

Swinburne University Of Technology

LABORATORY COVER SHEET

Subject Code:	COS30008
Subject Title:	Data Structures and Patterns
Lab number and title:	1, Using Visual C++
Author:	Dr. Markus Lumpe
Edited by:	Carmen Chai

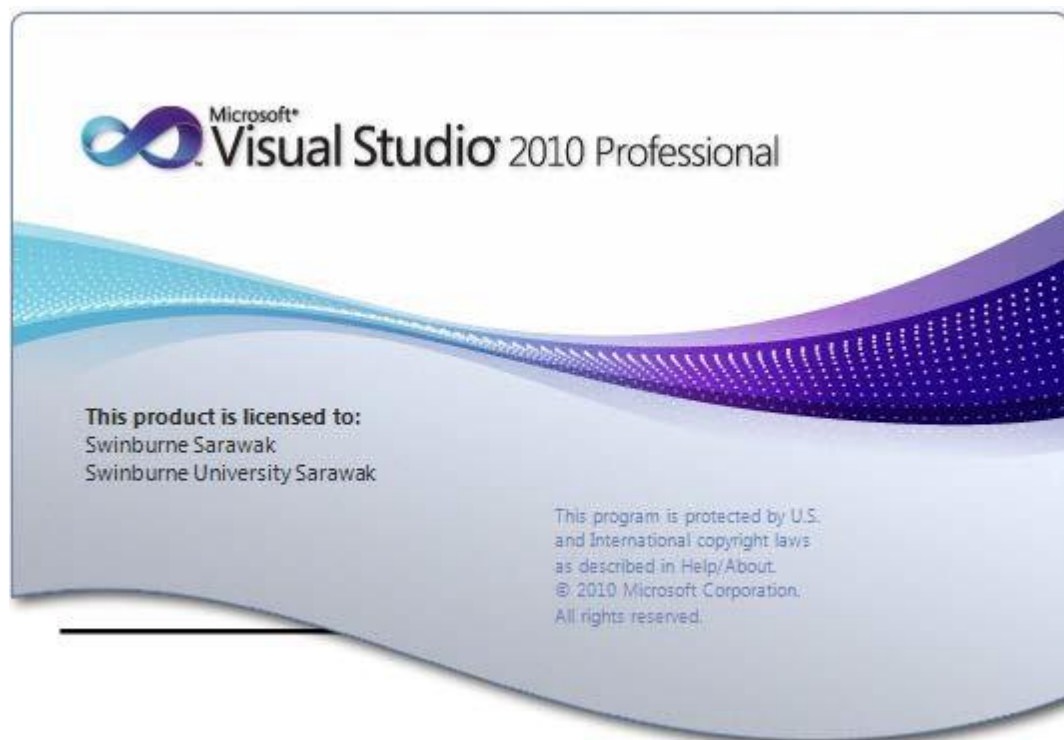


Figure 1: Visual Studio 2010 Startup Window.

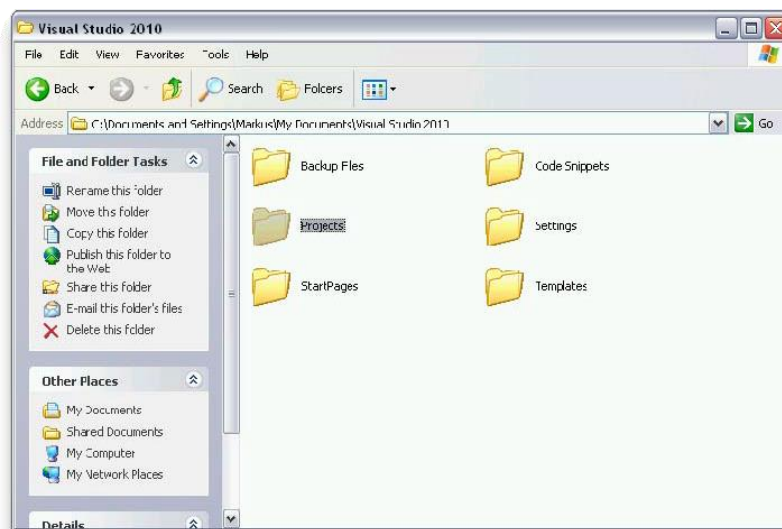
Where to get Visual Studio 2010

Visual Studio 2010 is available on all computers in the lab for this unit. You can loan the MSDN installation CDs from the library for installation into your own PC / Laptop.

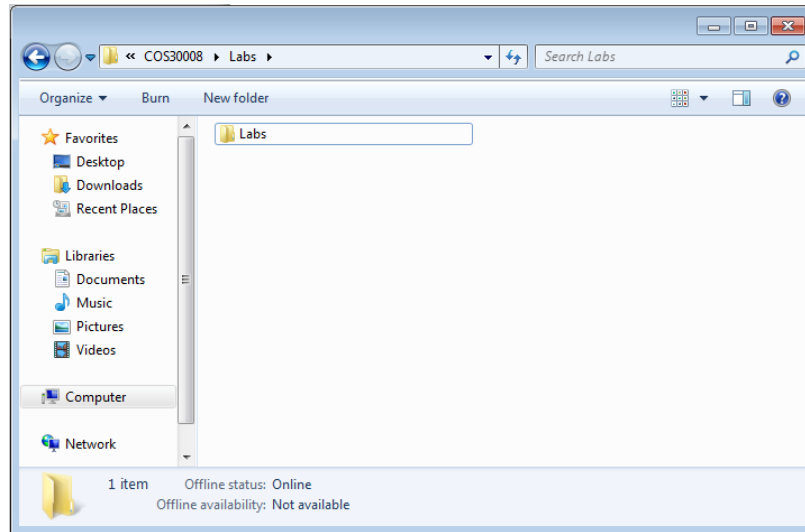
Lab 1: First Steps in C++ - Visual Studio

First Step: Select a working directory.

In order to develop a program, you need a [working directory](#). In Visual Studio 2010 (or Visual C++ 2010 express), this working directory is called [Projects](#) and located in the folder [Documents\Visual Studio 2010](#):

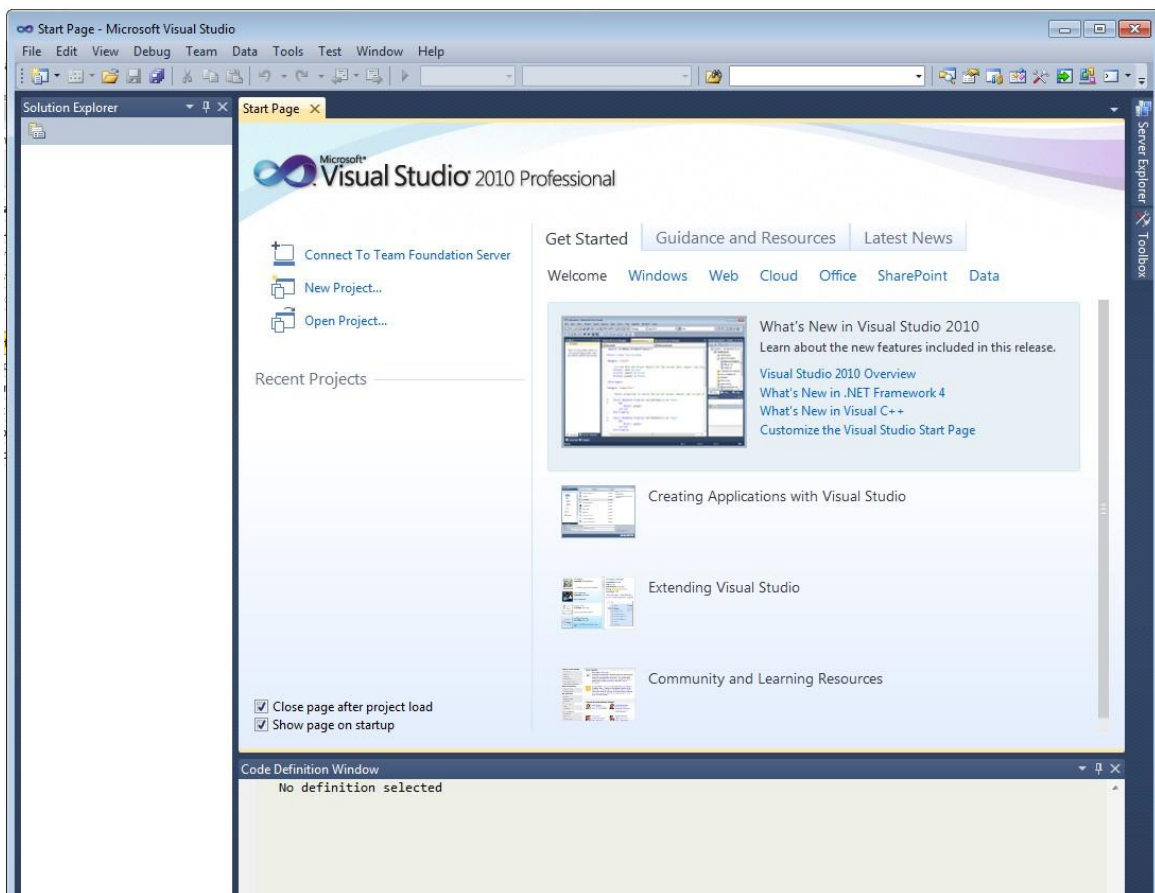


You can use this location, but in a shared environment like the lab computers in A308, the contents of this folder will be automatically delete when the computer shut down. Also, you may want to separate your projects based on you course work. For example, you may want to assemble all your files (i.e., lecture notes, assignments, and projects) in a semester-specific subject directory. For example, you could use [H:\Uni\COS30008\Programs](#) (or another suitable location, if you do not have write permissions for C:) as your working directory (and throughout this tutorial):



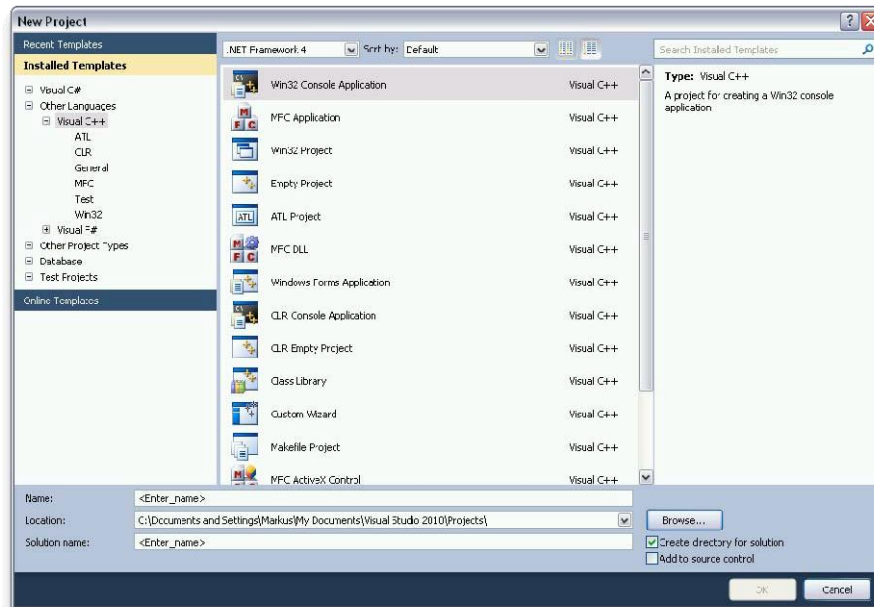
Second Step: Start Visual Studio and Select C++ Template.

Select Visual Studio 2010 from the Programs Menu and wait for the following screen to appear:

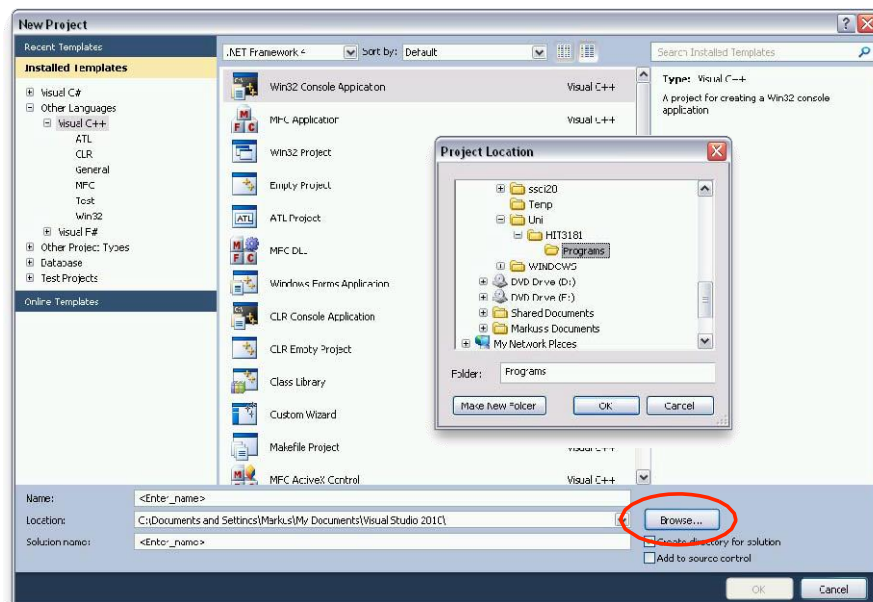


Select [New Project...](#) now:

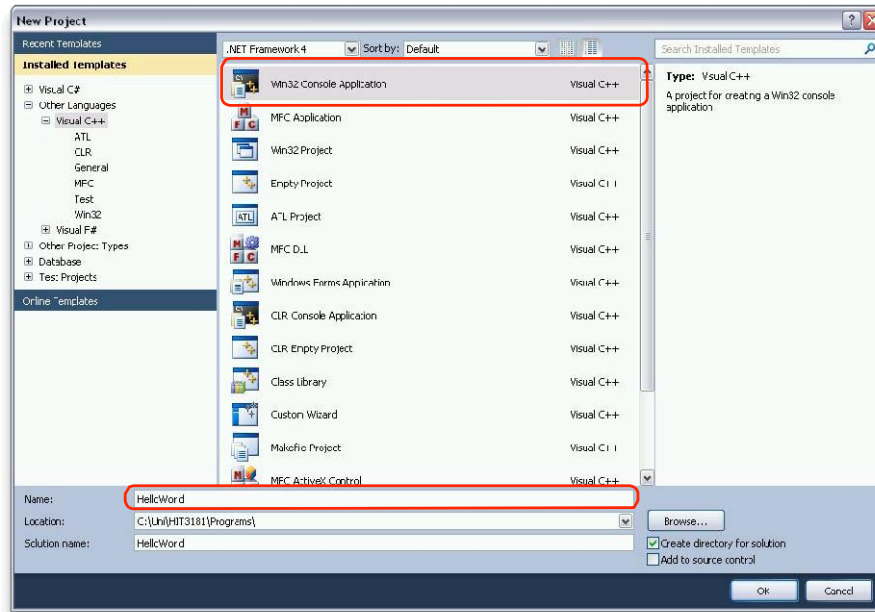
As a result, the New Project dialog window will appear. This is a so-called modal dialog window and it will stay on top of Visual Studio to force you to choose a proper project template. That is, you cannot do anything else before you have selected the type of project that you want to work on and you have completed the associated project-specific configuration steps:



You need to set the project location. That is click on the [Browse...](#) button, which will bring up the [Project Location](#) dialog. Select the desired folder:



Next, you need to select the project type. We will be using **Win32 Console Application** throughout the semester. Finally, you need to give the project a name. This name will also serve as the solution name (i.e., the name associated with the program we are developing). We shall use “**Hello World**”:

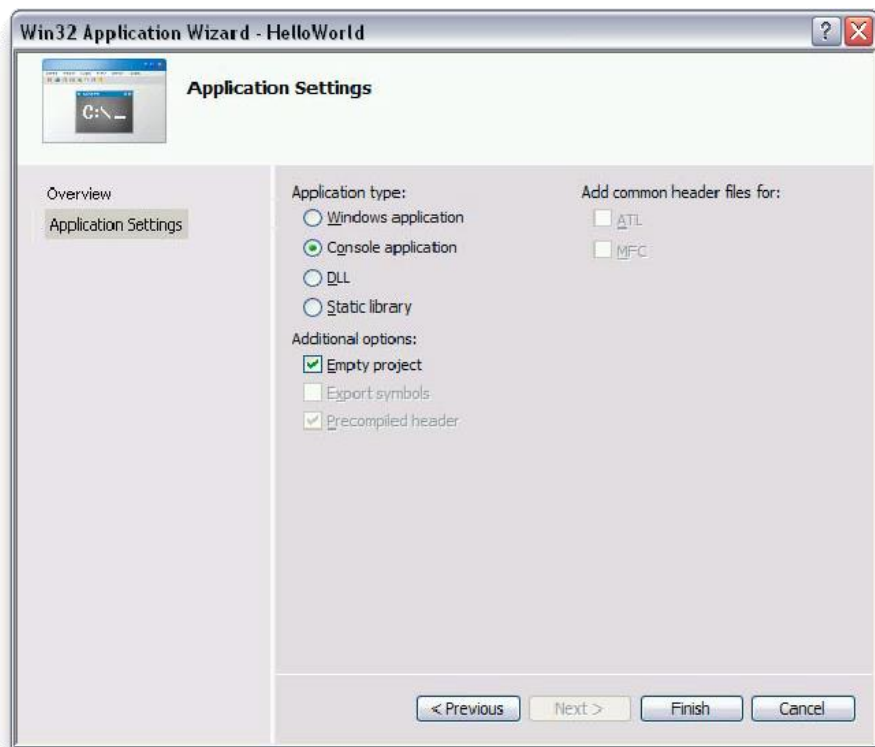


You should notice that the **OK** button is now enabled. That means, the basic settings have been completed. Press **OK**. Visual Studio will create the project. Yet, there is another step before we can proceed:

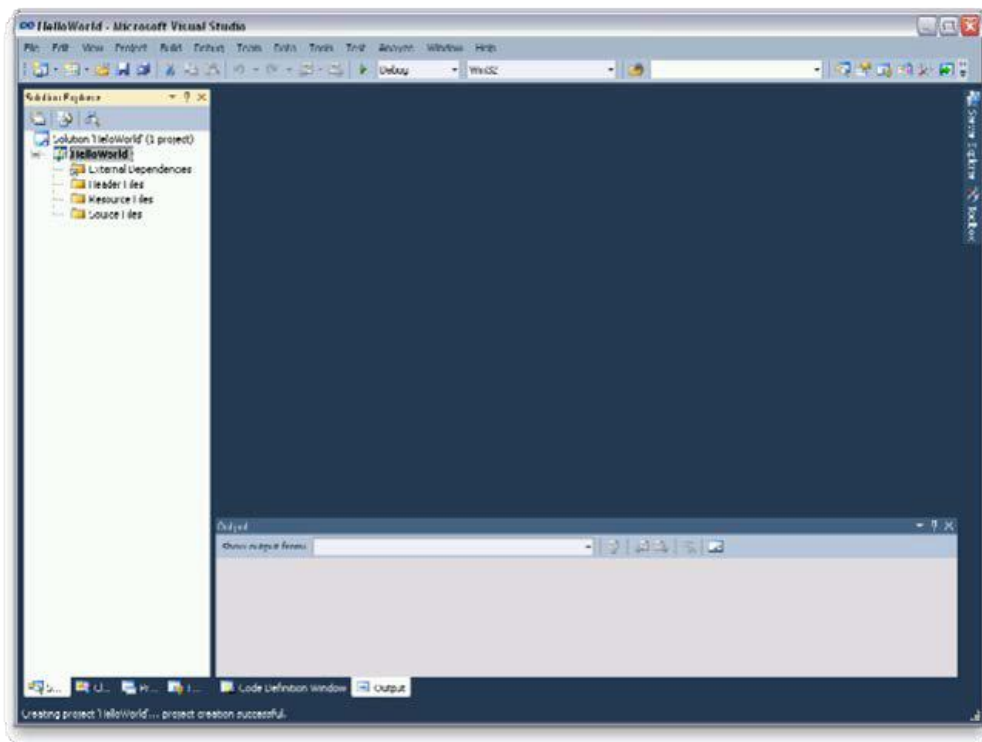


We see the [Win32 Application Wizard](#). This wizard allows for fine-tuning the project settings. We could just press the [Finish](#) button to accept the default, but this would be a bad idea. We should always inspect and control what Visual Studio is assuming about our intentions. Hence, we press the [Next](#) button.

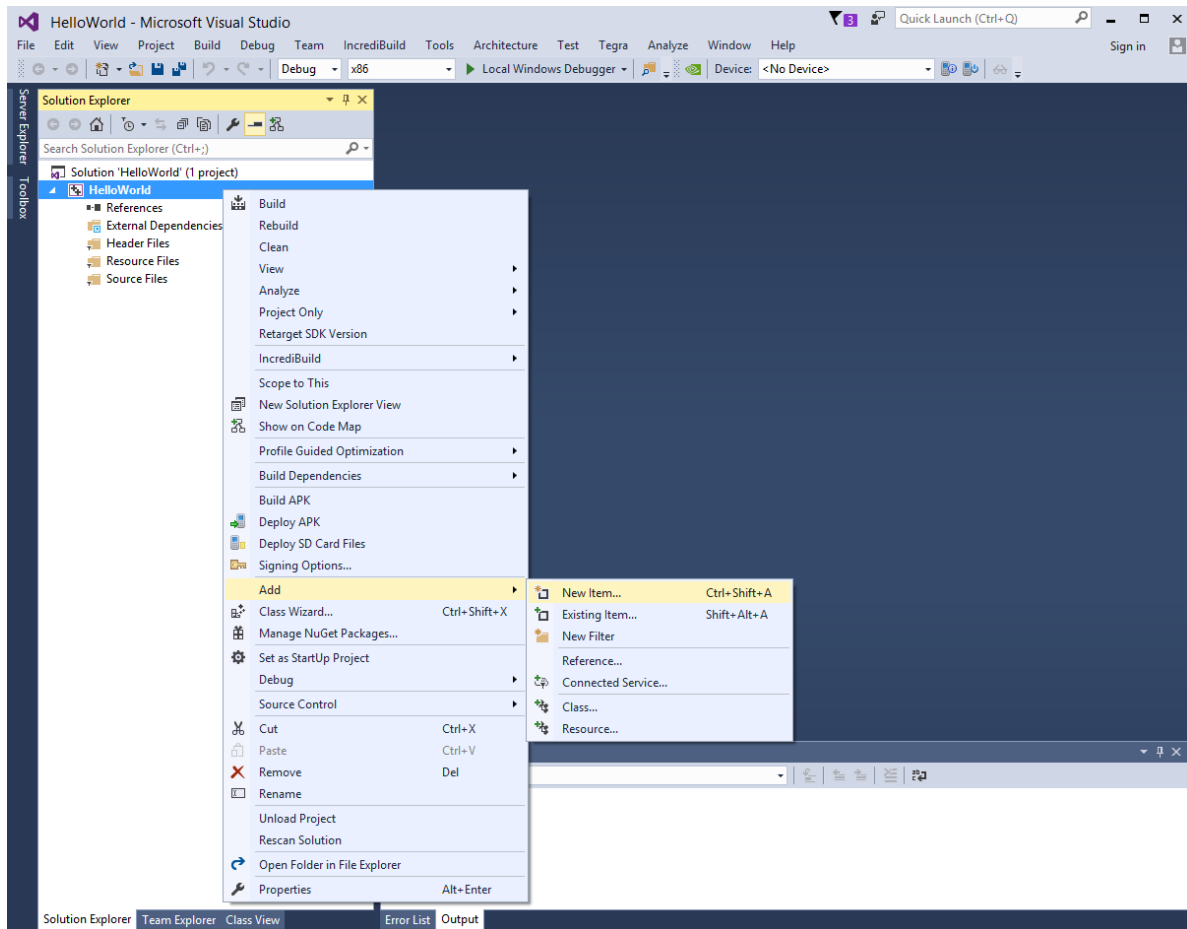
To keep things simple, we select [Empty project](#). This way we create an empty project – we are in complete control!



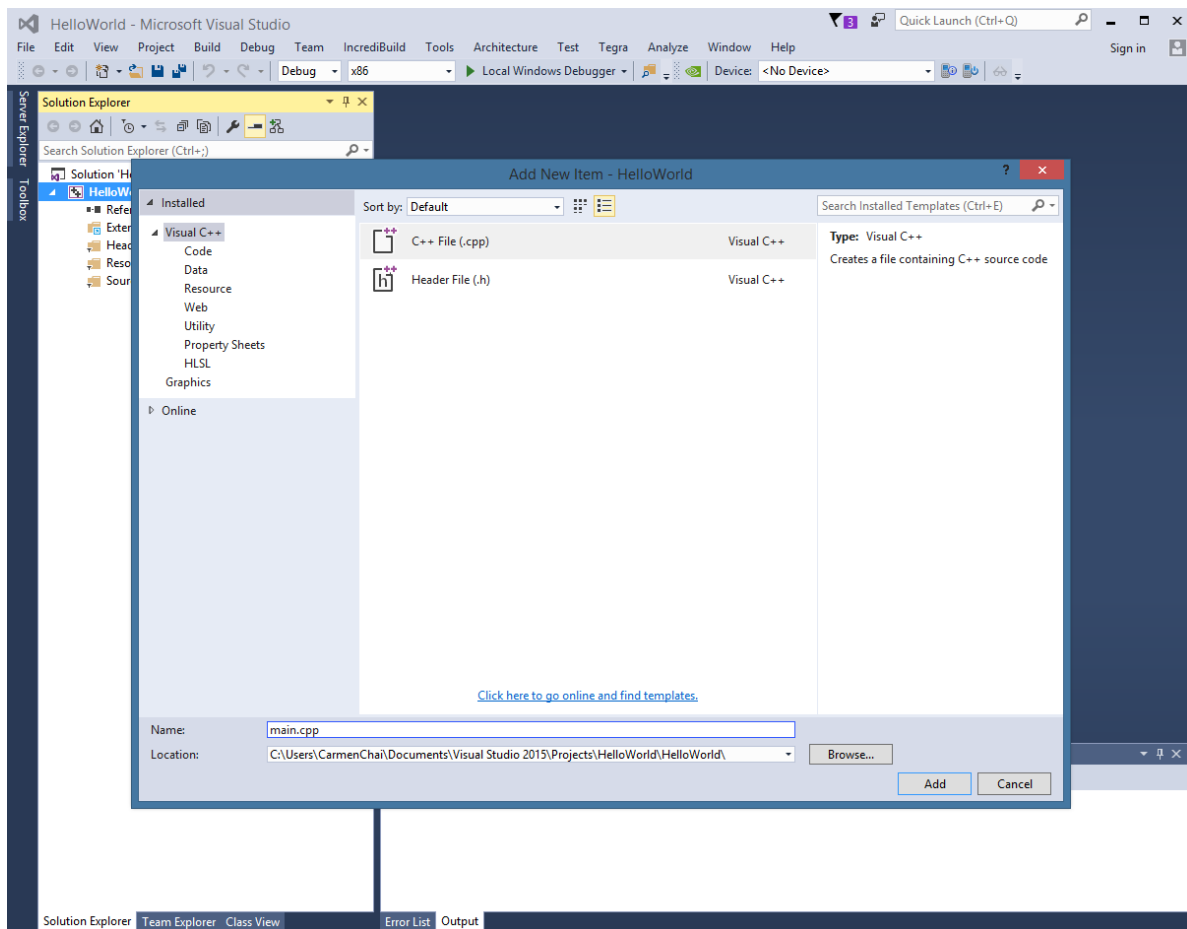
Configuration complete. There is just one option left: press the [Finish](#) button. The result should look like this:



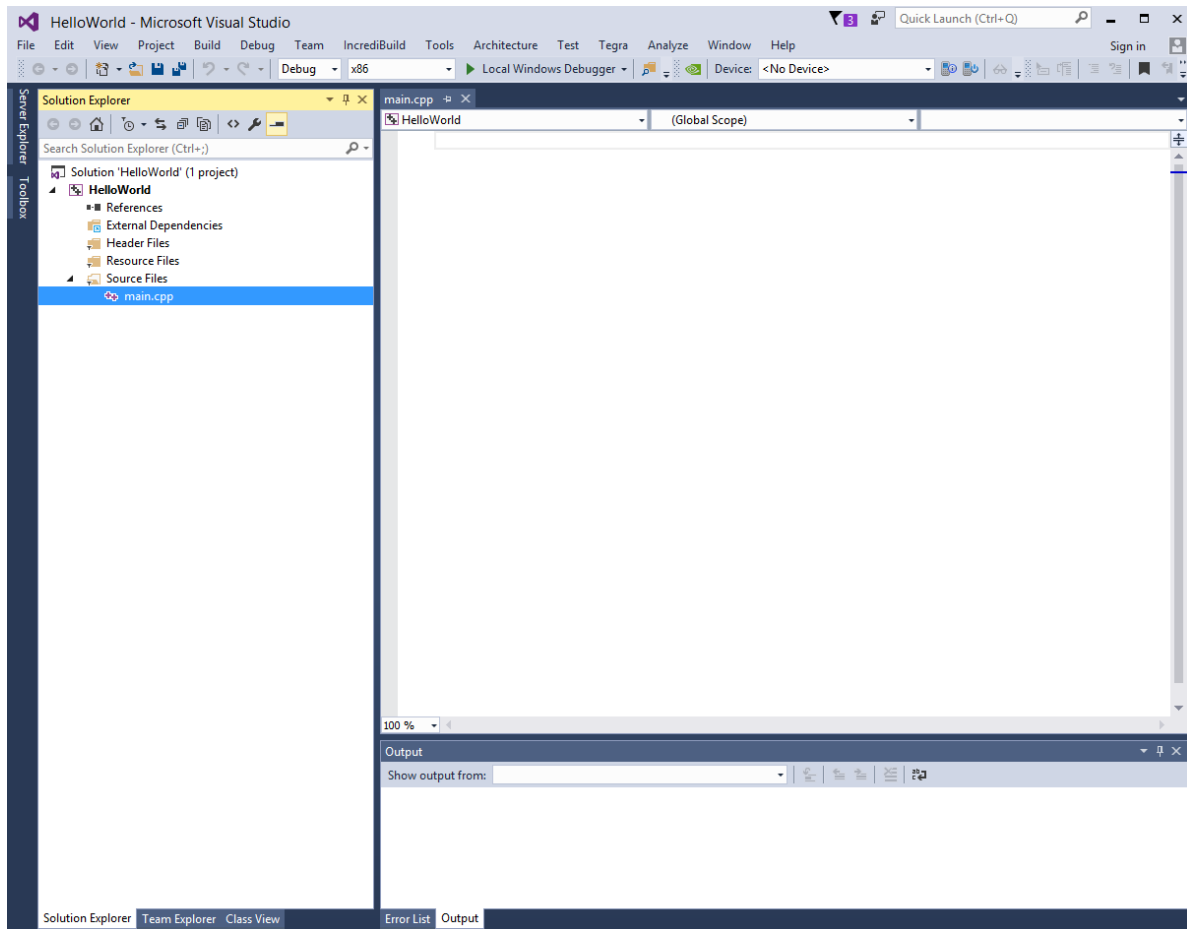
Now we add a **New Item**. We select **Code** of **Visual C++**, select **C++ File (.cpp)**, and type **main** as the name for the file:



Be sure to rename the file to “main.cpp” if it’s not. If we press the **Add** button, then an empty **main.cpp** file will be inserted into our project and Visual Studio will start the built-in code editor.



You will see the new file created under Source Files.



Third Step: Our first program.

```
main.cpp* X
HelloWorld (Global Scope)

#include <iostream>

using namespace std;

int main()
{
    cout << "Hello World" << endl;

    return 0;
}
```

In particular, we write:

<code>#include <iostream></code>	Include the definitions for console I/O. We are using two I/O values: <code>cout</code> (standard output) and <code>endl</code> (the environment-specific newline).
<code>using namespace std;</code>	Tell C++ to look for all library names in namespace <code>std</code> . Without calling this, we would have to type <code>"std::cout"</code> instead of only <code>"cout"</code> throughout the entire code.
<pre>int main() { cout << "Hello World!" << endl; return 0; }</pre>	<p>Our program just prints the string <code>"Hello World!"</code> to the screen</p> <p>At the end of the main function we <code>return</code> the value <code>0</code> – success – to the operating system (i.e., Windows)</p>

Why do we need to type `endl`? Why not just use `"\n"`?

`endl` appends `'\n'` to the stream and calls `flush()` on the stream. So

```
cout << x << endl;
```

is equivalent to

```
cout << x << '\n';
cout.flush();
```

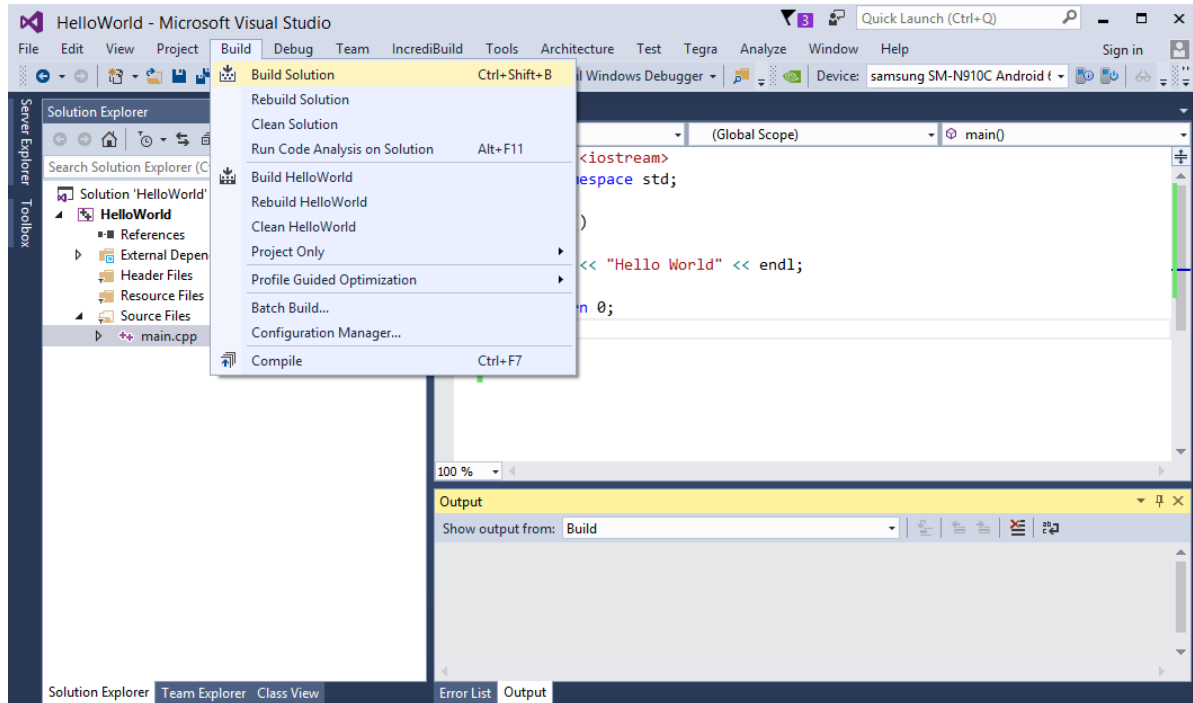
Reference:

<https://stackoverflow.com/questions/7324843/why-use-endl-when-i-can-use-a-newline-character>

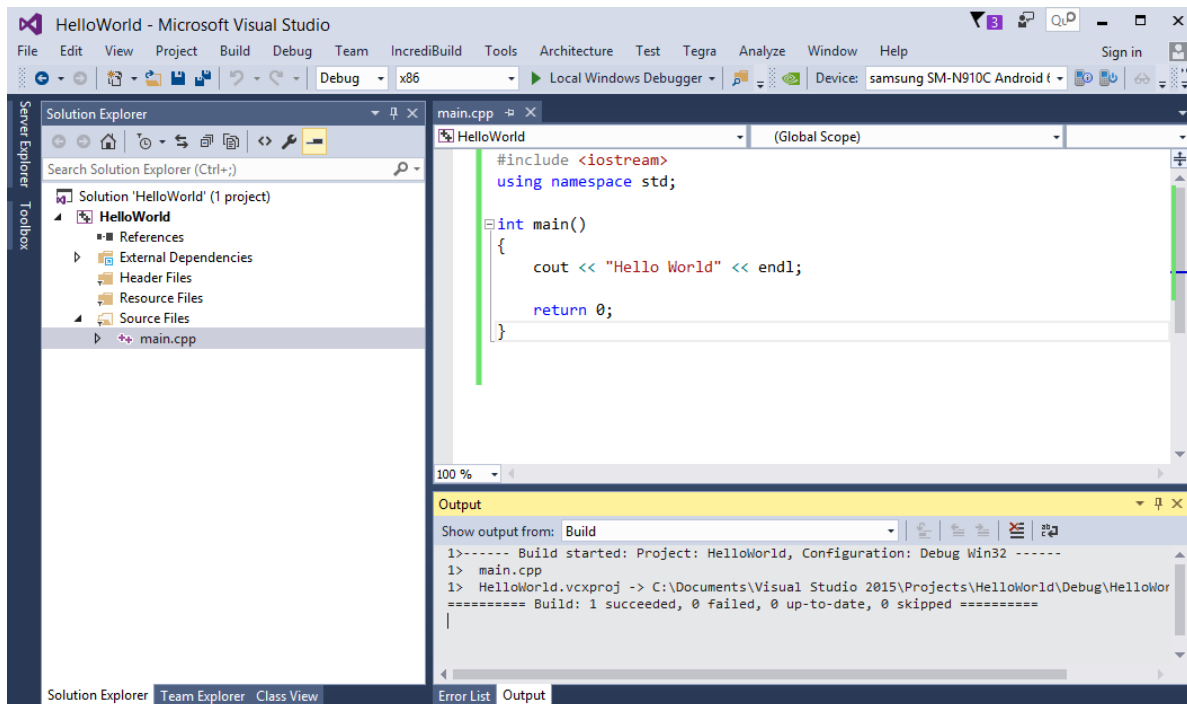
Also, did you know you can create your own namespace?

<https://www.geeksforgeeks.org/namespace-in-c/>

To build the program, either press **F6** or select the menu item **Build/Build Solution**



If everything goes well, the output should look like this:



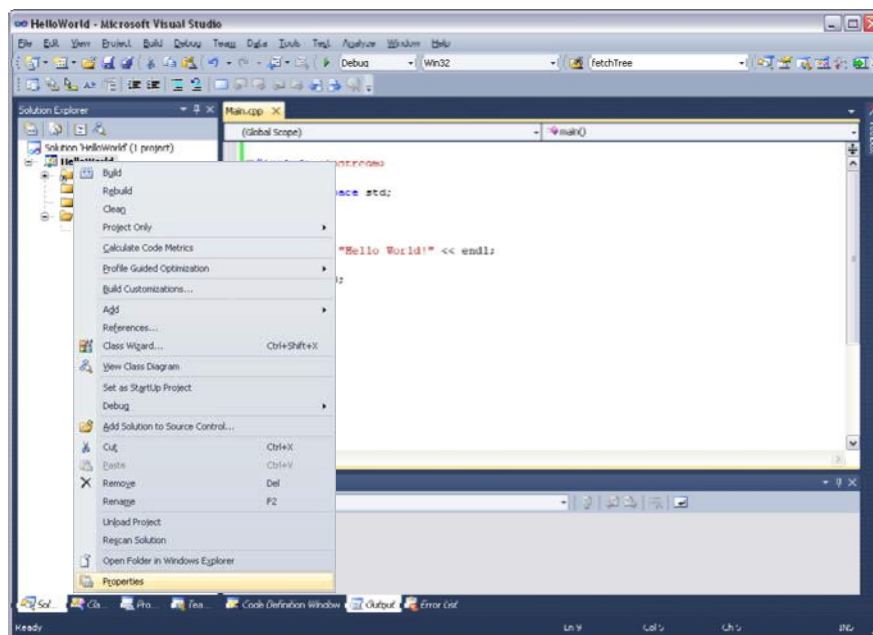
If there are any errors, check the source code, try to correct them, and rebuild your project until the build process succeeds.

The Incremental Linking Issue

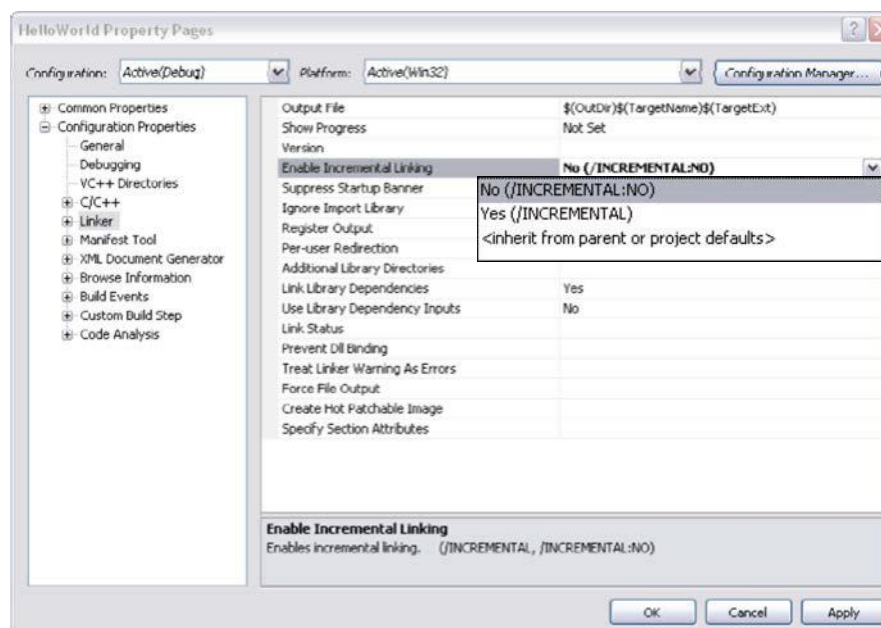
You may encounter

LINK : warning LNK4075: ignoring /INCREMENTAL due to /option specification

This happens on systems that have both Visual Studio 2010 and Visual Studio 2012(13) installed (You may not have this problem if only one version is installed). This issue can prevent you from building your program. There is, however, an easy solution. **You need to turn incremental linking off.**

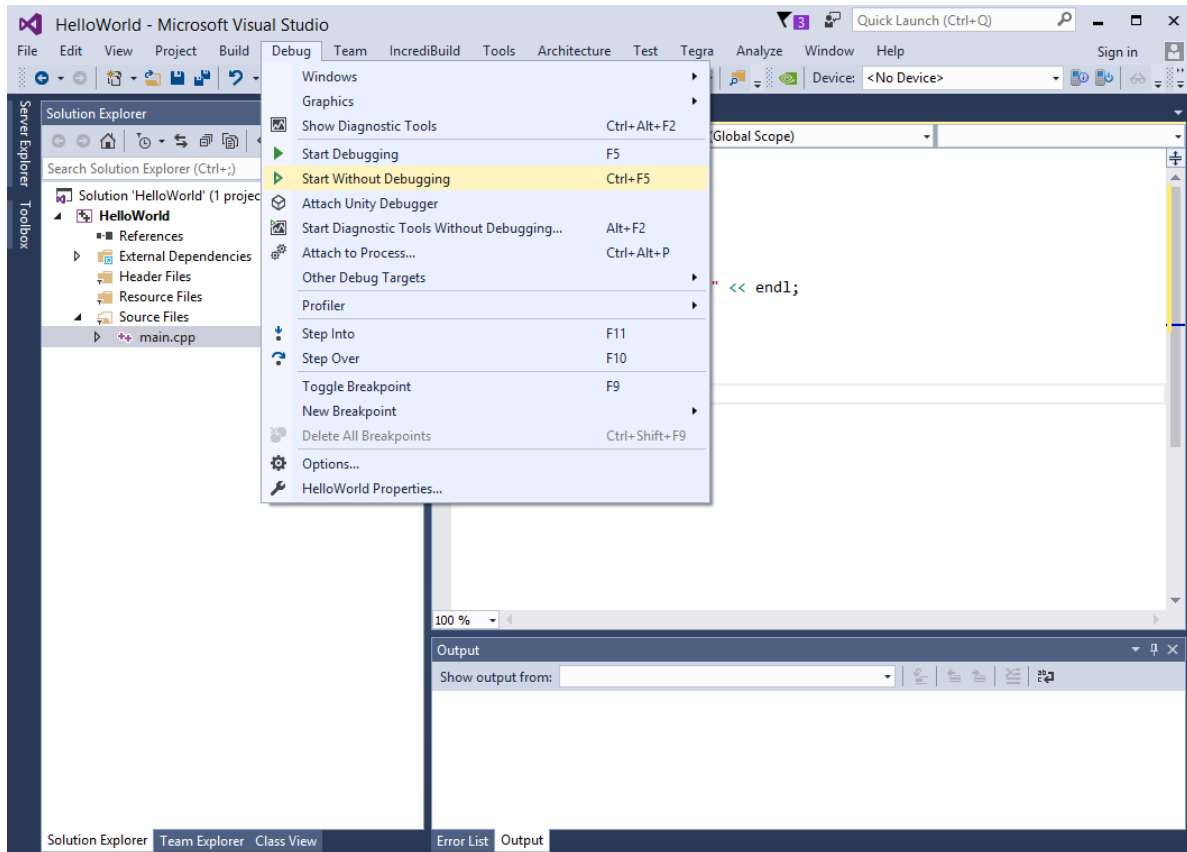


Select Properties/Linker and set Enable Incremental Linking to No:



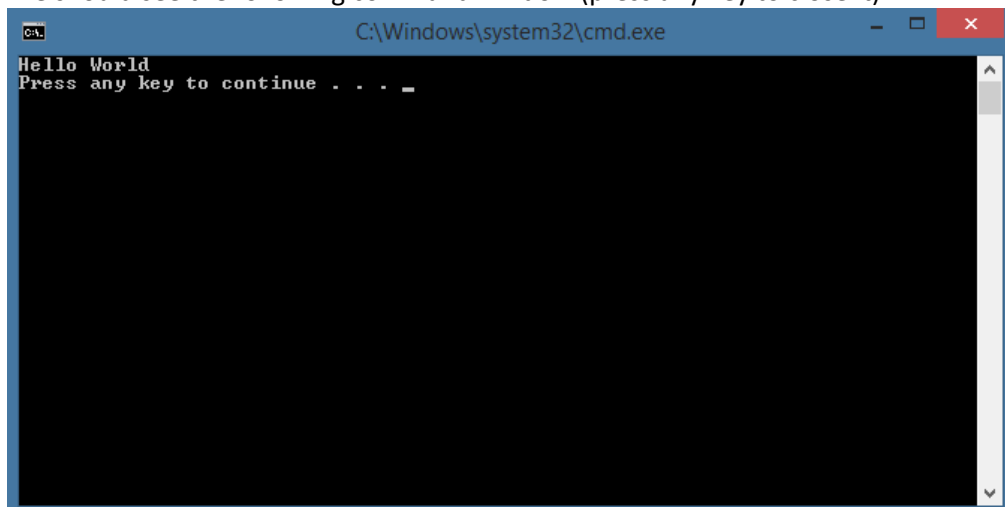
Fourth Step: Running the program.

To run the program press **CTRL+F5** or select **Debug/Start Without Debugging** from the menu:



You might see a popup window stating that the project is out of date and asking if you would like to build it. Click Yes to continue.

We should see the following command window (press any key to close it).



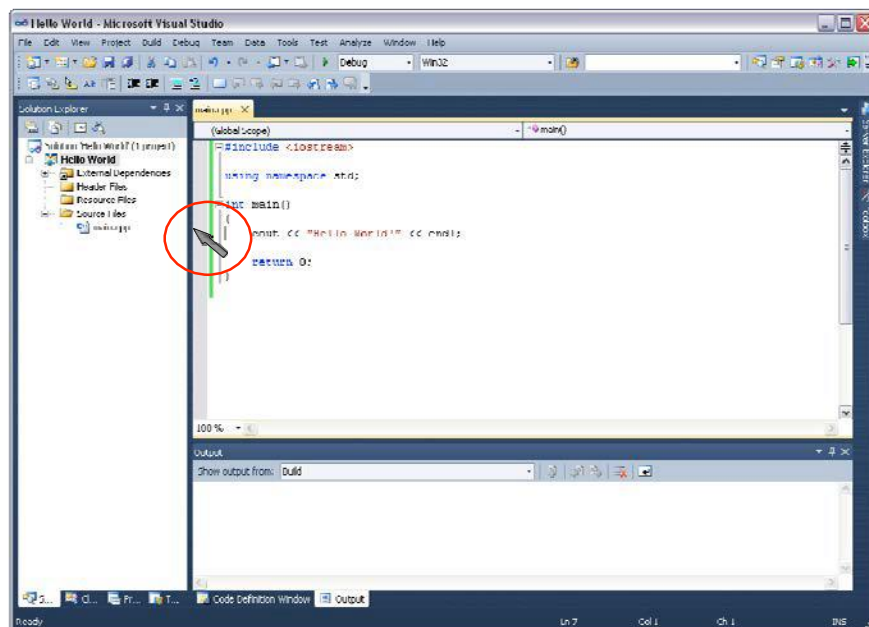
You have successfully built your first C++ program with Visual Studio.

Fifth Step: Debugging.

Now let's check out the debugging feature. To locate any bugs in our program, we need to test it. The simplest form of testing is debugging, a step-wise procedure to monitor how the program evolves. Visual Studio is equipped with a built-in debugger. You can activate a debugging session with **F5** or **Debug/Start Debugging**.

Try it. You will notice that the program starts, a command window appears briefly, and then disappears immediately without giving you a chance to see the output. This is normal. The debugger executes the program normally and stops only at enabled breakpoints. Since we have not specified one, the debugger will not stop anywhere and just run the program as usual.

To specify a breakpoint, we need to position the mouse pointer on the gray area left of the editor window:



Did you know that you can comment the codes out so that that line of code will not be run?

This is good for documentation and testing your code so you can narrow down where the problem is or if you want to edit an existing code but want to keep an original copy.

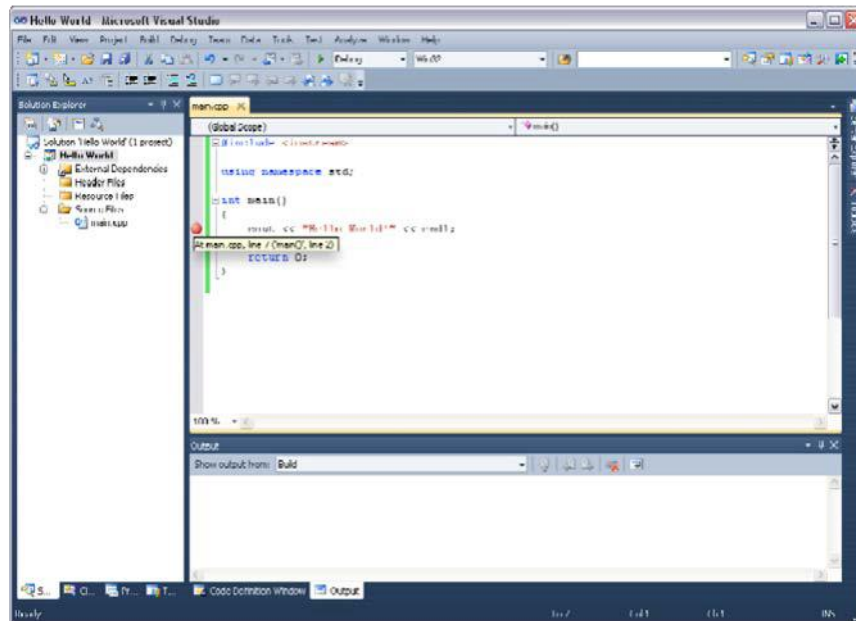
The simplest comment type is by line adding `//` to the beginning.

The other way is the block comment. Using `/* ... */` Anything in between `/*` and `*/` will be commented. For example:

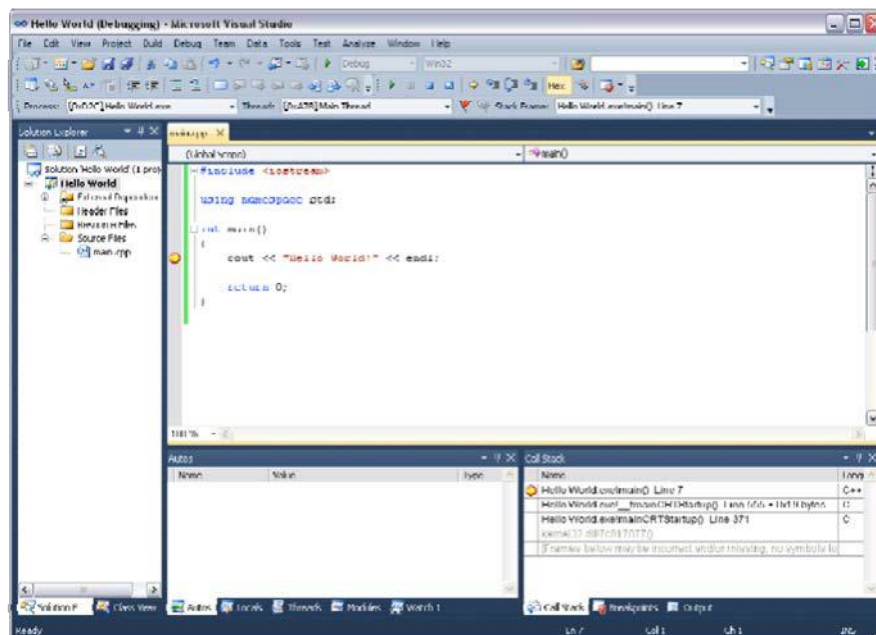
```
//single line comment
/*
//block comment
int main()
{
    cout << "Hello World" << endl;

    return 0;
}*/
```


To activate a breakpoint at the start of our program, click (left mouse button) on the area next to the first line in the main program:



Start a new debug session (either press F5 or Choose Debug > Start Debugging from the menu above) and you will see that the debugger stops at this line:



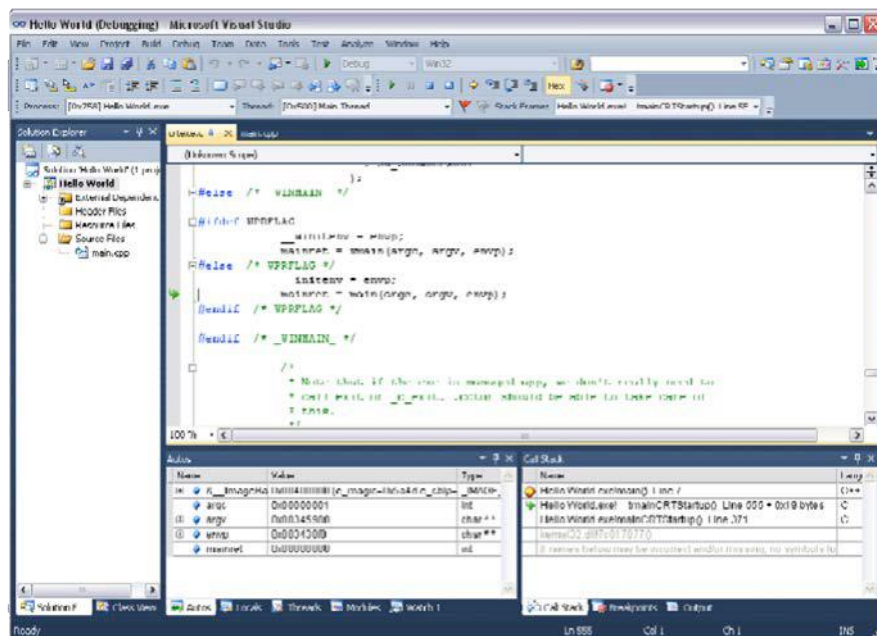
You have four options now:

- Continue (F5) – the debugger will resume the program until it reaches the next active breakpoint or terminates the program normally.
- Step Into (F11) – the debugger steps into the next function to be called and stops again. You can use this facility to inspect functions that you have written.

- Step Over (**F10**) – the debugger will execute the current line and stop again. The facility is useful, when you want to monitor the progress of your program in a step-wise fashion.
- Step Out (**Shift+F11**) – the debugger will resume the current function and stop at the line where the function has been called. This facility allows you to return to the so-called surrounding scope of a function. The surrounding scope of the function main is the C++ runtime environment.

Play with all options.

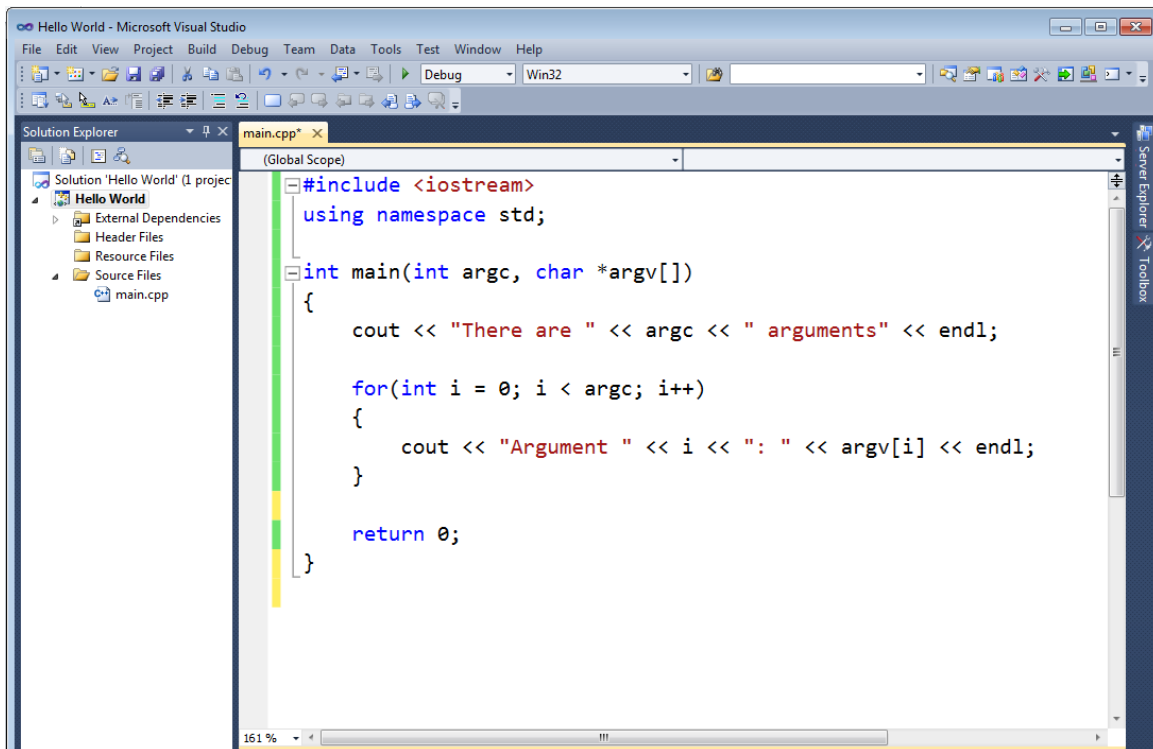
When you debug your program we can access and alter values of variables (to temporarily fix problems) and also switch between the different surrounding scopes using the call stack. For example, you can inspect the C++ runtime environment of main by clicking on the second entry in the call stack window. Try it.



The debugger will show you the line where main has been called and will display the values of the activate variables in the C++ runtime environment. We are not going to change anything here. It is just a demonstration of the abilities of the Visual Studio debugger. Later in the semester, we can use this feature to explore our own programs in more detail, if necessary.

Sixth Step: Working with command line arguments.

We change our main function to print the command line arguments to the screen. For this reason, we add `int argc` and `char * argv[]` as arguments to `main`. By convention, `argc` contains the number of arguments, that is, the number of elements in the array `argv`. The parameter `argv`, on the other hand, contains an array of **C-Strings** (i.e., char arrays terminated with `'\0'`). The element at index 0 is always the name of the command. We use a simple **for-loop** to iterate over the `argv`-array and print each element in turn.



```
#include <iostream>
using namespace std;

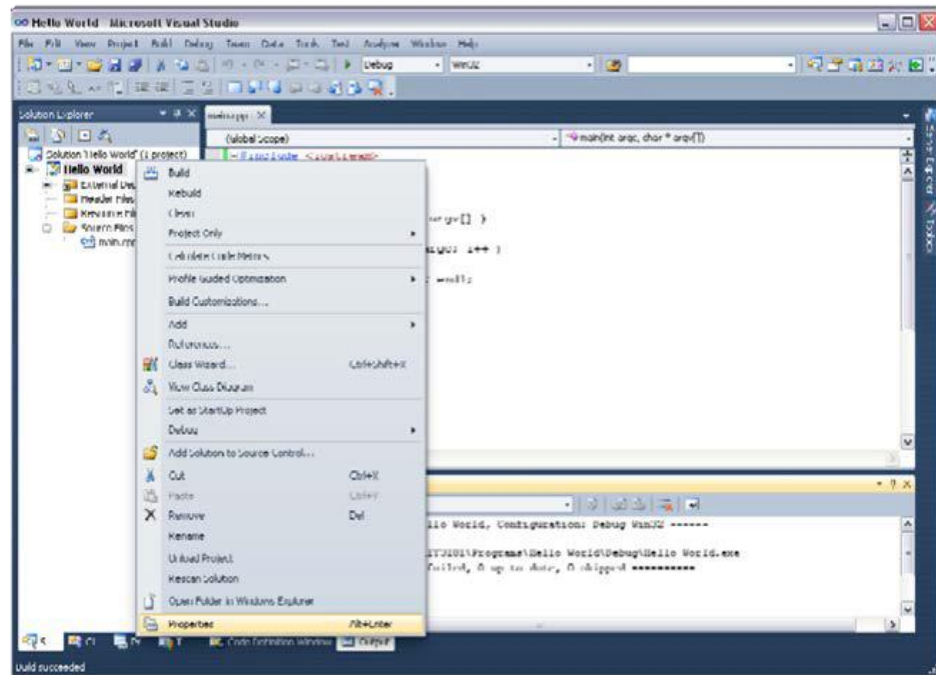
int main(int argc, char *argv[])
{
    cout << "There are " << argc << " arguments" << endl;

    for(int i = 0; i < argc; i++)
    {
        cout << "Argument " << i << ": " << argv[i] << endl;
    }

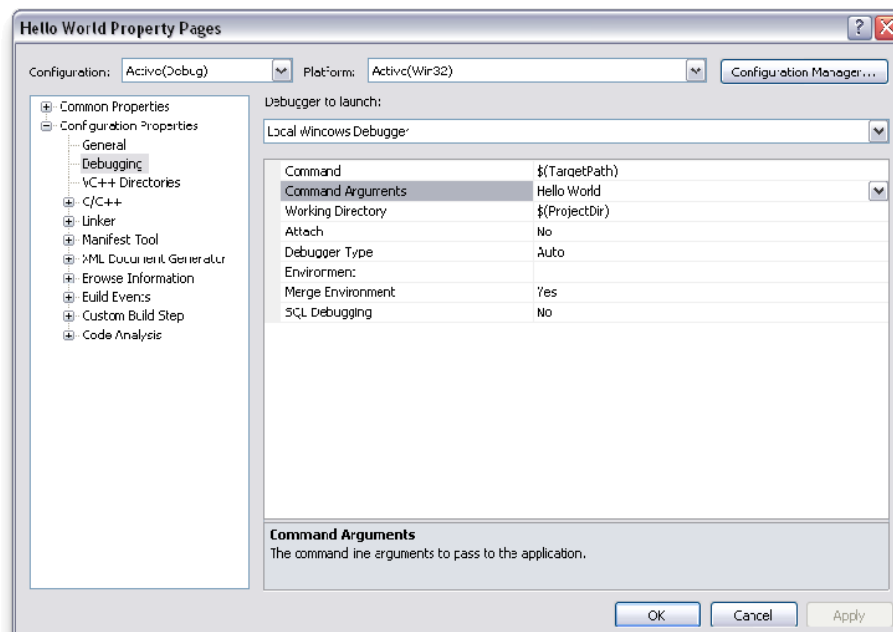
    return 0;
}
```

The screenshot shows the Microsoft Visual Studio IDE. The Solution Explorer on the left displays a project named 'Hello World' with a source file 'main.cpp'. The main.cpp file is open in the editor, showing the following C++ code. The code includes the `<iostream>` header, uses the `std` namespace, and defines a `main` function that takes `argc` and `argv` as arguments. It prints the total number of arguments and then iterates through the `argv` array to print each argument.

We need to tell Visual Studio that we would like to attach some [Command Arguments](#). In our example we write [Hello World](#), that is, we define two arguments [Hello](#) and [World](#). Application-specific setting can be applied in the [Property Pages](#) of the current project. You need to right-click on your project [Hello World](#) in the [Solution Explorer](#) and select [Properties](#):

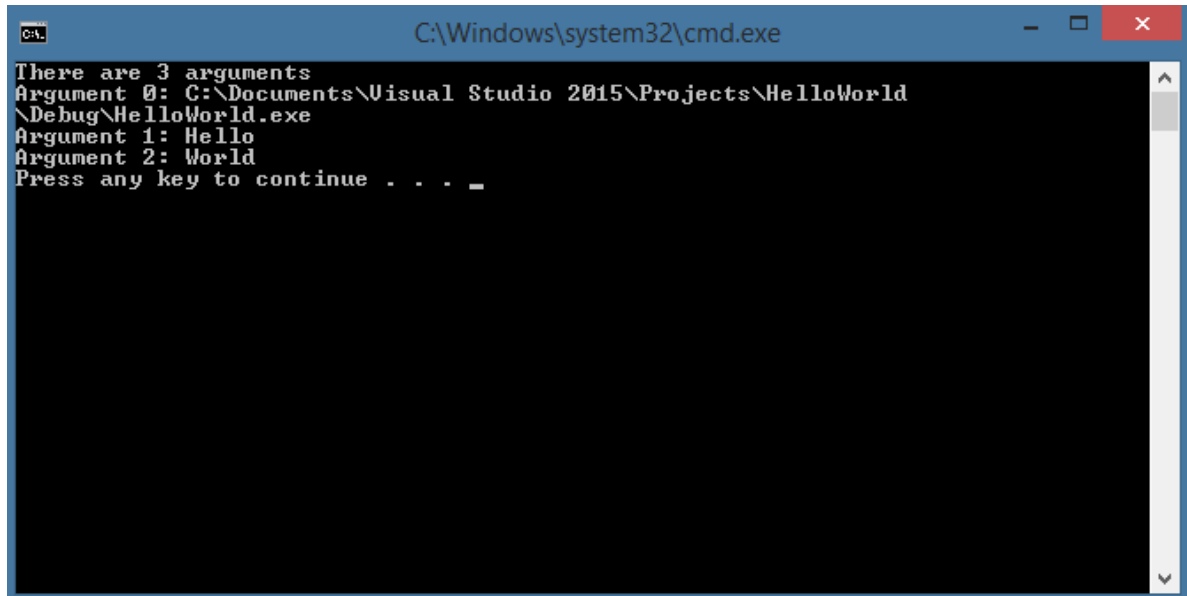


This brings up the following dialog window:



You need to select [Configuration Properties/Debugging](#) in the left pane. In the right pane you can now specify the desired [Command Arguments](#) (edit value).

Press [OK](#) to accept the settings and test the program using [Start Without Debugging](#). You may need to build it again.

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The window contains the following text:

```
There are 3 arguments
Argument 0: C:\Documents\Visual Studio 2015\Projects\HelloWorld
\Debug\HelloWorld.exe
Argument 1: Hello
Argument 2: World
Press any key to continue . . . _
```

In our example, the output should consist of four lines:

- 1) The total number of arguments.
- 2) The first argument, "Argument 0", which will be the full path of the program.
- 3) The line showing Argument 1
- 4) The line showing Argument 2

You have successfully completed this tutorial.