMode.Create);
```

**Step 8:** Now that we have the FileStream created and is set up to write the file to the specified location, we will change the text inputted from the user and convert the data to a byte array to be used by the FileStream. To ensure the new lines are persisted in the text file, replace the "`\n`" with `Environment.NewLine` using the string Replace method:

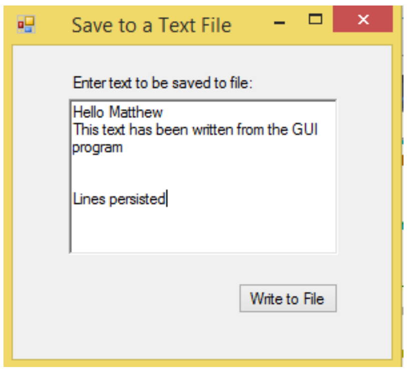
```
//Create a byte array
byte[] byteData;

//ensure new lines are taken in the text file
var text = richTextBox1.Text.Replace("\n", Environment.NewLine);

//Create byte array using the text
byteData = Encoding.ASCII.GetBytes(text);
```

**Step 9:** Now it's time to create the file and to do this, we need to write the data to the FileStream and clear the filename of the saveFileDialog :

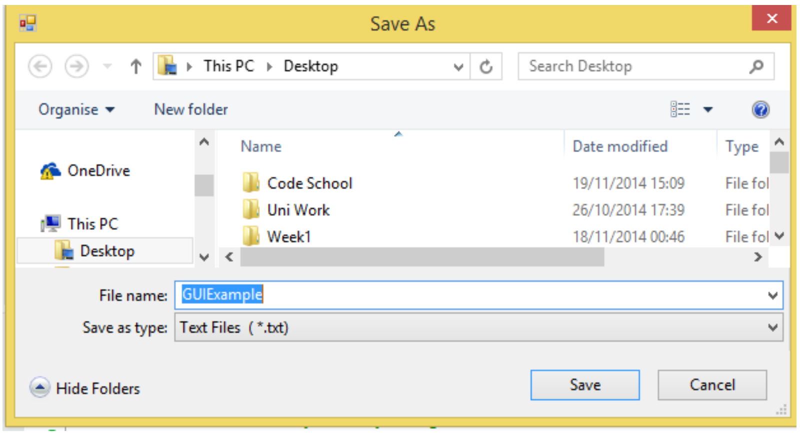
```
//write the data to the filestream which will be saved to the location
wFile.Write(byteData, 0, byteData.Length);

//Clear filename of dialog
saveFileDialog1.FileName = "";

//Close the filestream
wFile.Close();
```

*Remember to Close the filestream as this releases the file to be accessed by other processes/programs. Again due to the chance of exceptions, use a try/catch block with the FileReaders to ensure the program doesn't crash if an exception occurs.

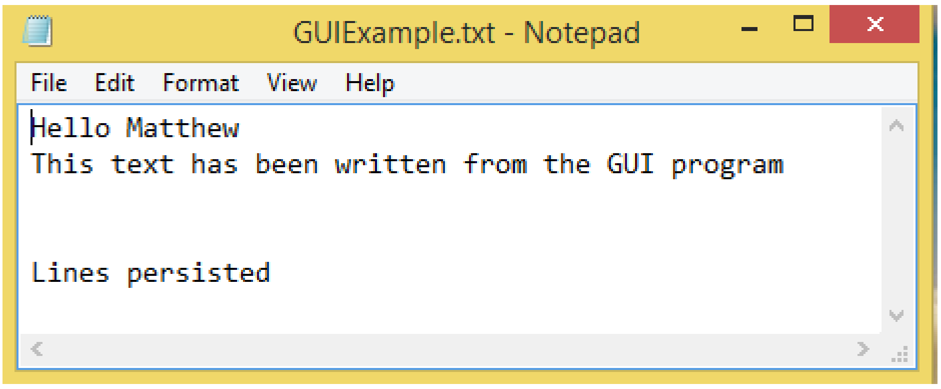
Example output shown below:



```
Hello Matthew
This text has been written from the GUI
program


Lines persisted
```



File name: GUIExample
Save as type: Text Files (*.txt)



```
Hello Matthew
This text has been written from the GUI program


Lines persisted
```

## Task 4: Using FileStreams to read from a file with a Console Application

**Step 1:** Open the Visual Studio File menu, and then select New (or press `Ctrl+Shift+n`). Then click on Project, select C# Console Application and name it `e.g. Prac08Task3`.

**Step 2:** We will create a string variable which will be the location where the text file will be saved to and the name of the file:

```
//location to load the file
string location = "C:\\Users\\user\\Desktop\\streamtest.txt";
```

**Step 3:** We will now create a FileStream which will read from a text file and print the content to the console:

```
//Open the filestream in read only mode
FileStream fileStream = new FileStream(location, FileMode.Open, FileAccess.Read);
```

**Step 4:** Next you will need to create a byte array setting the length of the array to the size of the file and set up variables which we need to read from the file:

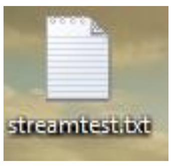
```
//Create byte array
byte[] buffer;

int length = (int)fileStream.Length;  // get file length
buffer = new byte[length];            // create buffer
int count;                            // actual number of bytes read
int sum = 0;                          // total number of bytes read
```

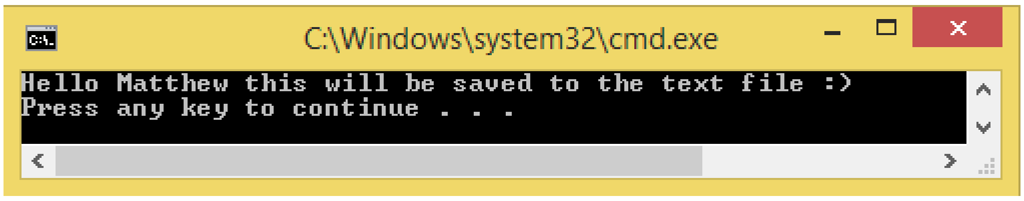
**Step 5:** We will now use a while loop and the `fileStream.Read` method to read from the file until all the stream has been read. We will then output to the console the contents of the byte array converted to a string using the Encoding class:

```
// read until Read method returns 0 (end of the stream has been reached)
while ((count = fileStream.Read(buffer, sum, length - sum)) > 0)
{
    sum += count;  // sum is a buffer offset for next reading
}

//Write the text out to console by converting byte array to string
Console.WriteLine(Encoding.Default.GetString(buffer));
```

*Remember to Close the filestream as this releases the file to be accessed by other processes/programs. Again due to the chance of exceptions, use a try/catch block with the FileReaders to ensure the program doesn't crash if an exception occurs.

Example output shown below:

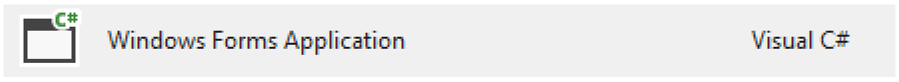




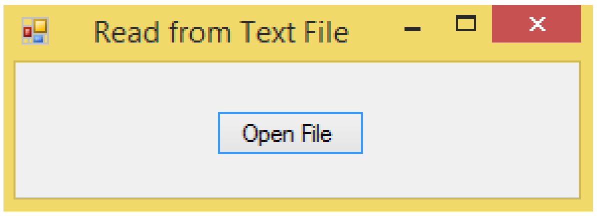
```
Hello Matthew this will be saved to the text file :)
Press any key to continue . . .
```

## Task 5: Using FileStreams to read from a file with a GUI

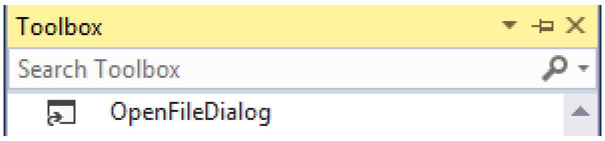
**Step 1:** Open the Visual Studio File menu, and then select New (or press `Ctrl+Shift+n`). Then click on Project, select C# Windows Forms Application and name it `e.g. Prac08Task4`.



**Step 2:** Create the following GUI using the toolbox to add elements to the form e.g. Button. You can reposition any element on the screen by clicking and dragging the element to the desired location. Remember to use the Text and Name properties to correctly name your buttons and change the title of your window:



**Step 3:** Add an `OpenFileDialog` component using the toolbox and drag and place on the form. This is a non-visible component:



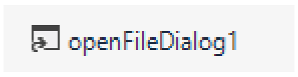
**Step 4:** Now that the UI has been set up, it is time to look at the code behind. We will first set up the properties of the `OpenFileDialog` using the constructor for the form and set the dialog to save the file as text file (.txt) as default:

```
public Form1()
{
    InitializeComponent();
    openFileDialog1.Filter = @"Text Files | *.txt";
    openFileDialog1.DefaultExt = "txt";
}
```

**Step 5:** Double click on the button in design view which will automatically generate a method which will be called when the button is clicked. This method will make the OpenDialog appear:

```
private void OpenFileButton_Click(object sender, EventArgs e)
{
    openFileDialog1.ShowDialog();
}
```

**Step 6:** Next we need to create a method that will be called when the file is selected using the Open File dialog button and the Ok button is selected. To do this, navigate to the designer view and double click on the OpenFileDialog which will be shown on the designer screen below the form:



**Step 7:** This will automatically generate a method which will be called when the OK button is clicked. This method is where the text from the text file will be read and displayed in a MessageBox. First assign the filename from the OpenDialog to a string variable to be used in the MessageBox:

```
//Get the location of the file and the actual file name to be used in the message box title
string location = openFileDialog1.FileName;
string fileName = Path.GetFileNameWithoutExtension(openFileDialog1.FileName);
```

**Step 8:** We will create a FileStream object which will deal with reading from a file. Once the FileStream is created we say specify the location of the file and the creation mode:

```
//Open the filestream in read only mode
FileStream fileStream = new FileStream(location, FileMode.Open, FileAccess.Read);
```

**Step 9:** Next you will need to create a byte array setting the length of the array to the size of the file and set up variables which we need to read from the file:

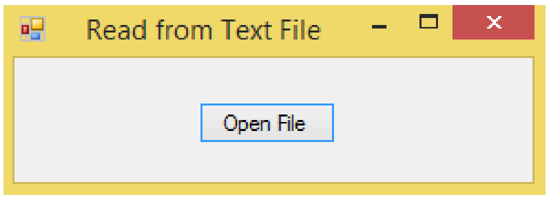
```
//Create byte array
byte[] buffer;

int length = (int)fileStream.Length;  // get file length
buffer = new byte[length];            // create buffer
int count;                            // actual number of bytes read
int sum = 0;                          // total number of bytes read
```

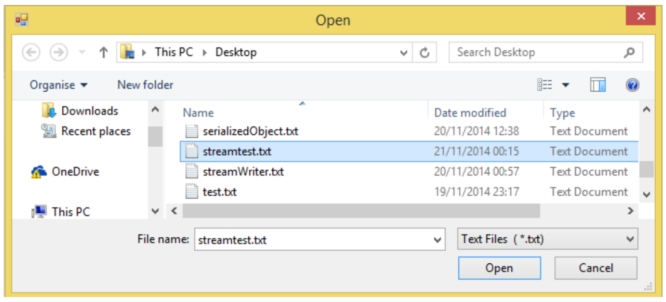
**Step 10:** We will now use a while loop and the `fileStream.Read` method to read from the file until all the stream has been read. We will then output to a MessageBox the contents of the byte array converted to a string using the Encoding class:

```
// read until Read method returns 0 (end of the stream has been reached)
while ((count = fileStream.Read(buffer, sum, length - sum)) > 0)
{
    sum += count;  // sum is a buffer offset for next reading
}

//Write the text out to a MessageBox setting the title to the file name by converting byte array to string
MessageBox.Show(Encoding.Default.GetString(buffer), fileName);
```

*Remember to Close the filestream as this releases the file to be accessed by other processes/programs. Again due to the chance of exceptions, use a try/catch block with the FileReaders to ensure the program doesn't crash if an exception occurs.

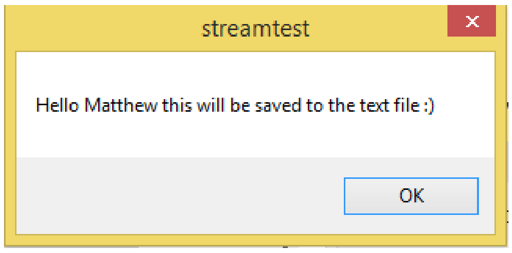
Example output show below:





| Name | Date modified | Type |
|---|---|---|
| serializedObject.txt | 20/11/2014 12:38 | Text Document |
| streamtest.txt | 21/11/2014 00:15 | Text Document |
| streamWriter.txt | 20/11/2014 00:57 | Text Document |
| test.txt | 19/11/2014 23:17 | Text Document |

File name: streamtest.txt
Text Files (*.txt)


streamtest

Hello Matthew this will be saved to the text file :)

## EXERCISE 2

### Task 6: Using BinaryWriters to write to a binary file

**Step 1:** Open the Visual Studio File menu, and then select New (or press `Ctrl+Shift+n`). Then click on Project, select C# Windows Forms Application and name it `e.g. Prac08Task5`.

**Step 2:** We want to write to a binary file an array of ints which will represent scores. To do this, open the Program.cs and create an array of type int that can store 10 numbers and assign numbers to each. You can create the array by hard coding the numbers or get the user to enter in each number.

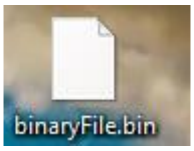
**Step 3:** We will create a `BinaryWriter` object which will deal with writing to a binary file. Once the BinaryWriter is created and specify the location of the file and the creation mode:

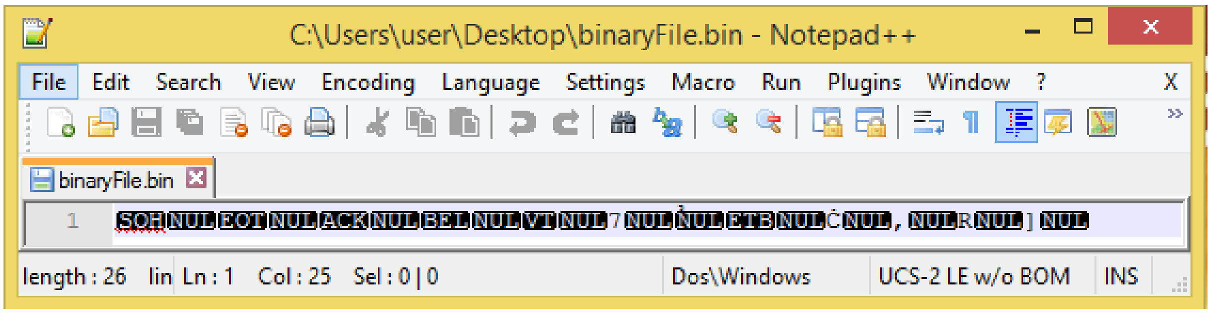
```
//Create a BinaryWriter and specify where to save the bin file and the creation mode e.g. create
BinaryWriter b = new BinaryWriter(File.Open("C:\\Users\\user\\Desktop\\binaryFile.bin", FileMode.Create));
```

**Step 4:** Now that the BinaryWriter is set up to write to a binary file to the specified location, we will write each score in the array to the file. To do this, use a `for` or a `foreach` loop and write each value:

```
//Loop through the score array
foreach (int score in scores)
{
    //write each score to the file in binary
    b.Write(score);
}
```

Example output shown below:




C:\Users\user\Desktop\binaryFile.bin - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ? X
binaryFile.bin

```
SOHNULEOTNULACKNULBELNULVTNUL7NULŃNULETBNULČNUL,NULRNUL]NUL
```

length : 26 lin Ln : 1 Col : 25 Sel : 0 | 0 Dos\Windows UCS-2 LE w/o BOM INS

*Remember to Close the BinaryWriter as this releases the file to be accessed by other processes/programs. Again due to the chance of exceptions, use a try/catch block with the BinaryWriter to ensure the program doesn't crash if an exception occurs.

## Task 7: Using BinaryReaders to read from a binary file

**Step 1:** Open the Visual Studio File menu, and then select New (or press `Ctrl+Shift+n`). Then click on Project, select C# Windows Forms Application and name it `e.g. Prac08Task6`.

**Step 2:** We want to read from a binary file to output to the console the scores. To do this, open the Program.cs and we will create a `BinaryReader` object which will deal with reading from a binary file. Once the BinaryReader is created, specify the location of the file and the creation mode:

```
//BinaryReader which will read a binary file from the specified location
BinaryReader b = new BinaryReader(File.Open("C:\\Users\\user\\Desktop\\binaryFile.bin", FileMode.Open));
```

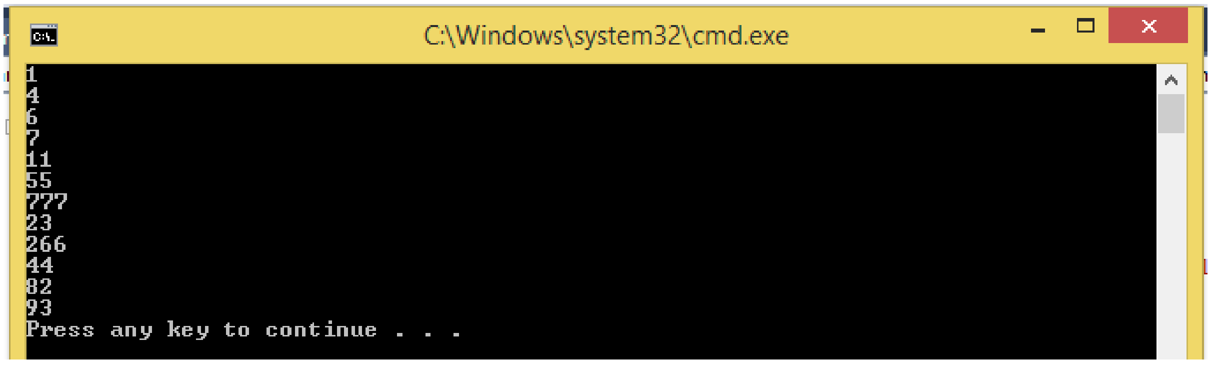
**Step 3:** Now that the BinaryReader is set up to write to a binary file to the specified location, we will create a position and length variable to be used to loop through the base stream:

```
// Position and length variables using the base stream.
int pos = 0;
int length = (int) b.BaseStream.Length;
```

**Step 4:** As the binary file contains only numbers, we will loop through the BinaryReader and read an integer and write the int to the console. We will increment the position variable by the size of an int (4) to loop through each number in the file:

```
while (pos < length)
{
    // Read integer.
    int v = b.ReadInt32();

    //Write number to console
    Console.WriteLine(v);

    // Advance our position variable.
    pos += sizeof (int);
}
```

Example output is shown below:


C:\Windows\system32\cmd.exe

```
1
4
6
7
11
55
777
23
266
44
82
93
Press any key to continue . . .
```

*Remember to Close the BinaryReader as this releases the file to be accessed by other processes/programs. Again due to the chance of exceptions, use a try/catch block with the BinaryReader to ensure the program doesn't crash if an exception occurs.

## Task 8: Using StreamWriters to write to a text file

**Step 1:** Open the Visual Studio File menu, and then select New (or press `Ctrl+Shift+n`). Then click on Project, select C# Windows Forms Application and name it `e.g. Prac08Task7`.

**Step 2:** We want to write to a text file using a `StreamWriter`. To do this, open the Program.cs and we will create a `StreamWriter` object which will write to a text file to the specified location. StreamWriter performs all the FileReader does in one statement as shown below:

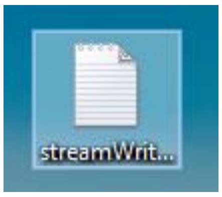
```
//Create a StreamWriter specifying the location to create the file
StreamWriter writer = new StreamWriter("C:\\Users\\user\\Desktop\\streamWriter.txt");
```

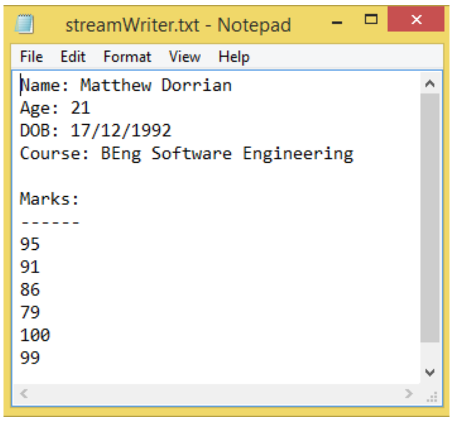
**Step 3:** We want to write to a text file using a `StreamWriter`. To do this, open the Program.cs and we will use the `WriteLine` method of the StreamWriter to write data to the file:

```
//Write data to the StreamWriter
writer.WriteLine("Name: Matthew Dorrian");
writer.WriteLine("Age: 21");
writer.WriteLine("DOB: 17/12/1992");
writer.WriteLine("Course: BEng Software Engineering");
writer.WriteLine();
writer.WriteLine("Marks: ");
writer.WriteLine("------");

int[] marks = {95, 91, 86, 79, 100, 99};

//Write each mark to the file
foreach (var mark in marks)
{
    writer.WriteLine(mark);
}
```

*Remember to Close the StreamWriter as this releases the file to be accessed by other processes/programs. Again due to the chance of exceptions, use a try/catch block with the StreamWriter to ensure the program doesn't crash if an exception occurs.

Example output shown below:





```
Name: Matthew Dorrian
Age: 21
DOB: 17/12/1992
Course: BEng Software Engineering

Marks:
------
95
91
86
79
100
99
```

## Task 9: Using StreamReaders to read from a text file

**Step 1:** Open the Visual Studio File menu, and then select New (or press `Ctrl+Shift+n`). Then click on Project, select C# Windows Forms Application and name it `e.g. Prac08Task8`.

**Step 2:** We want to write to a text file using a `StreamReader`. To do this, open the Program.cs and we will create a `StreamReader` object which will write to a text file to the specified location. StreamReader performs all the FileReader does in one statement as shown below:

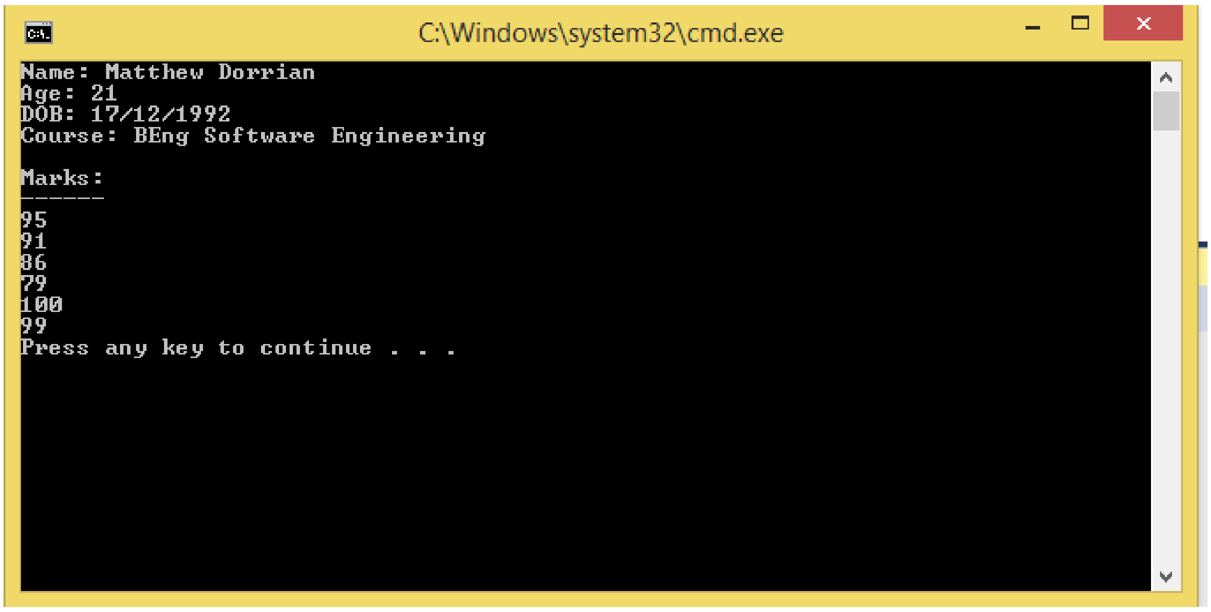
```
//Create a StreamReader specifying the location of the file to open
StreamReader reader = new StreamReader("C:\\Users\\user\\Desktop\\streamWriter.txt");
```

**Step 3:** We want to print each line from the file to the console so we will use the `ReadLine()` method of the StreamReader to read each line:

```
//Read each line in the file
string line;
while ((line = reader.ReadLine()) != null)
{
    Console.WriteLine(line); // Write to console.
}
```

*Remember to Close the StreamReader as this releases the file to be accessed by other processes/programs. Again due to the chance of exceptions, use a try/catch block with the StreamReader to ensure the program doesn't crash if an exception occurs.

Example output shown below:



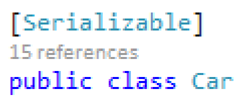
```
Name: Matthew Dorrian
Age: 21
DOB: 17/12/1992
Course: BEng Software Engineering

Marks:
------
95
91
86
79
100
99
Press any key to continue . . .
```

## EXERCISE 3

### Task 10: Using Serialisation to write and read objects to and from a file

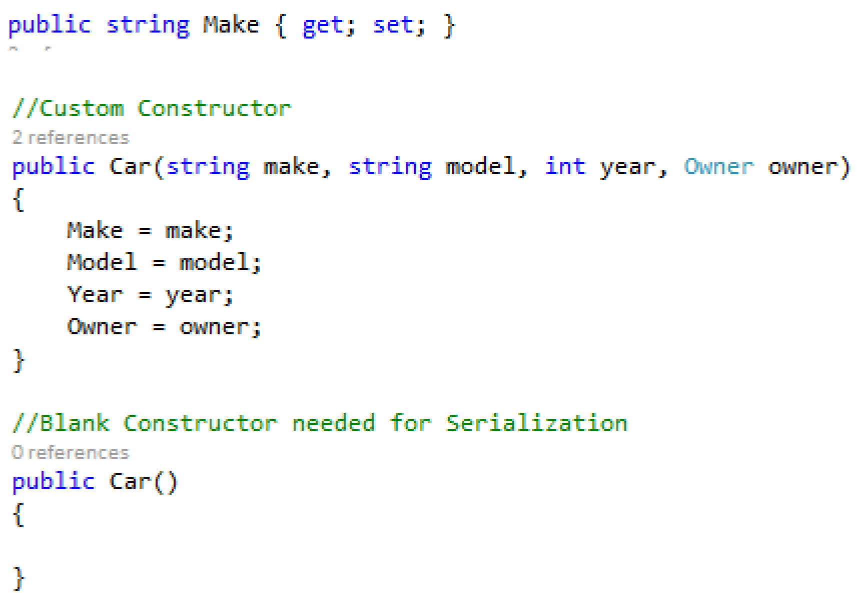
**Step 1:** Open the Visual Studio File menu, and then select New (or press `Ctrl+Shift+n`). Then click on Project, select C# Windows Forms Application and name it `e.g. Prac08Task9`.

**Step 2:** We will create a Car class which will hold information about a car e.g. engineSize, make, model and create an Owner class which will store the owner's name and age. We have to tag the classes as Serializable which means we can write objects to a file (Refer to Serialization in the PowerPoint):



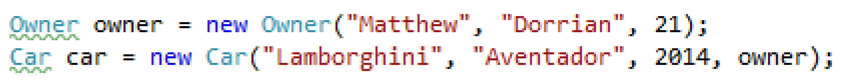
```
[Serializable]
15 references
public class Car
```

For the Serialization to work, there has to be a blank constructor but you can add your own constructor to instantiate the object and have the variables set to public:



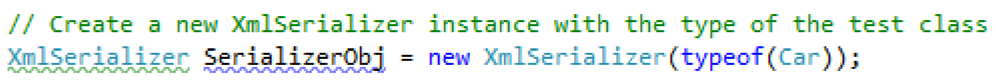
```
public string Make { get; set; }

//Custom Constructor
2 references
public Car(string make, string model, int year, Owner owner)
{
    Make = make;
    Model = model;
    Year = year;
    Owner = owner;
}

//Blank Constructor needed for Serialization
0 references
public Car()
{

}
```

**Step 3:** Create instants of the Car class which will consist of the car's details and the Owner object:



```
Owner owner = new Owner("Matthew", "Dorrian", 21);
Car car = new Car("Lamborghini", "Aventador", 2014, owner);
```

**Step 4:** To serialize the object to an XML file, we need to create an `XmlSerializer` object and set the type of object the Serializer will expect e.g. Car. Open Program.cs and create the XmlSerializer:



```
// Create a new XmlSerializer instance with the type of the test class
XmlSerializer SerializerObj = new XmlSerializer(typeof(Car));
```

**Step 5:** Create a `StreamWriter` and specify where to write the xml file to. Use the `Serialize` method of the `XmlSerializer` to write your object to the file location using the StreamWriter:

```
// Create a new file stream to write the serialized object to a file
StreamWriter WriteFileStream = new StreamWriter("C:\\Users\\user\\Desktop\\serializedObject.xml");

//Serialize method will serialize the object and write to the file
serializerObj.Serialize(WriteFileStream, car);

// Cleanup
WriteFileStream.Close();
```

*Remember to Close the XmlStreamWriter as this releases the file to be accessed by other processes/programs. Again due to the chance of exceptions, use a try/catch block with the XmlStreamWriter to ensure the program doesn't crash if an exception occurs.

Example output shown below:

```
  <?xml version="1.0" encoding="UTF-8"?>
- <Car xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <Make>Lamborghini</Make>
     <Model>Aventador</Model>
     <Year>2014</Year>
   - <Owner>
        <FirstName>Matthew</FirstName>
        <LastName>Dorrian</LastName>
        <Age>21</Age>
     </Owner>
  </Car>
```

**Step 6:** To deserialize the object from an XML file to assign to a variable, we need to create an `XmlSerializer` object and set the type of object the Serializer will expect e.g. Car. Open Program.cs and create the XmlSerializer:

```
// Create a new XmlSerializer instance with the type of the test class
XmlSerializer SerializerObj = new XmlSerializer(typeof(Car));
```

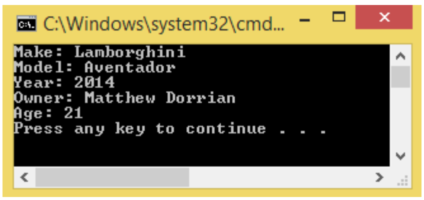
**Step 7:** We need to read the XML file so to do this, we need to create a FileStream to read the specified xml file:

```
// Create a new file stream for reading the XML file
FileStream ReadFileStream = new FileStream("C:\\Users\\user\\Desktop\\serializedObject.xml",
    FileMode.Open, FileAccess.Read, FileShare.Read);
```

**Step 8:** We will use the `Deserialize` method to read the xml from the file and cast the object created from the stream to a Car and assign to a Car object. Next we will print out the details of the returned Car object:

```
// Load the object saved above by using the Deserialize function
Car loadedCar = (Car)serializerObj.Deserialize(ReadFileStream);

//Call the PrintDetails of the Car to see the details of the car
loadedCar.PrintDetails();
```

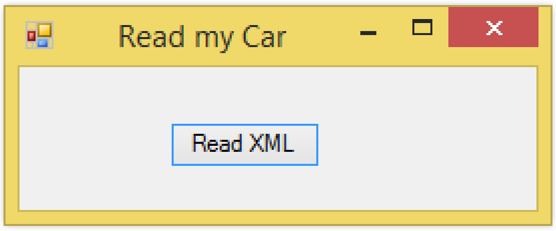
Example output shown below:

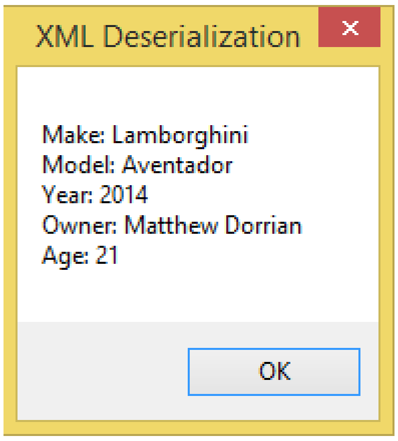


```
Make: Lamborghini
Model: Aventador
Year: 2014
Owner: Matthew Dorrian
Age: 21
Press any key to continue . . .
```

## Task 11: Serialization

To get used to serialization, create a GUI project which will show the details of a deserialized Car object in a MessageBox. Example output shown below:





```
Make: Lamborghini
Model: Aventador
Year: 2014
Owner: Matthew Dorrian
Age: 21
```