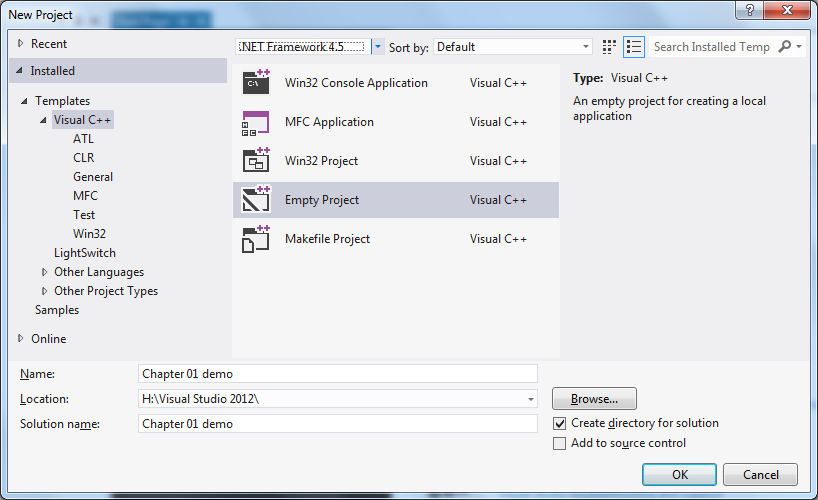
***Chapter 1* Introduction - Output and Input**

Let's demonstrate how to get started with Visual C++.

From the Start menu choose Visual Studio 2012.

Choose Empty project and Browse for the path where your program will be stored:-

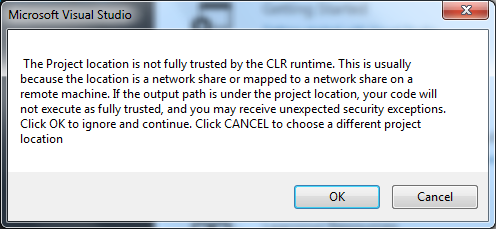


**Where to store your projects?**

* If you store all your projects separately within your own **H: drive** then you will use a lot of space.

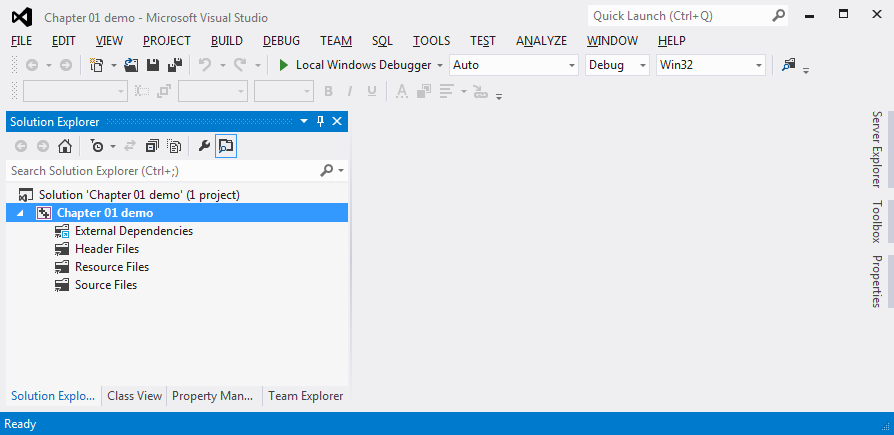
(I use one generic project and Paste specific code into it as required. This uses much less space but I do need to remember to store my code files individually.)

* You can access the **C:\Temp** folder but of course so can anyone else who uses the local PC. So again you need to make a separate save of any important work.
* You can store your data direct to an **external HD** if you wish. However this takes you out of the college backup system. All H: drive content is backed up overnight so we can recover important work if you delete it accidentally.

In this case I have chosen to store my work in my Home drive H:

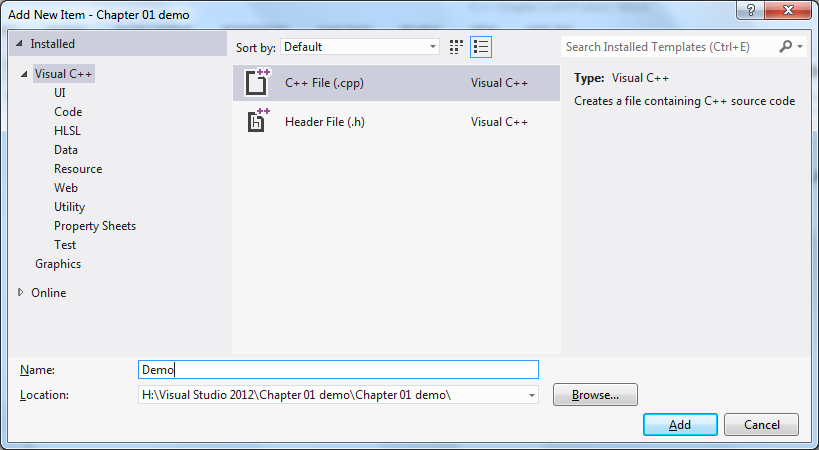
Of course this is a networked drive so Visual Studio gives a fussy warning about security over the network. You can ignore this message:-

The IDE looks like this when you open it:-

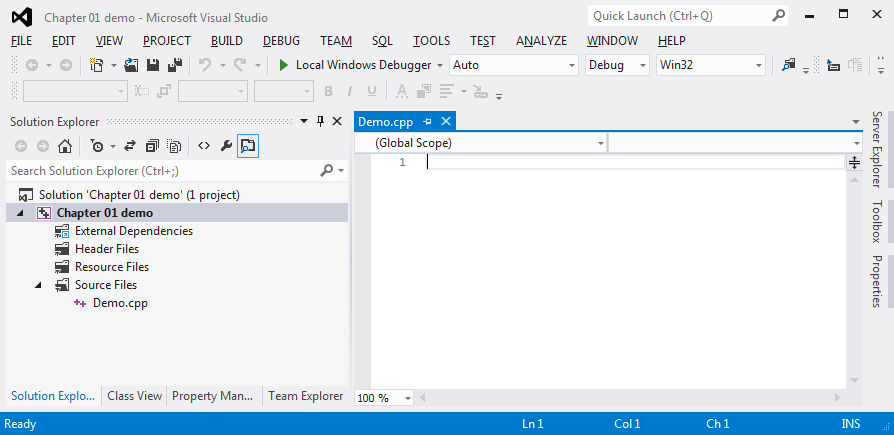


The main code area of the IDE is greyed out – we can't access it yet to enter our code.

Before we can do that we have to add a Source file to our project – the file which will store our actual C++ code.

Choose Project and then Add New Item. Choose C++ file and give the program a name:-

Now - at last - we are ready to add some code:-



Here's a first simple C++ program:-

**// A first demo C++ program**

**// PY August 2014**

**#include <iostream>**

**using namespace std;**

**void main()**

**{**

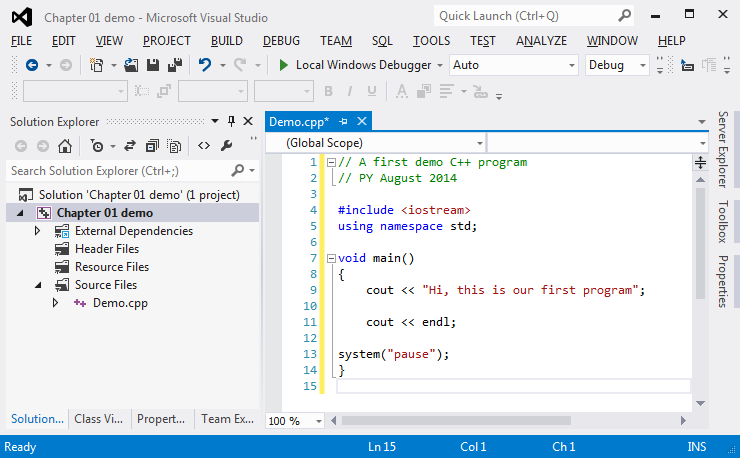
**cout << "Hi, this is our first program";**

**cout << endl;**

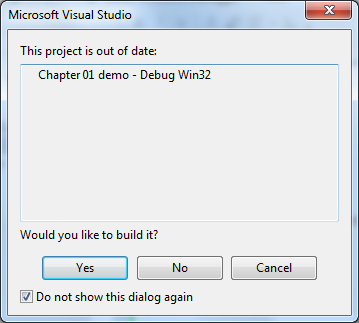
**system("pause");**

# }

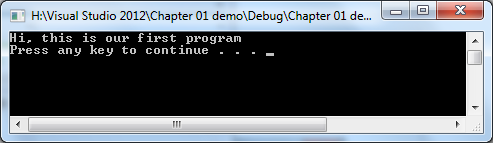
Type this code into your source file as accurately as you can. You will notice that Visual Studio immediately recognises some parts of the code and colour marks them:-



Before we can execute the program it must be **compiled**. We can achieve this from the Build menu (Build solution – F7) followed Debug (Start debugging – F5). Or we can just click on the green triangle which will force a compile if needed and then run if possible.

You will see a dialogue box and should answer Yes:-

If your code meets the rules of C++ then it will compile without errors and if you have typed it accurately then you should see the following very exciting output window:-



Was it worth all that effort?

If compilation reports errors then have probably made a typographical error in your code and you will need to fix it before trying again to Compile / Run your program.

Only when you have a clean compilation (no errors) can you actually run your program. Note that just because it compiles (i.e. meets the rules of the language) doesn't mean it will do what we want it to – or even do anything at all!

DOS windows are very old fashioned now but for the purposes of this unit we will be using C++ in this rather limited way to keep things simpler.

Of course this program is unrealistically trivial – it has no user input. Every time you run it you will see the same output.

Now we have seen what it does let's examine the code of the program.

There are several things in this program which have to be there in order for the compiler to accept it as a program. Just accept that all your programs will have the following:-

1. One or more **#include** … lines

In our case this will always be these two lines:-

**#include <iostream>**

**using namespace std;**

2. The line **void** **main()**

3. Braces **{ }** or curly brackets which enclose your program statements

4. A semicolon **;** at the end of each statement - to tell the compiler where the statement finishes

Note however that the **#include** line does not end with **;**

5. The final **system("pause");** line

All of our programs will also have those first few lines which start with **//**. These are **comment lines** and act as a label for the human reader to understand what each program is about. C++ doesn't need these lines – in fact it ignores whatever you write here. But you must get into the habit of including some basic info every time:-

* Description or purpose of the program
* Your name and the date

Even if you do not fully understand these yet it is helpful to see what happens if you make a mistake in putting them in. It will speed up your error seeking and detection later on if you spend time on this activity at this early stage.

First try the program and get it to work.

Then try making some of the deliberate mistakes (next page) and see how the compiler responds when you try to compile the program. Notice what you do and any error message(s) you get. Do any of the error messages make sense? If the program compiles without errors, what happens when you run it?

Try each of the changes in the following list and do some detective work to discover the effect of each change.

Note of the observed effects of the change. Does it cause a **compilation error** (i.e. the program won’t run at all) because we have broken the rules of C++ language? If you do get a compilation error have a good look at the error message.

Some of the error messages are very straightforward. If you miss out a semi-colon from the end of a line you will get a message saying **'; missing'**. But others are less helpful. Try to get into the habit of using the messages to remove any typing errors that creep into your code.

Some of the changes don't prevent the program compiling. They might have an effect at **run-time** – in other words the program runs but what it does is changed. And at least one of these changes has no effect whatsoever.

Don't forget to return your program to its original state after each change.

1. Miss out the line **cout << endl;**

2. Miss out the line **#include <iostream>**

3. Miss out one or both of the angle brackets around **'iostream'**

4. Miss out the line **void main** ...

5. Miss out one or both braces

6. Miss out one or more semicolons

7. Misspell the word **'main'**

8. Misspell the word **'void'**

9. Misspell the word **'cout'**

10. Misspell the word **'program'** in the **cout** line

11. Put a space between the '**o**' and the '**u**' of '**cout**'

12. Put a space immediately before the **'H'** of **'Hi'**. Then try five spaces

13. Try any other deliberate errors you can think of

*// Here's a second program*

*// PY Aug 2014*

**#include <iostream>**

**using namespace std;**

**void main()**

**{**

**cout << "This is the first line";**

**cout << "This is the second line";**

**cout << endl;**

**system("pause");**

**}**

What happens when you run this program? Now try a little change:-

**…**

**We’ve added \n**

**cout << "This is the first line\n";**

**…**

When you run the program notice the effect of adding '**\n**'. The '**\**' tells C++ to treat the next character as a special symbol - in this case a **new line** character.

The '**\n**' has the same effect as including **<< endl** in **cout**. Later in these notes we’ll use **endl** but you can use '**\n**' if you prefer.

What happens if you accidentally use a forward slash '**/**' instead of backslash?

Try adding another '**\n**' after the first one. What does that do?

What happens if you put a '**\n**' between the words '**second**' and '**line**'.

Try adding a **cout** statement containing only '**\n**' between the first **cout** statement and the second in the above program. What does that do?

Try including **'\t'** in the one or more of your **cout** lines. What does **'\t'** achieve?

Use all your knowledge to write a program which will display the following on screen:-

**These** **lines** **are**

**stepped** **in**

**by** **3** **characters**

**each** **time**

**Input and calculations**

Those first demo programs are not useful. They always show the same output. In real life we need programs to accept input data and vary their output depending on what input is received.

For example let’s say we want a program which accepts two numeric inputs from the user and adds them together before outputting the total. We don’t need to worry about the context but this is the sort of simple calculation needed frequently in programs / web scripts / apps etc.

In design terms we could summarise this problem as follows:-

Get first input from user

Get second input from user

Calculate answer by adding the two inputs

Show the answer to the user

This form of design is called Pseudocode – a complex name for a simple technique. We just jot down what we need to do in what order. We are then free to work out how to achieve each stage in C++.

In this case the design follows a very common pattern – Input -> Process -> Output. Which lines correspond to these three stages?

On the next page is a completed solution in C++ for this problem.

Read through the code carefully and try to work out what each line is doing.

Can you identify which lines equate to the four lines of our Pseudocode design?

// Program to introduce variables and input / output

// Chap01 Add.cpp

// PY Aug 2014

#include <iostream>

using namespace std;

void main()

{

// Declare the variables we will use

int inputScore1;

int inputScore2;

int outputAns;

cout << "Maths demo" << endl << endl;

// Get the input data from the user

cout << "Enter the first score:- ";

cin >> inputScore1;

cout << "Enter the second score:- ";

cin >> inputScore2;

// Calculate the answer

outputAns = inputScore1 + inputScore2;

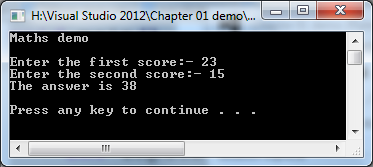
// Output the answer

cout << "The answer is " << outputAns;

cout << endl << endl;

system ("pause");

}

When it runs:-

There are important new features here which you will use in every future program you write:-

1. Variables

inputScore1, outputAns

1. Input statements

cin >> inputScore1

1. Assignment or calculation statements

outputAns = inputScore1 + inputScore2;

1 A **variable** is a **box** in memory where we can store data. We don't need to know where in RAM it is stored - we invent a symbolic name to tell C++ which data we mean. We also tell C++ what type of data we will store there. In this case we are using integers – whole numbers.

2 Input is achieved by **cin** (just as cout is used for output). The first cin tells C++ to:-

* Get the user's data from the keyboard buffer
* Store it in whichever area of memory has been allocated to inputScore1.

3 The equal sign has different meaning in C++. It acts as a **command**. It says "take the value of the thing on the right and store it in the variable named on the left".

So in this case C++ fetches the contents of the two inputs and adds them before storing the total in outputAns. It has assigned a value to outputAns.

These are three important ideas in programming so if you understand them already you have made an excellent start.

**Simple maths exercises**

Get the addition program working satisfactorily and test it with various combinations of inputs.Check that the results are correct. Try to find inputs which don't give the correct answer.

Then amend the program so that it subtracts the second score from the first. Test that thoroughly.

Then make a version which multiplies the two scores together. Test again.

Finally try to divide the first score by the second. If you test this carefully you should detect a problem here. Can you explain why this problem can occur?

**More maths exercises**

Design and create C++ programs to meet the following specifications.

In each case you should include:-

- appropriate **descriptive data names**

- some brief **comment lines** to describe and explain your script

- **good layout** to make it easy to read your code.

In each case **test your program thoroughly** to check that it meets the specification for all appropriate numbers. Remember that to test properly you must decide what data to use **in advance** and know what answer to expect from your program – otherwise you will just assume the program is correct. We should draw up a **test log**.

1.1 Design and create and test a script to prompt the user for the dimensions of a room in metres and output the **total floor area** of the room in square metres

1.2 Square any input number.

1.3 Cube any input number

1.4 Convert an input weight in kilos into lbs (1kg = 2.2 lbs).

1.5 Convert an input length in inches into metres (1m = 39.3in).

1.6 Convert an input length in feet and inches (two inputs) into metres.

1.7 Convert an input distance in miles into kilometres (5 miles = 8 km).

1.8 Convert an input distance in kilometres into miles.

1.9 Convert an input sum in pounds sterling into euros (£1 = €1.38).

1.10 Convert an input volume in litres into pints

(“A litre of water’s

A pint and three quarters”).

1.11 Convert an input temperature in degrees Celsius into Fahrenheit

(0o C = 32o F and 100oC = 212oF).

1.12 Convert an input temperature in degrees in Fahrenheit into Celsius.

1.13 Convert an input fuel consumption figure in m.p.g. into k.p.l. (1 gallon = 8 pints).

1.14 Convert an input fuel consumption figure in m.p.g. into l.p.100km

1.15 Convert an input speed from m.p.h. into f.p.s.

1.16 Calculate a percentage from two inputs e.g. inputs 37 and 50 – output = 74%

Some of these later problems are hard to test because most of us have little idea how many kpl is equivalent to 30 mpg for example. You will need to do some calculations on paper or by calculator first to decide what test data to use and what results to expect.

1.17 Prompt the user for two inputs – the price of an item – the quantity ordered. Calculate the gross cost of the order and display that cost.

1.18 As above but also calculate the nett cost including VAT @ 17.5% on the total cost and display all three answers – gross cost – VAT amount – nett cost.

1.19 Prompt the user for the nett cost of an item including VAT. Calculate what the cost of the item would have been before VAT was added. Display all three costs.

1.20 Prompt the user for an annual salary figure. Calculate and output the Income Tax payable on that salary figure using the following rules:-

- The first £9,700.00 is exempt from income tax

- Income tax is paid on the remainder at 25%.

1.21 Prompt the user for a total budget figure and for the individual cost of an item e.g. a new laptop PC. Calculate and output the exact number of the item which can be bought within the budget:-

e.g. *Budget = £1000*

*Cost of item = £300*

*Number of items within budget = 3*

Note that we want the number of items – so the answer is 3 (not 3.333).

1.22 As above but also output the amount of money left in the budget left after the items have been purchased - i.e. in the above example there would be £100 left after purchasing 3 items at £300 each from a budget of £1000.

1.23 Prompt the user for the titles and prices of three individual DVDs. Calculate and output the total cost of the three prices.

1.24 As above but also prompt for the quantity of each DVD title ordered.

e.g. *DVD Title = Car Chase*

*DVD Cost £ = 11.95*

*DVD Quantity = 1*

*DVD Title = Greatest Goals*

*DVD Cost £ = 12.95*

*DVD Quantity = 5*

*DVD Title = Gardening For Beginners*

*DVD Cost £ = 24.95*

*DVD Quantity = 1*

*Total cost = £ 101.65*

1.25 As above but add VAT to the total cost.

1.26 As above but after calculating VAT include a post and packing charge. P&P is calculated at a basic charge of £1 plus £1 per DVD ordered.

e.g. In the above example the P&P will be £8 - £1 plus £7 for 7 DVDs ordered.

For the following problems use any convenient value for **п** (pi) – e.g. 3.142

Test your answers thoroughly.

Try to format your results to show and explain the answer clearly.

1.27 Calculate the area of a circle of any given radius

1.28 Calculate the volume of a sphere of any given radius

1.29 Calculate the volume of a cylinder of any given height and radius

1.30 Calculate the surface area of a cylinder of any given height and radius

1.31 Calculate the volume and surface area of a cone of any given height and radius