WORKSHOP 1 – INTRO TO C# & VISUAL STUDIO

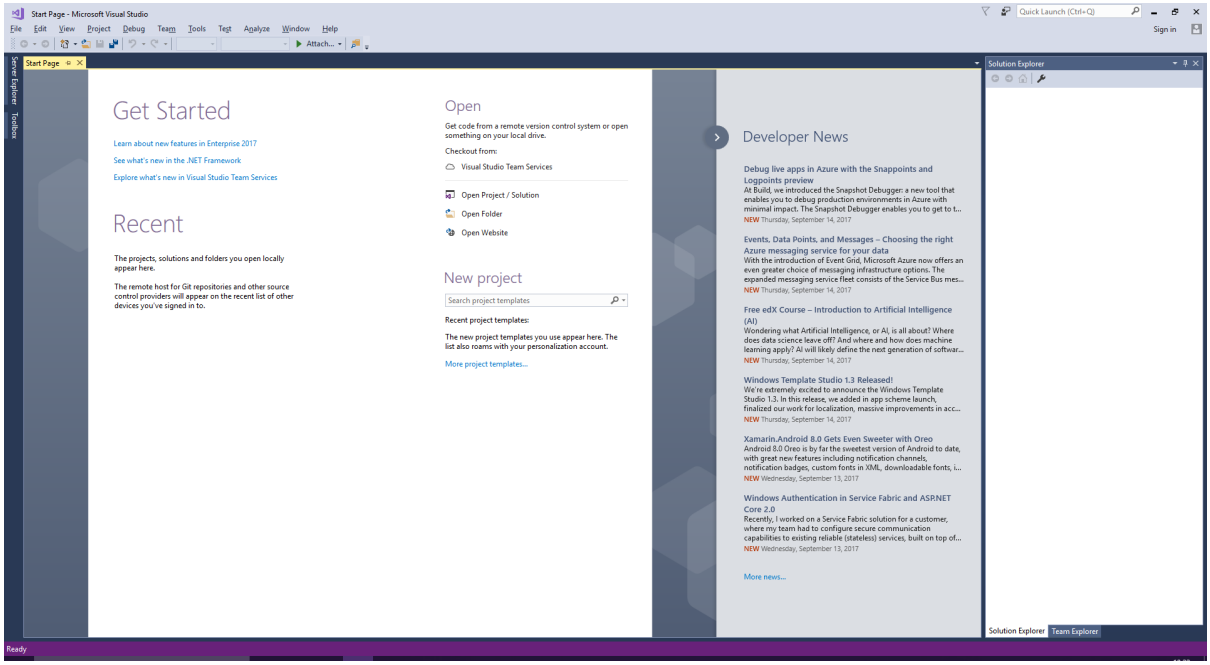
PURPOSE OF THE WORKSHOP

The purpose of this week is to get you ore familiar with C# and the Visual Studio IDE. This week should not be too difficult and will allow you to take the knowledge you already know from Java and use that to solve the problems here. Because this is a new language to you, there will be constructs that you do not know how to use. I will provide material to help you, but be prepared to do your own research to find out how to do specific actions.

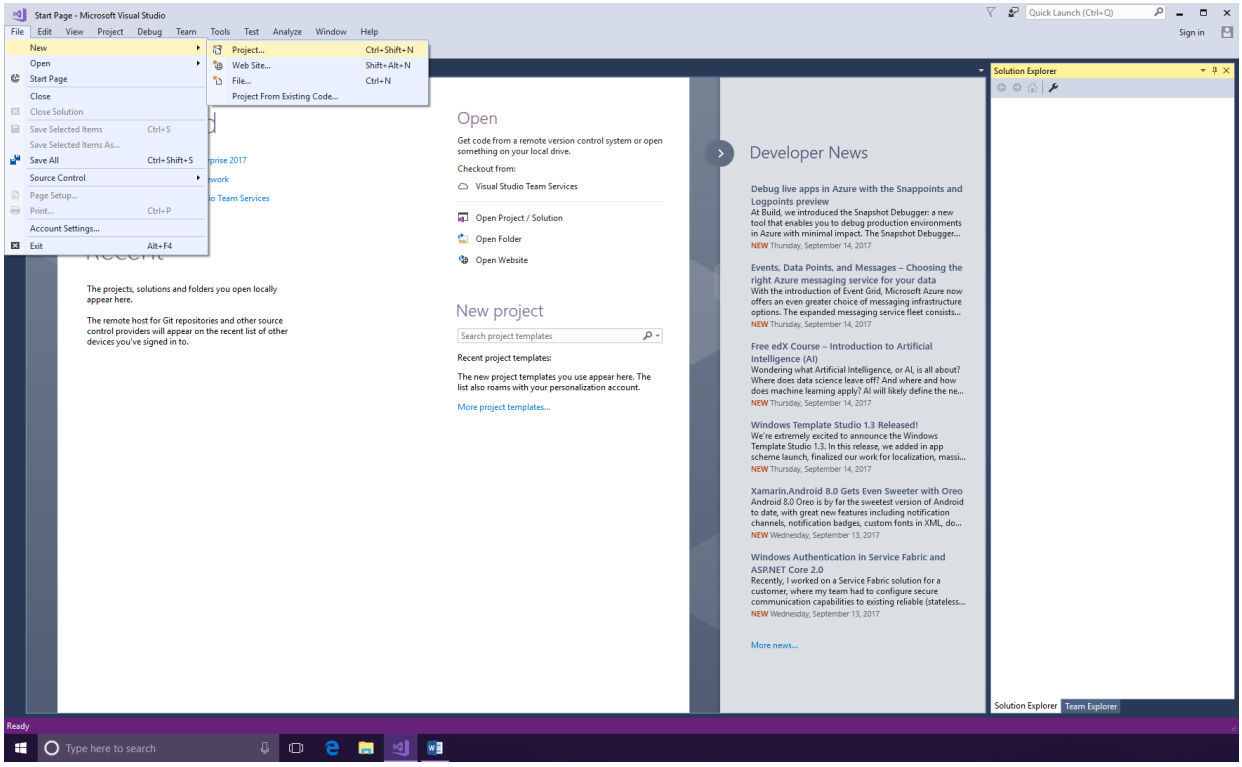
When you start Visual Studio 2017 for the first time you will be faced with some configuration information. It is up to you whether you choose to sign in or not, also I will leave the theme choice up to you.

PART 1: CREATING A C# CONSOLE PROJECT

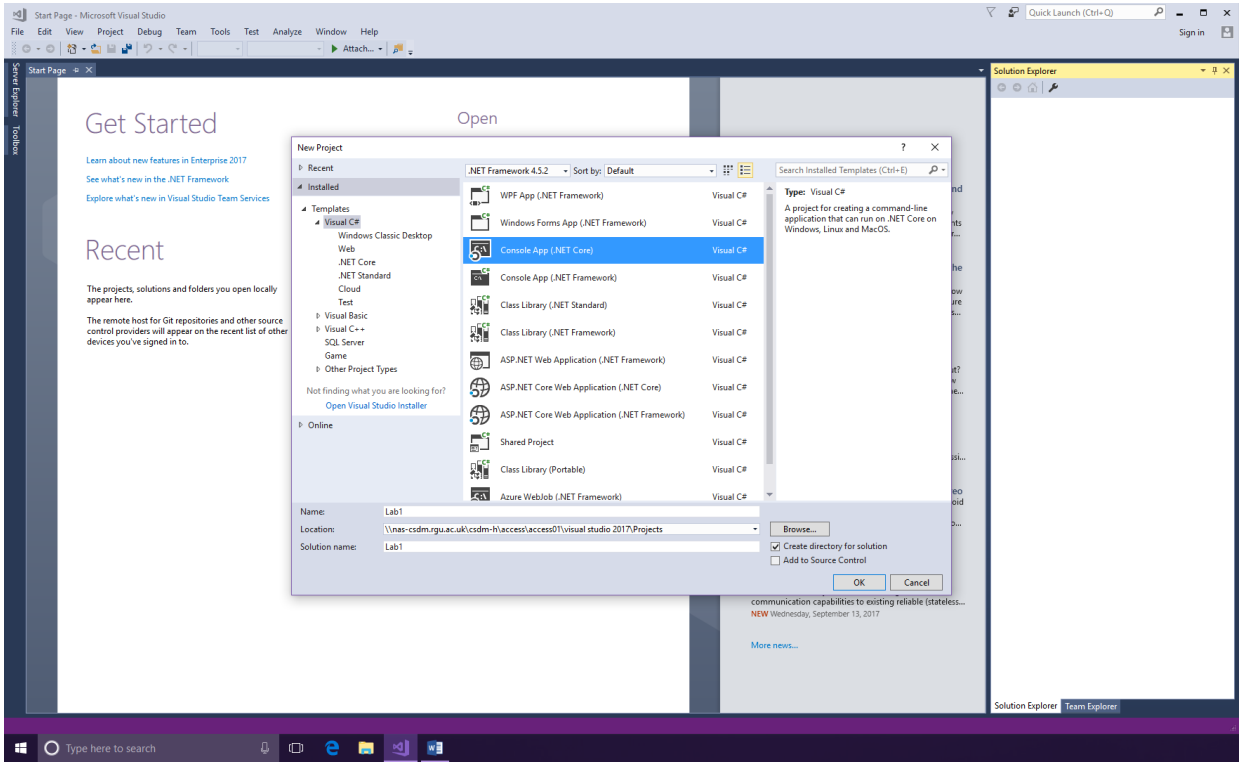
Once you get past the ~~spammy~~ introduction parts to visual studio you should be faced with a page that looks rather like the one below.



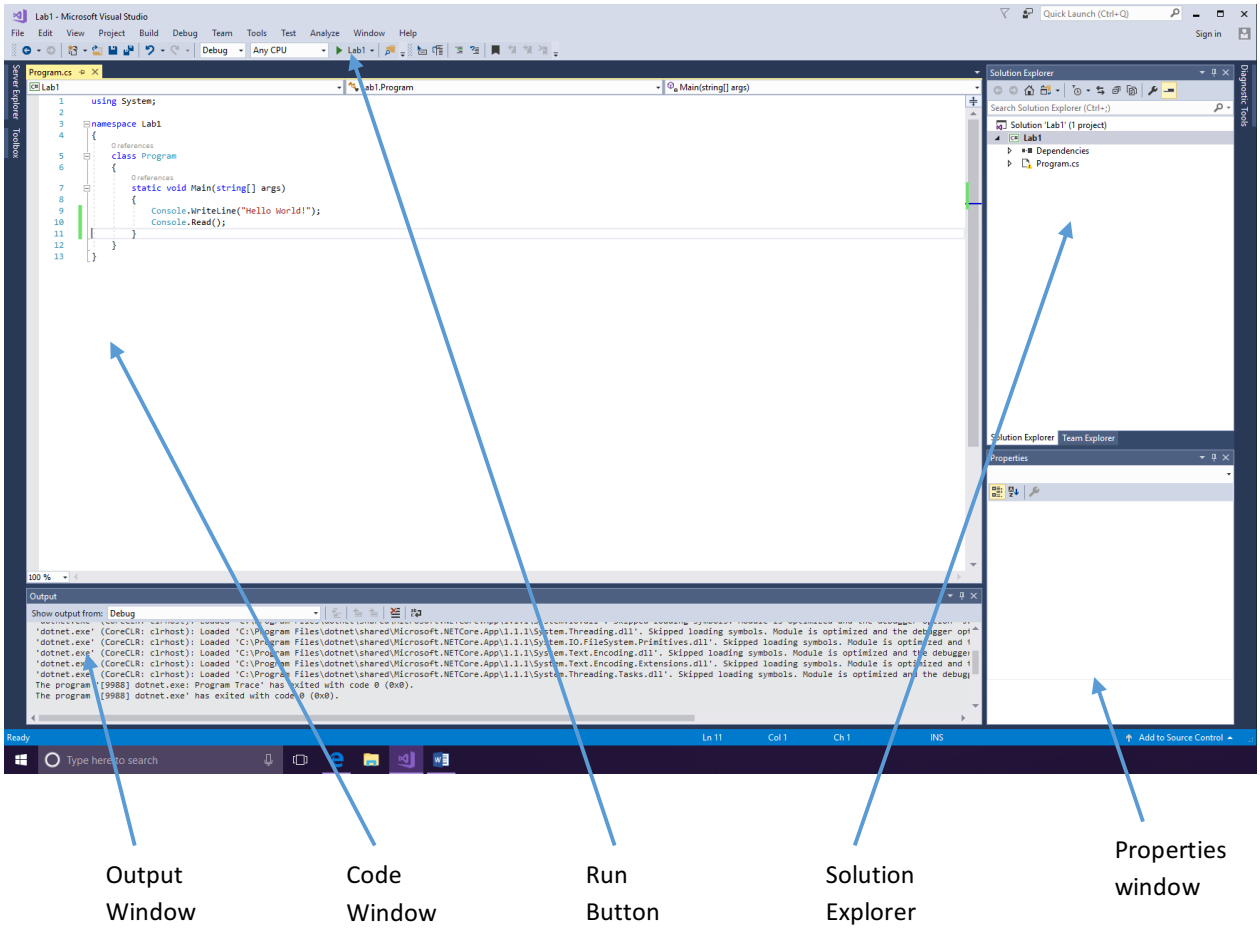
From the **file menu** select **new** then **project** this will give you the new project window.



From here you need to navigate to **Templates -> Visual C#** then from the list that appears select **Console App (.net Core)** note: .NET core is the newer, cross platform variant of the .NET framework. Give your project a sensible name, and store it on your H drive. You can set up source control (GIT) here as well if you like.



Finally, you will meet the main project screen. You should see that the template has already created the main method for you.



Output window – where debug and error info will appear.

Code window – where you craft your works of excellence.

Run Button – compile and execute your code.

Solution Explorer – your solution (project) structure.

Properties window – allows you to set run-time properties for some files and components.

When you hit the run button the code will compile and run. You will see in the screenshot above that I have added an extra line **Console.Read();** this locks the console (command line) window open, by waiting for you to hit a key, so you can see any output from your application. Try this now and make sure that the compiler is working properly.

If you want to set command line arguments to test you can do it through the following steps

1. With a project selected in **Solution Explorer**, on the **Project** menu, click **Properties**.
2. Click the **Debug** tab.
3. In the **Command line arguments** field, enter the command-line arguments you wish to use.

PART 2: READING A FILE USING SIMPLE FILE IO

When we move on to building our compiler, one of the first things you are going to do is read in a file containing the source code for processing. For your first C# project you should create a simple file reader that takes a file name as a command line argument.

When you created your project you will have seen that the template created the initial program.cs file for you For this exercise you should create a new file, within your project called, **MyReader.cs**. From here you should read any command line arguments and pass these to an instance of your other new class, **MyReader.cs**.

Your **MyReader.cs** class will use the command line argument, ie a filename, to find a file and process it. You are free to use any file reading technique you wish but **each line of the source file should be processed separately**

Your application should have at least the following methods,

ReadFile – the read file method will take a file instance and read the file given line by line

ProcessLine – the process method will process the file line by line. It should prepend a line number to each line, then write the line to the console.

Hint: In the cheat sheet on moodle have a look at the fileIO section. This will tell you how to read in a file using the StreamReader or BinaryReader

PART 3: EXTENDING AND INHERITING FROM MYREADER.CS

So writing to the console is not that spectacular. When we actually want to generate code for our target machines we need to create some kind of output.

Create a new file called **MyOutputReader.cs**, this file should contain a class with **inherits** from your **MyReader.cs**.

Remember to check back at the lecture to see how to do this in C#.

You should override the process method so that it instead of writing to the console a new output file is created.

PART 4: CREATING FILE STATS

Have a read of the C# documentation on interfaces https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/interfaces/

Now create an interface called StatsGenerator which should define the following methods

getWordCount – gets the total number of words

getCharacterCount –gets the total number of chracters

getLineCount – gets the number of lines

getFirstLetter – gets the first letter of every word

getFirstWord – gets the first word of every line

getEndLine– gets the last character of every line

Remember, just like Java, an interface does not contain any implementation, it is simply a way of enforcing what methods any class's that implements the interface has.

Now following the example in the lecture, implement the interface in your MyOutputReader.cs adding the required methods and developing the required functionality.

For the processes to work fully you may need to create an internal storage system for the input file before you write it out. You can use an array or arraylist for this as you see fit.

Details on C# arraylist here https://msdn.microsoft.com/en-us/library/system.collections.arraylist(v=vs.110).aspx

You should update your process method to write the file stats to a separate file called stats.txt

PART 5: SIMPLE CHECKING

Once of the major parts of a compiler is checking the code being compiled is valid. Can you write a method that checks each line of the input file ends with a ; or a +. If it does not, then no output file should be generated and error should be printed to the console defining what line the error occurred on.

Can you include code to ignore any text in a line that starts with a #