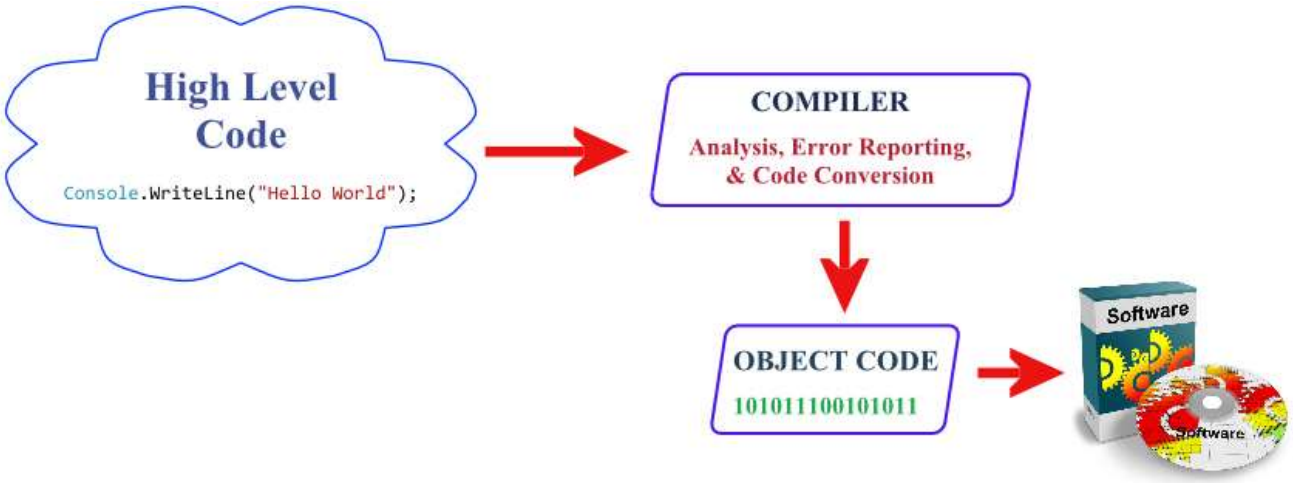
MODULE 1: INTRODUCTION TO PROGRAMMING IN C#

TUTORIAL 1: USING VISUAL STUDIO FOR C# PROGRAMS

Introduction

In this course, you will learn to write computer programs in a language called C# (spoken as 'C sharp'). A computer program can be broadly defined as a set of instructions that perform some given task. Programs provide instructions to a central processing unit (CPU) that carry out these instructions at a very rapid speed. We write our programs in a form that is easily understandable, using a high-level language such as C#. However, this program has to be converted into the 'simple' instructions that the processor can understand (object code) and this process is known as *compilation*. The process is illustrated below:



Most programmers use high level languages such as C#, C++, Java, Visual Basic, and many others – however for some applications e.g. control systems, developers may choose to write in a low-level language in which the instructions equate to the instruction set for a specific processor.

The C# language is a popular choice amongst software developers and you will develop your programming skills using this language throughout most of this course and become proficient in C#. With this knowledge, you will find it easy to transfer these skills to similar languages such as Java and C++.

These languages feature *Object Oriented Programming (OOP)*. For professional software development, OOP provides facilities in which robust software can be created – you will learn more about the benefits in Module 2 of this course and gain practical skills in writing OOP programs.

You need to understand the basic programming concepts that apply to most high-level programming languages and, for the moment, we shall concentrate on those aspects. We start off by looking at some simple examples and creating programs in C# using Visual Studio which provides an 'integrated development environment (IDE)' in which we can write, compile, and run our programs.

Study Guidelines

If you have not used Visual Studio before and you want to download it to your own computer, you should start off by looking at the following video:

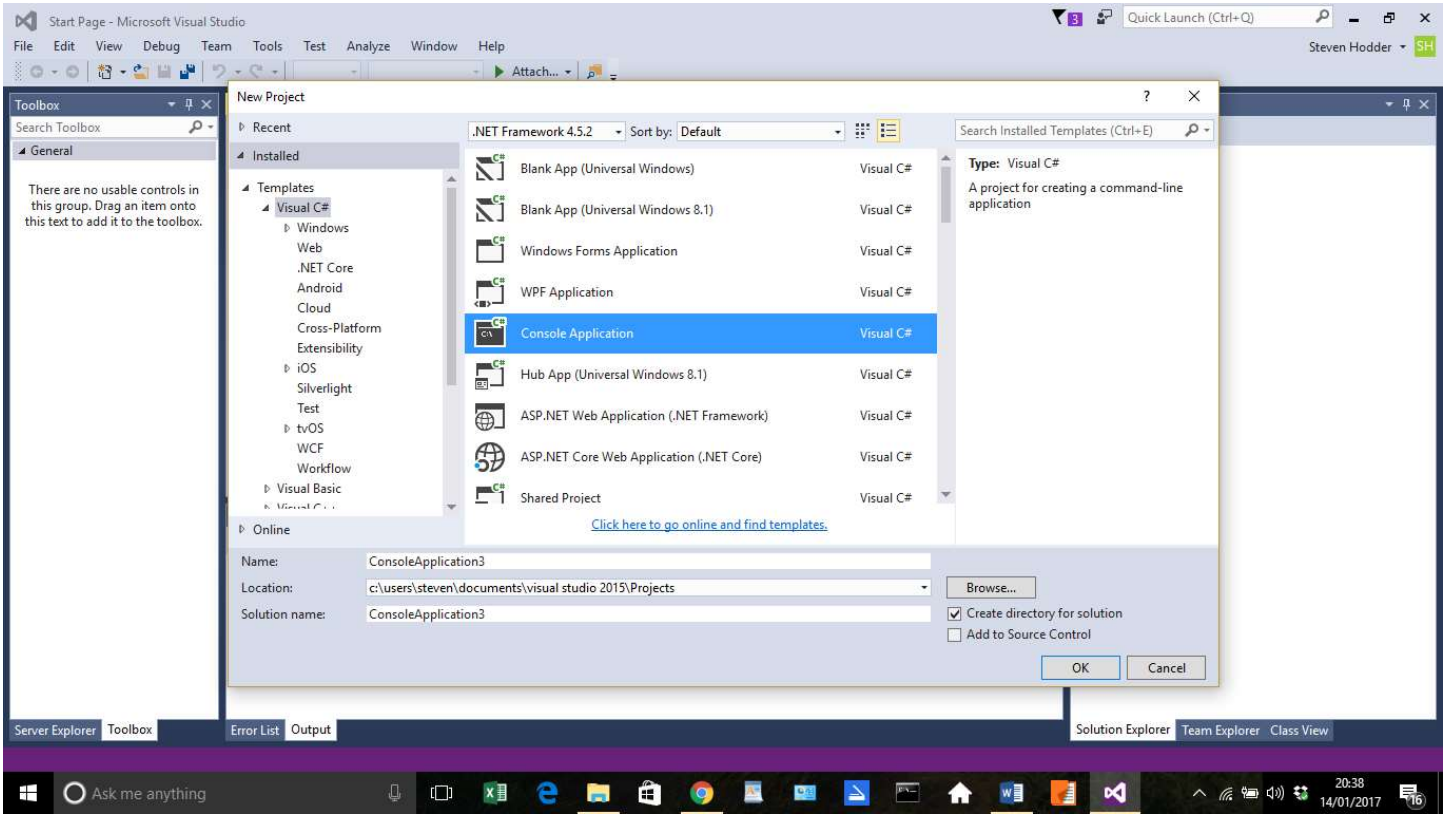
https://channel9.msdn.com/Blogs/dotnet/Get-started-with-VS-Code-using-CSharp-and-NET-Core

Visual Studio is pre-installed in the JBC computer studio and you can start by creating the traditional 'Hello World' program to ensure that you can write, compile and run a simple program.

1. Console Applications

Open Visual Studio and click on File | New | Project and look under the options for C#.

You will be presented with a list of project selections, as shown below:



Two of these will be important for this module: Console Application and Windows Form Application. For this exercise select Console Application, change the name (and location if required) and click OK.

This provides you with the basic structure for a C# console application. By this we mean that you will create a program involves only text dialogue between the user and the computer. Such programs are not as attractive as ones typical of Windows programs but they will teach you a lot more about programming!

Your screen should show:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

To create your first 'Hello World' program, change the latter part of the program to:

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("hello, world");
            Console.ReadLine();
        }
    }
}
```

To run the program, click on the green arrow (start) button in the toolbar. A new window will appear showing your program output. If we had not included the Console.Readline() command, the window would have closed as soon as it had printed the message. Instead, the program waits for an input from the user – this can be anything or nothing followed by a carriage return ('Enter'). Alternatively, you can run the program using Debug – Start Without Debugging from the main menu (or press Ctrl and F5 together).

Notes:

The template program provided when you select New -> Project …. contains more than you need for this simple exercise. You could trim the start to:

```
using System;
```

which would simply incorporate the necessary routines for simple input/output. Alternatively, you could omit this altogether and use the following line of program code:

System.ConsoleWrite(……);

Note the use of Write and WriteLn to output to the console text that may or may not require carriage returns (new lines). As mentioned, the Console.ReadLine command is used to stop the program display disappearing before the user can read it. We basically require the user to press a key while the screen displays a message

before the program finishes. However, we shall use this command to get information from the user. Try modifying your program as follows:

```
static void Main(string[] args)
  {
      String userName;
      Console.Write("What is your name? ");
      userName = Console.ReadLine();
      Console.Write("Hello ");
      Console.WriteLine(userName);
      Console.ReadLine();
  }
```

You will be required to develop many console applications in this course even though these may not reflect the typical programs that users of Windows systems would use. However, by writing these programs, you will learn, in detail, about program structure and content which will be important in your further studies. Later in this module. you will also develop GUI (graphical user interface) programs – also referred to as 'Windows Form Applications'. Visual Studio provides many easy-to-use facilities for you to develop impressive programs and these have their own knowledge requirements.

Additional Notes

You can now create and run very simple C# programs and you have seen the idea of being able to store a value (in this case a piece of text) identified by some name (in this case *userName*). In computer programming, we refer to such items as v*ariables* which have different types (in this case *String*) according to the information being held. In this tutorial, you will learn about the different data types and create simple programs to do various things with them.

If a build fails, you should respond with "No" to the *"There were build errors. Would you like to continue and run the last successful build"* dialogue. By double clicking on an entry in the "Error List" the relevant line with the fault is displayed.

At the end of this tutorial, you should be able to write simple console applications involving:

- Simple console input/output commands
- Different variable types: float, int, char, string
- Appreciate the use of layout and scoping

1. Console Input/Output Commands

In a console application, the user supplies information to the program by means of the Console.Readline() method e.g. userName = Console.ReadLine(); When the program runs and reaches this statement, it pauses and waits for the user to type something in. This will always be in the form of a string (text), even if you type in a number! We shall consider this a little later.

To output a piece of text, we use Console.Write("What is your name? "); or Console.WriteLine("What is your name? "); The difference is that Console.Write leaves the cursor at the end of the text, whereas Console.WriteLine moves the cursor to the start of the next line (a so-called carriage-return and line-feed).

Note that we used two program commands to output our result:

```
Console.Write("Hello ");
Console.WriteLine(userName);
```

An alternative way of achieving this would be:

```
Console.WriteLine("Hello {0} “,userName);
```

At runtime, the {0} is replaced by the text stored in the string username. If we wanted to output a number of variables, we would use {0} {1} {2} and so on within the inverted commas and list the variables afterwards e.g.

```
Console.WriteLine("x = {0}, y = {1}, z = {2}",x,y,z);
```

2. Different Variable Types

So far we have only met one type of variable (or data type) – String. The string can be initialized when it is declared e.g. String message = “Hello World!”;

We can hold a single character in a String type. Alternatively, we can use the char data type e.g. char c = “B”;

If we want to store numbers, we need to distinguish between integers (e.g. 23, -72, 0. 957) and floating-point numbers (e.g. 3.14, -2.5, 75.0). These are denoted by ‘int’ and ‘double’ respectively. They can be declared and initialized as shown:

```
int x = 2;
double y = 5.1;
```

3. Type Conversions

```
int i = Convert.ToInt32(Console.ReadLine());
char b = Convert.ToChar(Console.ReadLine());
```

What could go wrong when we use the above in a program?

4. Arithmetical Operations

Most calculations can be done using the standard operators: +, -, *, /. A typical program statement would be:

```
x = x + y;
```

Note that this is **not an equation**. It is an instruction that says ‘Add the value of y to the current value of x and store the result in x’.

We often want to add 1 to the current value of a variable (incrementing). This can be done as e.g. i = i + 1; Or more simply i++; A variable can be decremented using i--;

The usual rules of operator hierarchy apply e.g. 3 + 4*6 -1 is calculated as 3 + 24 -1. Brackets can be inserted around parts of a calculation to ensure that the part within the brackets is calculated first.

You should avoid mixing integer and floating point values.

5. String Operations

Two useful operations are Substring and Length as illustrated below:

```
static void Main(string[] args)
        {
            String testStr,newStr;
            Console.Write("Input a 5-letter word ");
            testStr = Console.ReadLine();
            newStr = testStr.Substring(1);
            Console.WriteLine(newStr);
            newStr = testStr.Substring(1, 2);
            Console.WriteLine(newStr);
            int i = testStr.Length;
            Console.WriteLine("The string {0} has {1} letters",testStr,i);
        }
```

Practical Exercises

These exercises involve some simple console programs to establish the basic principles. There are some exercises that we shall work though within the tutorial – some will involve working in pairs. This will be followed by further exercises. Some of these are assessed (core exercises) and others that are a bit more challenging that you can use to further develop your programming skills. All the practical exercises are available with a Visual Studio Solution that presents the individual exercises within a framework that outlines the steps involved and gives you the basic program structure to start your implementation. This also contains any relevant notes and an indication of what you are expected to submit for your internal assessment. This aspect of the module is additional to the core skills requirement but it is hoped that most apprentices will attempt at least some of the further exercises. It is expected that students will explore other on-line resources, although there should be sufficient material within that indicated elsewhere in this course. Students are invited to submit any further work for informal internal assessment which can be used as a positive inclusion towards their employment profile.

Notes on Good Practice

Choose variable names that indicates their purpose e.g. myName. It is common practice to choose variable names such that they begin with a lower-case letter but use an upper-case letter for a second part, for example: int myAge; Variables should be initialized where necessary to avoid unpredictable results. This can be done when they are declared e.g. int i = 0; Note that you can also declare and intialise more than one variable of the same type in one statement e.g. float x = 2.5, y = 3.9;

Always pay attention to program layout using suitable indentation to clearly indicate structure. Include comments when they can usefully provide an explanation of what is being done.

Assessment

In order to ensure that you are making satisfactory progress on this course, you are required to demonstrate your understanding of core concepts as well as your ability to design, develop and test various software implementations for given scenarios. Further details can be found in the 'Core Exercises' document on Moodle.

Links for Further Study

The material contained in the introduction together with this handout should be sufficient for you to successfully complete the core exercises. If you wish to learn more, the following links are recommended:

C# Tutorial and Programming:

http://www.completecsharptutorial.com/

Video Introduction to Programming in C#:

https://www.youtube.com/watch?v=OBsGRqXzOhk