CO1404 : Introduction to Programming

Week1 - Introduction to Visual Studio and Simple programs

Lab Summary

This lab worksheet helps you write your first programs. Students who have experience programming should find it straightforward at first. However, there will be advanced questions later on. There's a lot of reading in this worksheet as we introduce the tools required. Later worksheets will tend to be shorter, with more work for you to do!

Remember: If you have any questions or require assistance, please get your tutors attention. We are all here to help!

Objectives

This Lab aims to achive the following objective(s) below:

PO.1: Introduce the development environment and programming language you will be using for this module.

Reading Material

Below is some light reading material to accompany the lab worksheet.

Microsoft Visual Studio

Throughout this module will will be using ***Microsoft Visual Studio***. As we have discussed in the lecture it is and IDE (Integrated Development Environment). This software has been made available to you in many different locations to accommodate your learning preferences. The software is readily available at the following locations:

- **UCLAN Lab PC's** - via the Uclan Network.
- **Azure Labs Virtual Machine** - Visit https://labs.azure.com/ (Note: you will have to have registered, please check your UCLAN email for a registration link)

Want to install Visual Studio on your own machine?

Alternativley, If you have your own machine (desktop or laptop), and would prefer to install it on that, you can. I have provided the links below install **Microsoft Visual Studio - Community Edition** on `Microsoft Windows 10` and `Apple Mac OS` .The partiucalr edition comes you will download comes with the plugins you require pre-configured so you do not have to do any additional configuration (*YAY!*).

- **Microsoft Windows Users** : https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community&rel=16
- **Apple Mac Users** **:** https://visualstudio.microsoft.com/thank-you-downloading-visual-studio-mac/?sku=communitymac&rel=16

Please note :
This module has been designed using Visual Studio installed on the ***Windows operating system***. Therefore, if you are using a Mac the user interface will look slightly different. But the core features and steps defined are very similar. If you have any questions please ask.

About : C# and .NET Framework

C# (pronounced *C-Sharp*, like the musical note) is a general purpose, multi paradigm programming langauge developed by Microsoft that runs on the .NET Framework. C# is used to develop web apps, desktop apps, mobile apps, games and much more. For those who are curious, more information can be found https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework

---

Lab Activity

Before you continue : make sure you have access to Visual Studio. As you are about to begin on your UCLAN programming adventure.

1. Creating your first "New Project"

When you first start Visual Studio, you may be asked what settings you wish to use. Select **Visual C# Development Settings**. This should only happen the first time you run it.

You begin on the **Start Page**, which links to various online resources and allows you to open or create projects. We want to ***create a new project***. Almost all your projects in this module are created this way so get familiar with these steps. As a mistake here can result in getting unusual errors later on, so be careful.

To create a new project we:

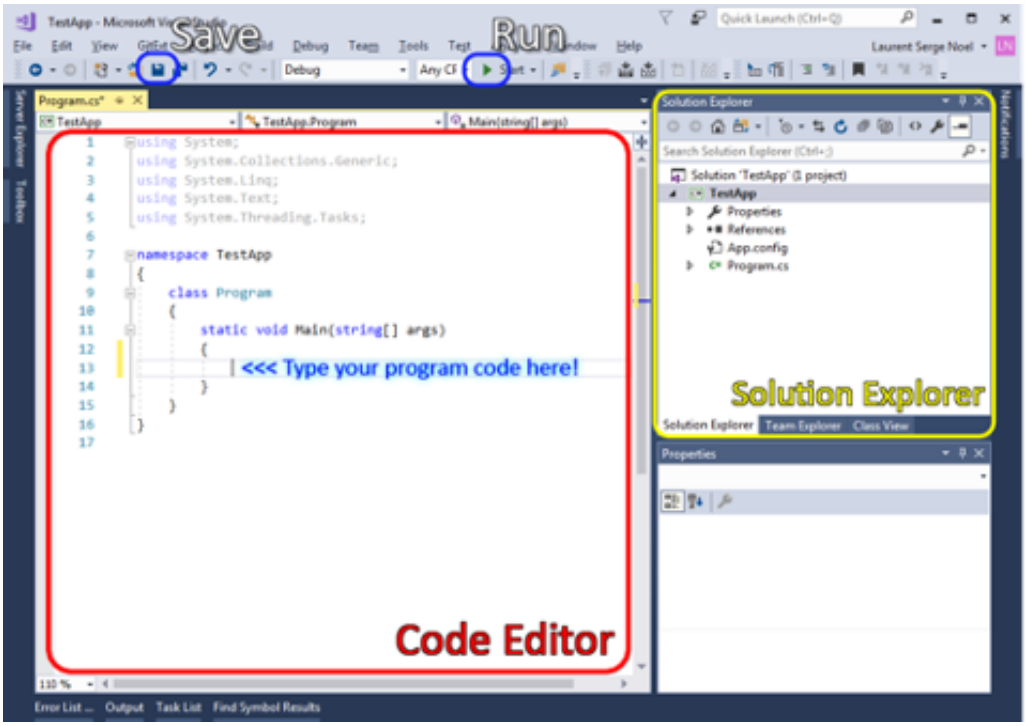
1. On the **Start Page** choose `Create new project…` . Alternatively, in the main menu choose `File > New >  Project…`.

2. From the left choose `Installed > Visual C#`, then from the list on the right choose `Console App (.NET framework)`.
3. Towards the bottom of the window, give a name to your project. Call this project **TestApp**. Also browse to a suitable location – I suggest you make a new folder **CO1404 Introduction to programming** amongst your documents just for this module. It is inside of that folder I would recommend you save all your projects associated with this module.
4. Finally, make sure the option `Create directory for solution` is **unchecked**. The remaining settings should be fine, so now press OK.

If you have followed the steps above correctly, after a moment or so some program code will appear. This is *shell code*: some lines that all your programs will need - so they are created automatically. We will look at them in a moment.

2. Let's explore Visual Studio IDE

Once you have created your project Visual Studio should look something similar to the image below (don't worry if it doesn't look exactly like this as view varies depending on the version or setup). For this module the main aspects of the user interface have been highlighted. They are **Code Editor, Solution Explorer**, and **Toolbar Commands**.



Code Editor

This is the main area. The code editor is a fairly standard text editor for writing/editing your code. You will notice that some of the words are coloured. This feature is known as **Syntax Highlighting**. It identifies different types of programming keywords. You will find this becomes more useful as your program abiliites improve and you gain more experience.

Solution Explorer

The solution explorer contains all the files and references used by your project.

- In your early programs you will have only one file of interest, called `Program.cs` by default. This is the file open in the text editor. You can rename it if you wish, and in later programs you might want to create multiple files to help structure your programs.
- The files are collected into a `project`, which you called `TestApp`. This is in turn collected into a `solution` also called TestApp. This higher level structure is only important for larger programs.

Toolbar Commands

The icons highlighted should already look familair to you:

- **Save Icon**: saves any changes you have made to your project.
- **Run Icon** : Will run your program.

Note: You will to use other features from the toolbar in the future.

Miscellaneous Information

Listed in the bullet points below are some miscellaneous bits of information that you may or may not need depedning on how Visual Studio presents it self to you.

- On some installations you will also see the **Properties pane** (but you may not, it depends on your setup). For this module, this pane is not needed until you cover Visual Programming later in your course. So you can close this pane if you wish. Get it back from the View menu.
- You might notice text near the braces that says **0 references**. This is an advanced feature that we don't want. Right-click on the that text, choose **Codelens Options...**", then uncheck **Enable Codelens**.
- You can change the **theme** of visual studio for those who prefer **Dark Mode**
    - Go to **Tools > Options**
    - On the left you will see **Environment** press that and click **General**
    - You will see a group of options called **Visual Experience**. You will find a drop down menu with various options, choose your option and press **OK**

3. Let's understand the Shell Code

```
using System;

namespace Test
{
    class MainClass
    {
        public static void Main(string[] args)
        {
            //Your code goes here !
        }
    }
}
```

Above is an example of what the **shell code** looks like. We will rarely change the shell code in this module, but here's the same quick overview from the lecture:

- The `using` statements indicate what parts of the pre-written C# library code our program will use. If we want special features (e.g. file features, network features) then we add more using lines here
- The `namespace` line, which has the name of your project, ensures your project is kept separate from other projects and system code
- Notice the **curly brackets** or **braces** – the symbols `{` and `}`. They keep the code into *blocks*, which we will see the importance of later. The braces are joined with a dotted line so you can see where each one starts and ends.
    - In Visual Studio, you can press the little `+` and `–` signs at the left to expand or compact these blocks to help readability in a larger program.
- The `class Program` and the line marked `Main` are the part that will contain your code in your early programming.
- Finally, your code goes inside the innermost braces `{ }`. So make a new blank line in-between them and let's start programming.

4. Writing your first program - "Hello World"

At this point we have familiarised ourselves with the IDE and we understand the main components of the shell code. When typing code it is good practice to use the tab key to indent the code one time more than the braces surrounding it. This will help make your code more readable. If you need, please ask your tutor to show you what this means.

Now it is time to create our very first program, follow the instructions below:

1. Between the innermost braces, type the following statement:

    ```
    Console.WriteLine("Hello World");
    ```

Note:

- **C#** is ***case sensitive***. That means capital letters are considered different to lower case letters. When copying program code copy the capitals exactly. When writing your own program code you must use capitals consistently (e.g. "Console" is different to "console")
- Also notice the semi-colon `;` at the end of the line. Semi-colons are required to separate *statements*, although they are not needed in all cases, e.g. they're not needed after braces `{` `}`. You will probably forget these a few times at first, don't worry - you'll get used to them soon.

2. Compile and run your program by pressing the **Play button** in the toolbar.

3. Wait a minute what happend? Nothing or did something flicker? Why? Well the program you have written simply just writes a line of text to the console. Once your program has finished displaying the text is exits. To see our hardwork, we need to add a line to pause while we look at the program result. Add the following statement underneath the statement you added previously.

   ```
   Console.Readline();
   ```

   This will cause your program to pause whilst it waits for the user to enter something, you will need something similar in many of your programs.

4. Now run it again. This time it should display the text and then wait for you to input something into the console window.

This program works by using pre-written **methods** (ReadLine and WriteLine) from the C# **Console library**.

- The **console** is a window (usually with white text on a black background) that allows a very direct interface between program and the computer.
- Although console programming is rather limited it is a great way to get familiar with core programming concepts without other features getting in the way. So we will only create console programs in this module, in later modules you will create a wider range of programs.
- We will often methods from the pre-written libraries provided with the C# language to simplify the code we have to write.

5. Different Ways to Run your program

There are several different ways of running your program inside of Visual Studio. But this is only useful to programmers. What if we wanted to share this with other users?

1. In your operating systems file browser, (Explorer in Windows or Finder in MacOS), navigate to the folder where you saved the project, then look in the folder `bin` and then `debug`. You should find a file of type **Application**" with the name of your project (hover over the file to find if it's an application or not). It is this file you can share with others.

    - This particular version of the application is know as a **Debug build**. The program has been compiled to help us debug it in case there are problems. That means it will be a little larger and potentially slower than normal. This makes very little difference to our simple programs and we will often need to debug. But you can create a *release build* using the drop-down list near the top of Visual Studio. However, use debug builds most of the time.

2. Sometimes we do not need to program to completely finish all its operations. We can stop the program at any time. If you run the application by clicking the **Play** button in Visual Studio you should be able to see a **Stop** button, press it and the applicaiton will terminate the program early. This is especially useful if things get stuck!

Tips:

- In **Microsoft Windows** you can double click the application to run it like any other program.
- To change the icon for the application: select the project in the Solution Explorer, right click, Properties, change icon at bottom of "Application" tab (use the .ico file provided with this week's material on the module webpage). Changing the icons only affects the executable when you run it from Windows, not when running from within Visual Studio. Try it now, it will change what you see in the task bar and in the top left of the console window.

6. Errors and Documentation

One advantage of using a IDE like Visual Studio is the error detection features it provides us. Lets explore a few of these features now.

Error Detection

1. In the code editior type a `%` somewhere in the space after `Console.ReadLine();`. Note the red squiggle that appears after a moment, like a spelling mistake. This is a ***syntax error***: you're not allowed to type a `%` there in C#.

    If you hover over the squiggle to show an error message. Note that this error confused Visual Studio into producing multiple error messages, of course the other errors are caused by the first error. This is not uncommon.

2. Try to run the program with the error still in it, you are asked whether you want to run the previous version. This is rarely useful, so check "Do not show this dialog again", the press **No**. Also notice an **Error List** pane appears.

3. Remove the `%` and try deleting one of the semi-colons `;` to prove that they are necessary.

   Hovering over the missing semi-colon error can be difficult, so find its error in the **Error List** (if the Error List is not there, go to the menus and choose **View > Error List**). Double click any error in the Error List and it will take you directly to the error in Visual Studio. Fix any errors before moving on.

Access Documentation

1. Hover over `ReadLine`, `WriteLine` or `Console` for quick documentation.

   In windows you can click on ReadLine and press F1 for full documentation. (note: recently you might need to create a Microsoft account to do this). Note that the documentation is written for experienced programmers and is too complex for novices so don't worry about understanding it right now. However, it can sometimes be a good source for example code. You can also select errors in the Error List and press F1 for more info

7. Now let's create another Program

Save your project (find Save All in the tool bars). Then create a new project from scratch and call it **NameEntry**.

1. Using your knowledge from the previous program, make this new program display your name on the screen. Remember to add a `Console.ReadLine();` statement so the program waits for you to press return before returning to Visual Studio.

2. Next we write some code that will ask the user to type their name then display it. We use the `ReadLine()` method again – this method waits for the user to type something. Before we ignored what the user typed. In the first program, we ignored what the user typed. This time we will store what they type in a `string` variable.

Two new concepts here, which we will see in more detail later:

- Think of a ***variable*** as a box with a name on it that holds some information we need in the program, in this case the name typed by the user.
- A `string` is a sequence of characters (`char`), which can be letters, numbers or punctuation. Just what we need to hold what the user might type.

To do this ,we have to **declare** a variable. We do this by stating what type of variable is it and provide give it a name. The code below is how we declare variable with a type of `string` called `username`:

```
    string username;
```

Put this declaration at the start of your program (i.e. inside the braces `{` `}` of `Main`, but before the Write/ReadLines statements that you have written).

3. Now on a new line add the following statement after the delcaration you made in the previous step.

   ```
   username = Console.Readline();
   ```

   This will now assign the variable `username` with the text inputted by the user as a result of the `Console.Readline()` method.

4. Now update the `Console.WriteLine("Hello World");` statement to output the `username` variable instead. When you get it right you should be able to type your name (or anything) and the program will repeat it back at you.

   Tip : you don't put quotes around variable names.

8. Tweaks and Changes

Now let's make some improvements to the user interface.

1. Use some more `WriteLine` statements to make the program display `"Enter your Name"` before it requests the user to input text into the console. Also display the text `"Hello"` before displaying the user's name.

2. The `WriteLine` method always moves to a new line. There is also another method `Write` that doees not. Using this information update your program using `Write` and `WriteLine` methods so that the console output looks like the example below:

   ```
   Enter your name : [User inputs their name]
   Hello, [user's name appears here]
   ```

   Note: the squared brackets represent input from the user.

3. We can write out text and variables together. Examine the following statement:

   ```
   Console.WriteLine("Your name is: {0}", username);
   ```

This statement will replace the `{0}` with the values of the `username` variable. This saves us having to write two statements to achieve this. You can also use `{1}`, `{2}` etc. and add even more variables. However, there is no similar thing for the `ReadLine` method.

Now update your program to have the user to input their **first** and **second** name separately. You will need two `string` variables. Use the statement above to write the names out in reverse order. It should look like the example below.

```
You are Campbell, Daniel
```

4. Finally, output the names in upper case using `<variableName>.ToUpper()` Replacing `<variableName>` with those you have chosen to represent the user's first and second name. There are many supporting **methods** like this in C#. Too many to go through in this module, but a few convenient ones will be introduced as we progress

5. Save your work!

9. Working with Numbers

Create another new project, this time call is **SimpleOperators**. Now in the last project we used `string` variables to hold text. Here we are going to become famialir with our first numerical variable the **Integer**.

1. In C# we use the keyword `int` to declare an integer variable. In your new project's `Main` method declare an integer variable called `userHeight` , as shown below:

```
int userHeight;
```

2. In this program we're going to ask the user for their own height (just a number, let's assume it's in cm) and the heights of their two neighbours. To input a number from the user in C# we have them input a string and then we convert that string to a number. Add the following code to your program to do that:

```
string heightString;
heightString = Console.ReadLine();
userHeight = int.Parse(heightString);
```

So why can't we just use the number how it is typed in by the user? Well, most programming languages don't see strings as numbers, even if that's what they contain. A `string` is a sequence of **characters**. A character is a letter, digit or punctuation mark. Interpreting the meaning of a sequence of characters is called **parsing**. So we use a C#

library `Parse` method to interpret our string as a number – this method will read the digits one at a time and form the number. It will also spot problems, for example if we type letters instead of numbers.

3. Now complete the input part of the program: you already have the code to input your own height. Add some code to write some text to the screen first so the user knows what they are doing, e.g. `"Please enter your height in cm: "`.
4. Add further code, to ask for and input heights for your two neighbours. This code will clearly be very similar to the code you already have. But you will need more variables for these new heights.

Now time for some fun. Let's perform some calculations. Imagine a different program with two integers `a` and `b`. I want to put their **sum** (`+`) in another integer `c`. This is how to do it:

```
int c;
c = a + b;
```

We can do other basic arithmetic, just note for division you use `/` and for multiplication you use `*`. You can combine operations and use brackets `(` `)` too:

```
c = a + (b * b)
```

Now let's get back to reality. In your program you should have three heights. Now we want you to calculate the MEAN (aka - average) of the three values.

1. Delcare a new integer called `averageHeight`.

2. Now use arithmetic to calcualte the MEAN of your three height variables and assign the result to the `averageHeight` variable you declared in the previous step.

   Note: To calculate the MEAN you the work out the sum total of the three heights and divide by the total number of variables (`3`).

3. Finally display the average height with a suitable message.
4. Update your program to output the average height in inches. To convert cm to inches, divide by 2.54.
   - Depending how you do this you may get an error. The problem is that the height in inches is no longer an integer, as it has a decimal point (e.g 72.4 inches). This is called a **floating point** number.

- You can make a variable for a floating point number using the `double` type instead of `int`. Use a variable like this to help solve any errors:

```
double heightInches;
```

5. Your floating point output will probably have too many decimal places and be unpleasant to read, e.g. you might get a result like `48.1234812542735 inches`. You can solve this by using `{0:F2}` instead of `{0}` in your `WriteLine` method. This will display a fixed 2 decimal places only, e.g. `48.12`. There are other formatting possibilities, which we will discover in later weeks.
6. Finally, update your program again to output the average height in feet and inches, e.g. 70 inches is 5 feet 10 inches. Format the output message nicely.
   - **Tip:** Must calculate a remainder for this: use the `%` sign like the `/` sign. For example;

```
70 / 12 = 5  // feet
70 % 12 = 10 // inches (as this is the remainder i.e (70 = 5x12 + 10))
```

     This needs some initiative, but ask your tutor if you need help.
   - **Note:** You can use integer or floating point values, it's up to you.

Disclaimer : *The following "Advanced Challenges" are intended as fun exercises designed to push your knowledge to the limits. I encourage you all to give them ago, ask questions if you get stuck but most importantly don't stress about them.*

---

10. Advanced Challenges

Fancy a challenge? Then you are at the right place.

Advanced Challenge 1:

In the final project **SimpleOperators** it is possible to reduce the number of variables needed in this program. A sensible solution will use only one string variable. However, can you go further and simplify the code to use no string variables at all?

Advanced Challenge 2:

Update your program so the user inputs the heights in feet and inches and displays the average in feet and inches. Ask the user separately for feet and inches, but then convert to inches only to do the averaging correctly, then convert back for output.