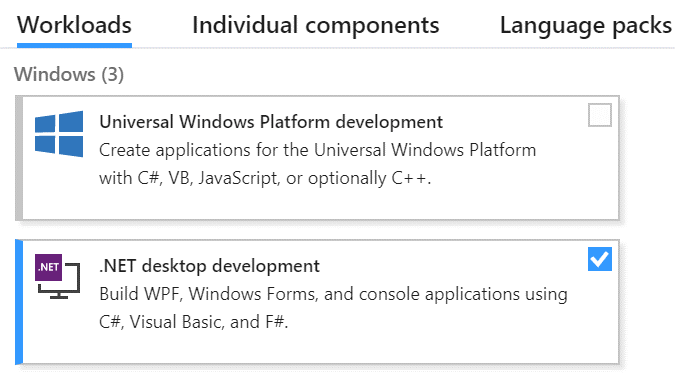
# Chapter 1 – Welcome to C#

C# is an object-oriented language that enables developers to build a variety of applications that run on the **.NET Framework**. You can use C# to create Windows applications, Web services, mobile applications, client–server applications and database applications, to name but a few. Game developers can use C# and Unity to create 2D and 3D games. In order to teach some of the fundamentals of programming through C# this guide will use the console mode application. This is a type of application where output is to a plain command line type interface.

## Downloading a C# IDE

An IDE is an integrated development environment – a software application that is used for software development. The IDE shown in this guide is Visual Studio Community 2017. A free download is available at <https://visualstudio.microsoft.com/vs/community/> After downloading the IDE, make sure that you install the .NET Desktop Development (see Image 1 below) as you will need this to access the correct project templates.

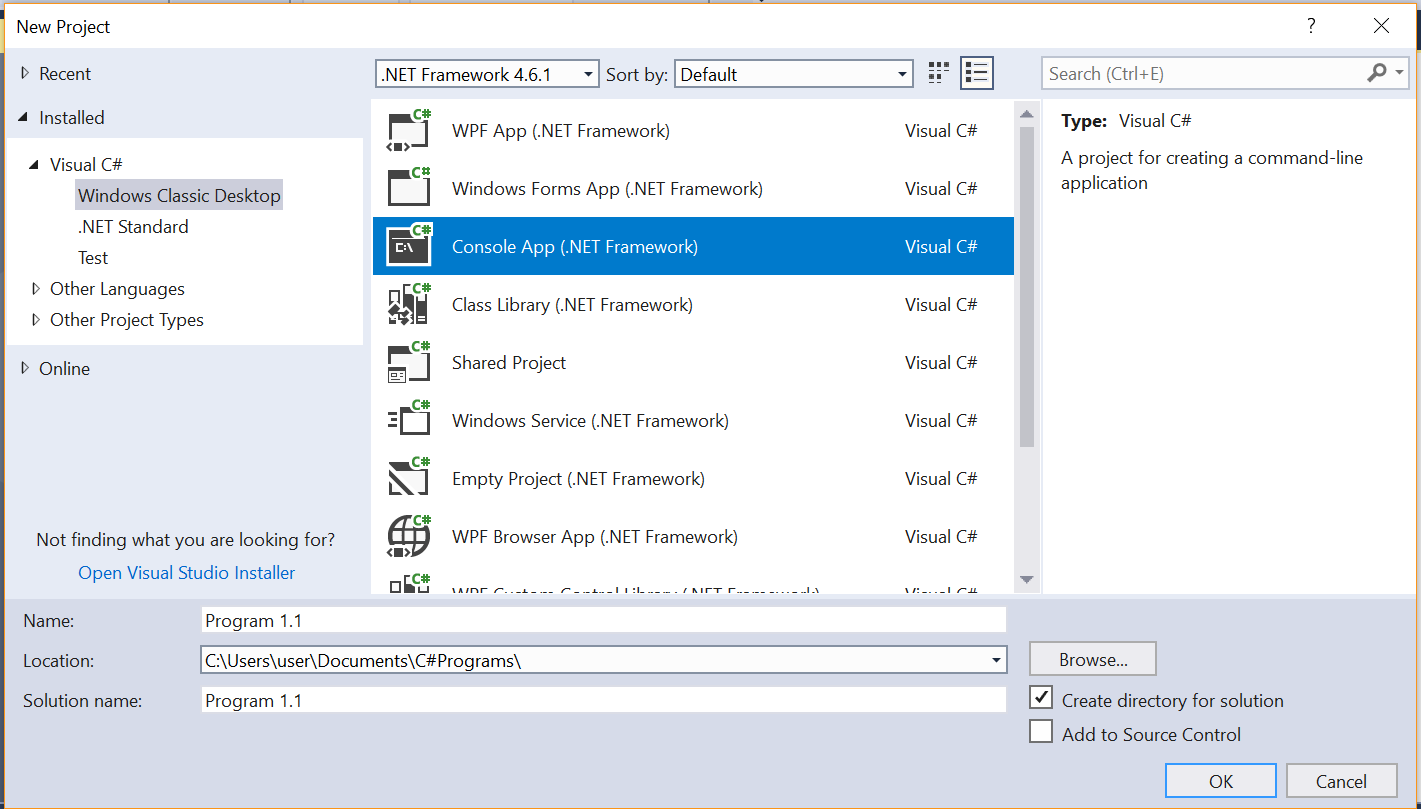
**Image 1 – Visual Studio Installer**



## Opening a new console mode application

Before you can start to code, you need to open a new console mode application. This can be done from the Start Page or by going to the File menu and selecting New – Project. It is then necessary to select a language and mode (C# Windows Classic Desktop) and then a project type (Console App). Image 2 below shows how this is done.

**Image 2 – How to choose a new console mode application**

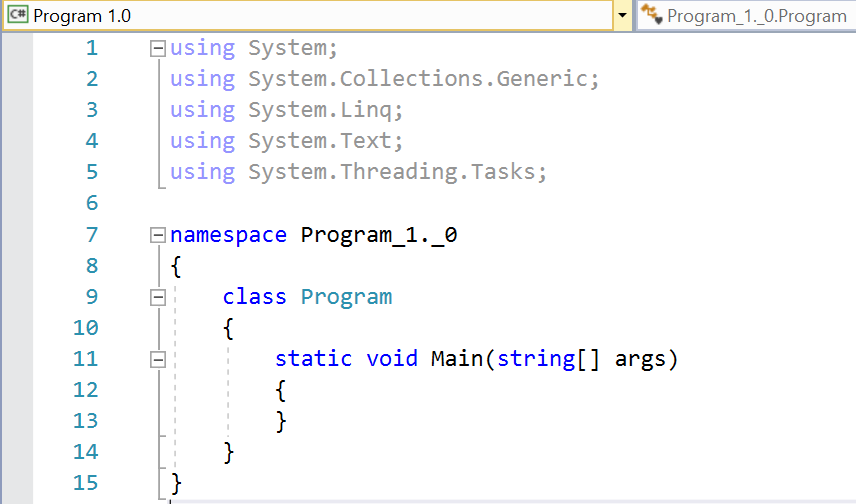


Projects should be appropriately named and saved in a suitable location

## The console mode application template

Image 3 below shows the console mode application template.

**Image 3 – console mode application template**



**using** – allows the use of other object types in this namespace

**namespace** – allows the organisation of related code elements

**class** – code that defines the methods and attributes of an object

**static void Main** – a procedure in console mode where execution of the program always starts

*Until you get to lesson 7 – the code we write will go between these two curly brackets*

**Blocks of code** (such as namespaces, classes, procedures, loops and selection statements) are defined using curly brackets {} – one to open the block and one to close the block



**{ }**

**Coding Task: *Program 1.1 – Outputting to the console window***

Complete the program below and run it. Try changing the string (“hello World”) to be output.



A line of code that completes an action is called a **statement**. Each statement in C# must end with a semicolon.

Note the curved brackets after **Console.WriteLine**. We use these brackets to **pass data or arguments** to a method. In this example, the string argument “hello World” is being passed to the WriteLine method. **String data** is enclosed in speech marks.

We can start the execution of our program by clicking the Start button on the toolbar.



The Console.ReadLine() statement at the end of the Main procedure is a tactic employed to keep the console window from closing immediately the program finishes executing so we can see our output. It is also used to read data into our program from the console window as we see in program 1.2.



**{ }**

Coding Task: *Program 1.2 – Input and output*

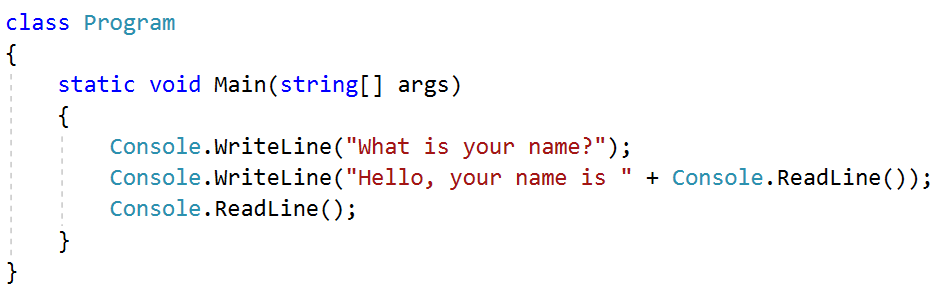
A more likely scenario for a program is that we will need to input and output.

|  |
| --- |
| **Pseudocode** |
| OUTPUT “What is your name?” |
| INPUT name |
| OUTPUT name |

This can be achieved in two different ways (see below) – try each method.

**Program 1.2 v1**

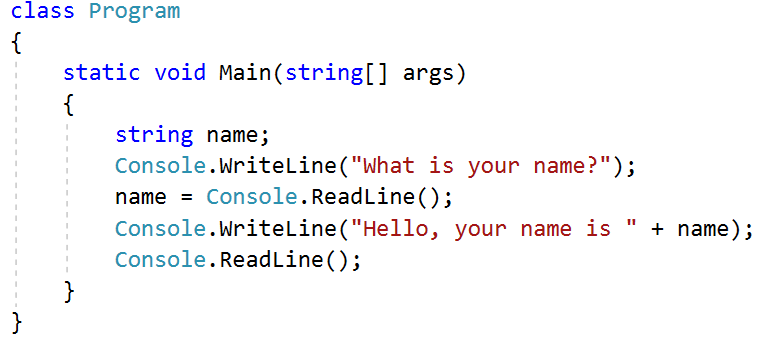
Try changing the question to achieve different outputs.



In C# we use the **+** symbol to join (concatenate) strings together. In this case, the literal string “Hello, your name is” will be joined with **reading in** from the console window whatever the user wrote in response to the question “What is your name?”.

**Program 1.2 v2**

This version of the same program uses a **variable** to store the input name. A variable is a storage location in memory that is used to hold values used in our program that might change. We will learn more about variables in Chapter 2.



Here we are using a string variable called ‘name’ to hold whatever is read in from the console window in response to the question. This value is later joined (concatenated) with “Hello, your name is” and output to the console window.

## Concatenating strings using placeholders

In the previous programs we used the + operator to join strings together. When it is necessary to join

multiple strings, it may be easier to use the placeholder method.

**Method 1 – joining strings using the + operator**



**Method 2 – joining strings using a placeholder for variable values**





The numbers in the curly brackets refer to the contents of the variables stated after the end of the string. **{0}** is the first declared variable, **{1}** is the second declared variable, etc.



**{ }**

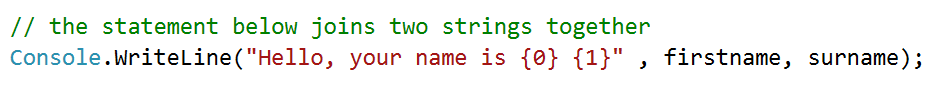
Coding Task: *Program 1.2*

Try changing program 1.2 so that it reads in a first name and a surname and uses the   
placeholder method to produce the output.

|  |
| --- |
| **Pseudocode** |
| OUTPUT “What is your first name?” |
| INPUT first name |
| OUTPUT “What is your surname?” |
| INPUT surname |
| OUTPUT “Your name is” and first name and surname |

## Adding comments to a program

Adding comments to a program allows us to explain, either to ourselves or to others, how our program works, and is useful for explaining tricky bits of code.



To add comments, we simply type in two forward slashes followed by the comments we want to make. The IDE software ignores these lines when compiling code for execution.



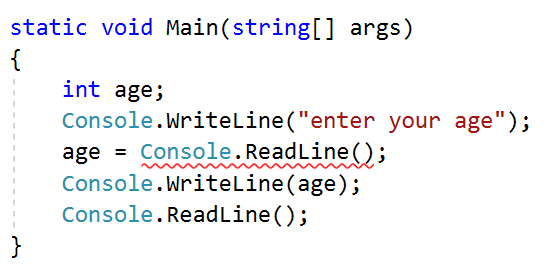
**{ }**

Coding Task: *Program 1.2 (continued)*

Add comments to program 1.2 to explain how your code works. Try to imagine that you are explaining   
to a non-coder how your statements are working in the program.

## Reading in non-string values and casting data types

So far in the programs built we have only read in string (text) values. Obviously, computer programs have to deal with many types of data (integers, strings, Boolean, characters and real numbers). While this topic will be covered in more detail in Chapter 2, let’s see what happens (in the program code below) when we need to read in from the console window non-string values.



This program will not even execute (a red underline indicates a build error) as the program is trying to pass a string value (from the console window) to an integer variable

The string value therefore needs to be **converted** to the data type of the variable we are trying to assign it to. See the corrected example below.



Alternatively, the **Convert** method can be used for many different data types. The example below converts to a 32-bit integer.





**{ }**

Coding Task: *Program 1.3 – Retro Sales*

Study the scenario below and the pseudocode provided, then build a program in C# to   
calculate the total price of a car.

*Retro Sales is a local car dealership which sells a range of retro cars. It wants a program it can use to* ***provide customers with the total price of a new car****.* ***VAT is charged*** *at a fixed rate of 20% of the cost of the car.*

**Models Sold:** Retro Original (£16,500) / Retro Panoramic (£17,500) / Retro Cabriolet (£25,000)

**Trim Package Costs:** Silver (£2,500) / Gold (£1,950)

**Car Tax**: £30 for 12 months

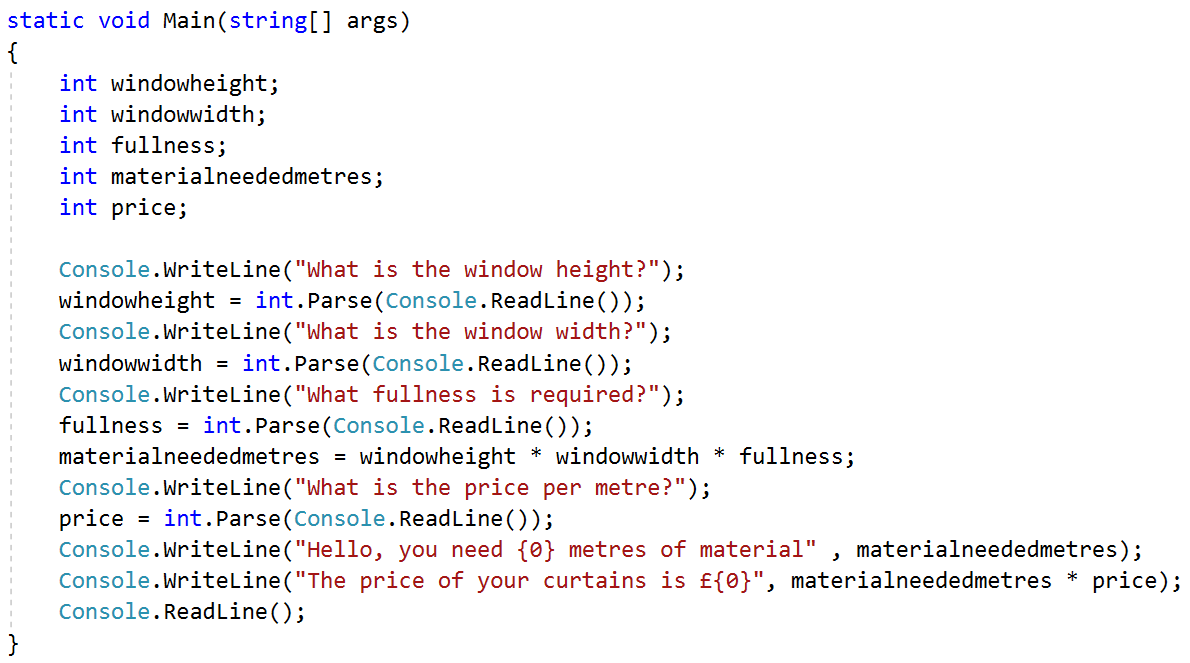
|  |
| --- |
| **Pseudocode** |
| OUTPUT “What model do you want?” |
| INPUT Modeltype |
| OUTPUT “Please enter the price of your model” |
| INPUT Modelprice |
| OUTPUT “What trim package do you want?” |
| INPUT Trimpackage |
| OUTPUT “What is the cost of your trim?” |
| INPUT Trimprice |
| CarTax = 30 |
| Price = ModelPrice + Trimprice + CarTax |
| OUTPUT "The model you have chosen is " and model name |
| OUTPUT "The trim you have chosen is " and Trim name |
| OUTPUT "The price for your car is " and price |

## Chapter 1 - Consolidation Tasks

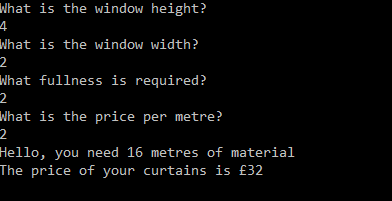
Program 1.4

Sarah makes curtains for a living. She has asked for a calculator that enables her to work out how much material she needs to make a set of curtains for a particular window, and how much that amount of material will cost her.

Study the code for the program below and answer the questions underneath.



The program **produces this output in the console window** when the curtain window height is 4 metres and the width is 2 metres. The fullness required is double the width, and the price of the material is £2.00 per metre.



1. Explain how the program outputs to the console window in this program. **[3 marks]**

1. Explain how values typed in by the program user are input into the program. **[3 marks]**

1. Why have some input values used the **.parse** method? **[2 marks]**

1. Sarah wants to make curtains for a window with the following values:

**Window height:** 8 metres

**Window width:** 3 metres

**Fullness:** 2

**Material price:** £4.67

**When the program runs with these values it crashes**.

Explain why this happens and suggest ways to avoid it. (TIP! You may find it easier to   
build the program to see what happens.) **[3 marks]**

1. This question will involve you **building program 1.4 using the text file provided**. Sarah now wants to calculate the VAT payable on the curtains – **add some code** to this program that will perform that job for her. The final output should now be the price plus the appropriate amount of VAT.

VAT should be calculated as 20% of the price. **[4 marks]**