

Swinburne University Of Technology*Faculty of Science, Engineering and Technology***LABORATORY COVER SHEET**

Subject Code:	COS30008
Subject Title:	Data Structures and Patterns
Lab number and title:	1, Using Visual C++
Lecturer:	Dr. Markus Lumpe

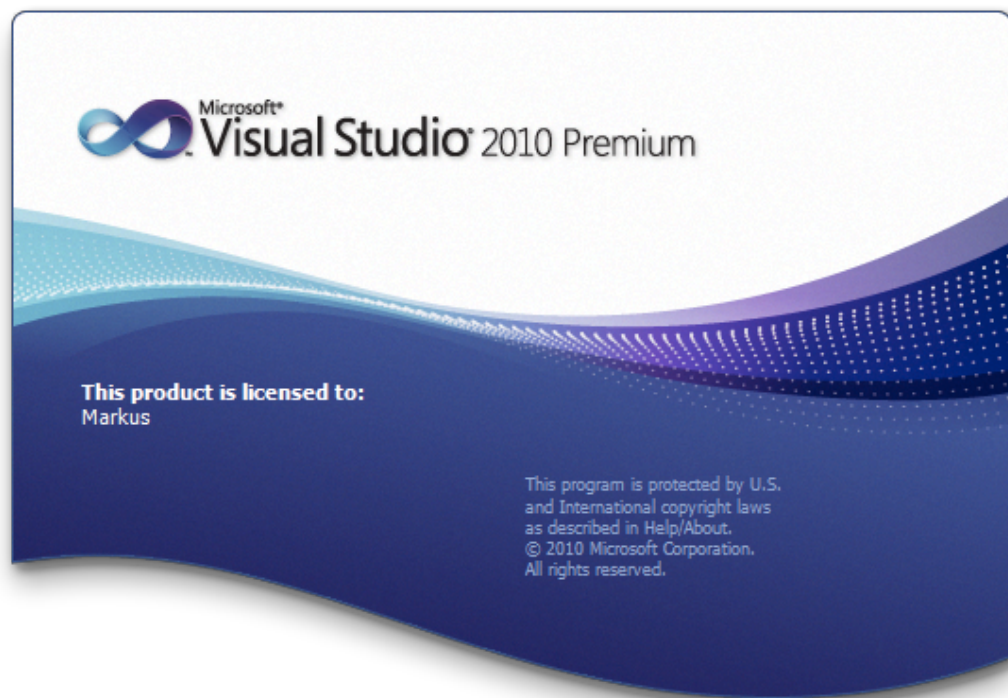


Figure 1: Visual Studio 2010 Startup Window.

Where to get Visual Studio 2010

Every student enrolled in a unit of the Faculty of Science, Engineering and Technology can obtain a valid copy of Visual Studio via the faculty's ELMS Campus Portal:

<https://elms.ict.swin.edu.au>

Use as "Product Search" string "visual studio 2010."

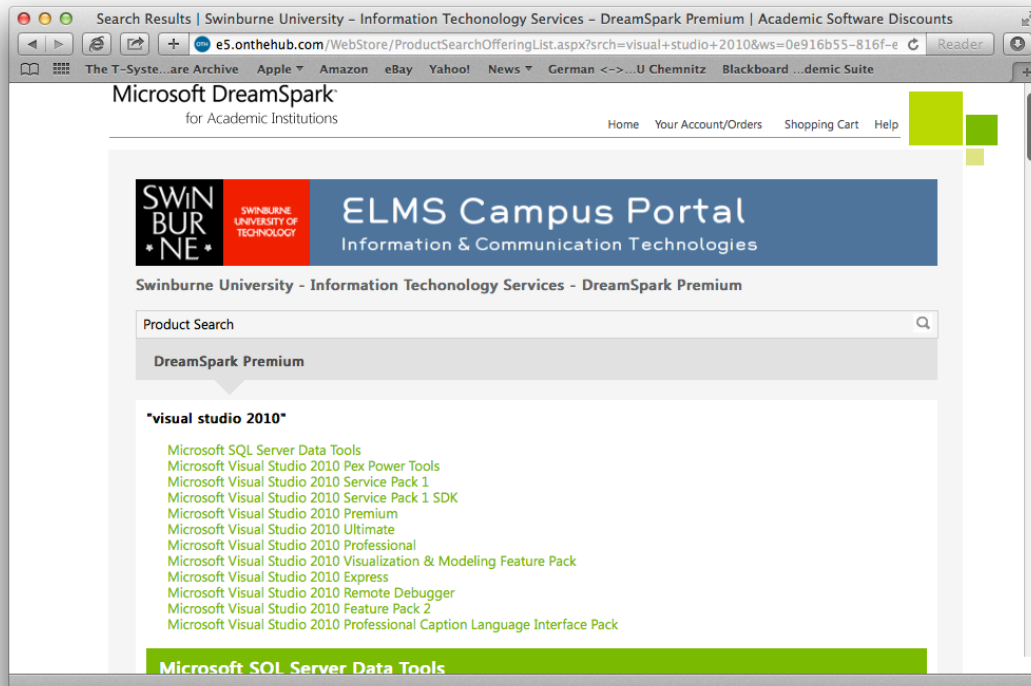


Figure 2: ELMS Portal.

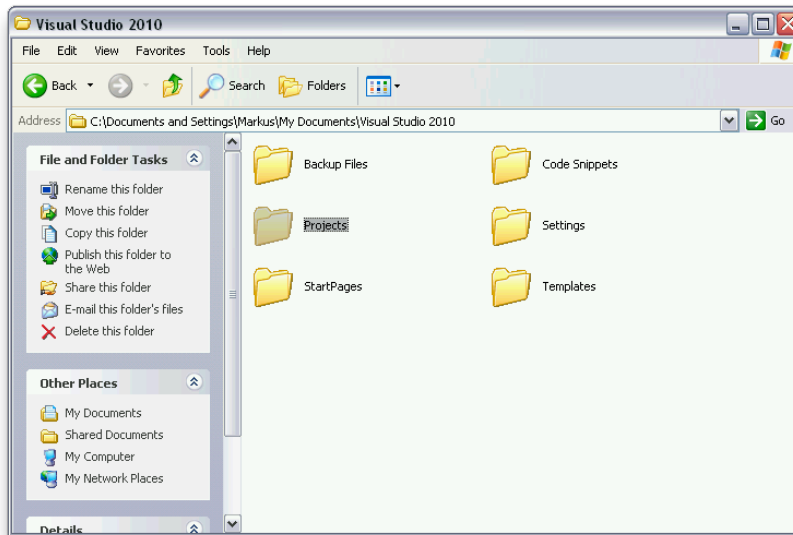
The "Professional" edition provides everything needed in COS30008.

Do not use the "Express" edition! The user interface is different and the variations in the user interface will make debugging more cumbersome.

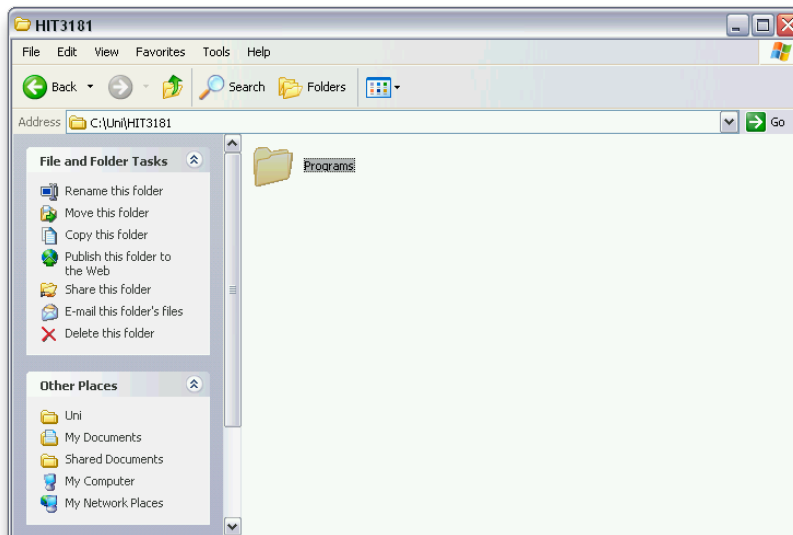
Lab 1: First Steps in C++ - Visual Studio

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 [My Documents\Visual Studio 2010](#):

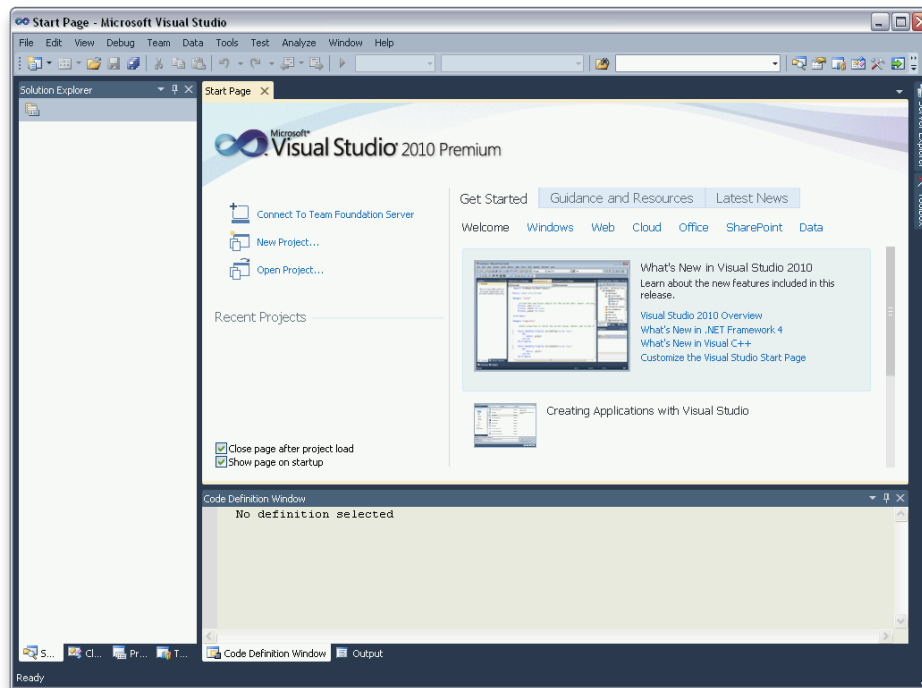


You can use this location, but in a shared environment like the lab computers in ATC625 there is no guarantee that the contents of this folder will be retained between sessions. Also, you may want to separate your projects based on your course work. For example, you may want to assemble all your files (i.e., lecture notes, assignments, and projects) in a semester-specific subject directory. For example, you could use [C:\Uni\COS30008\Programs](#) (or another suitable location, if you do not have write permissions for C:) as your working directory (and throughout this tutorial):

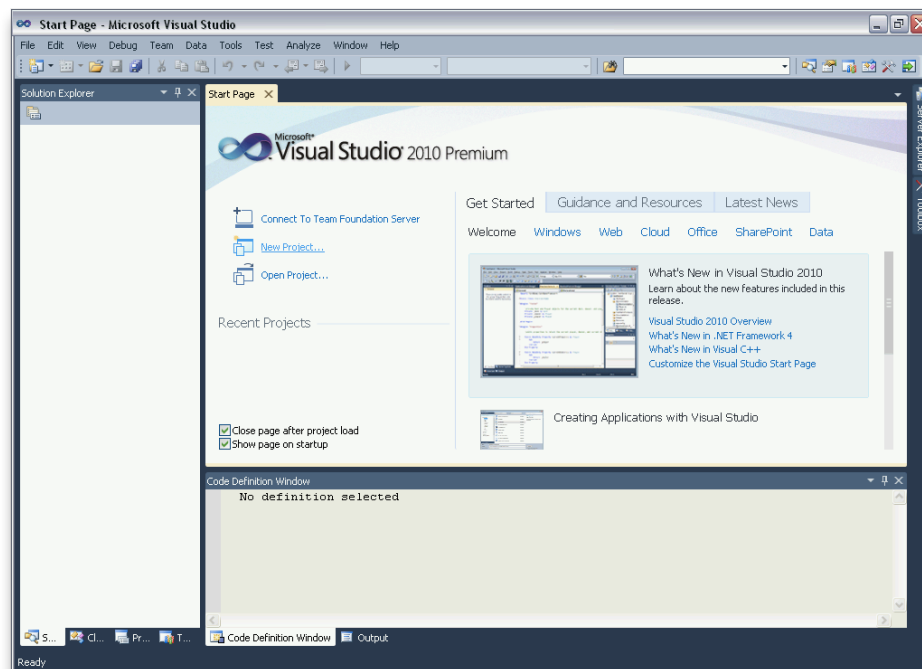


Second Step: Start Visual Studio and Select C++ Template.

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

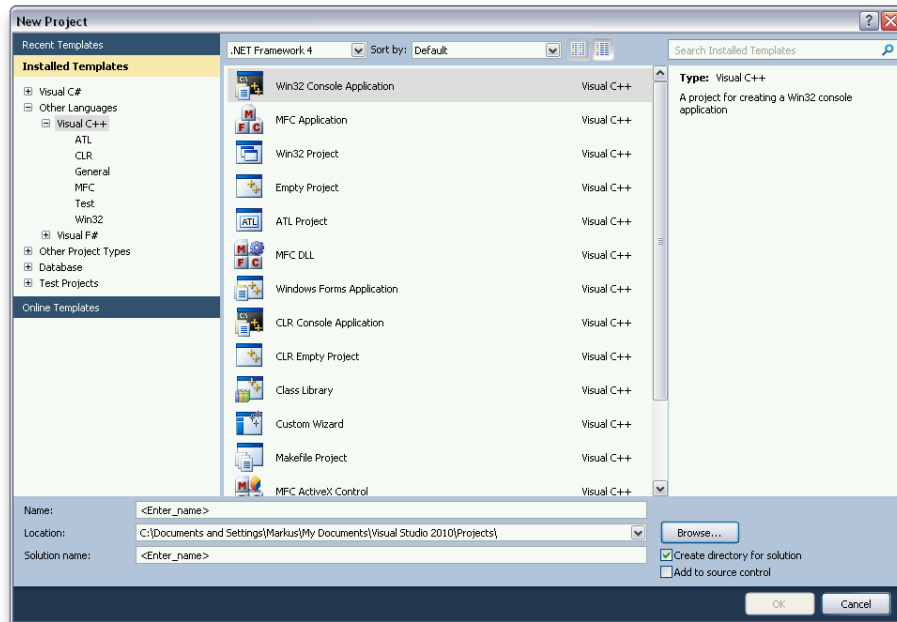


Select **New Project...** now:

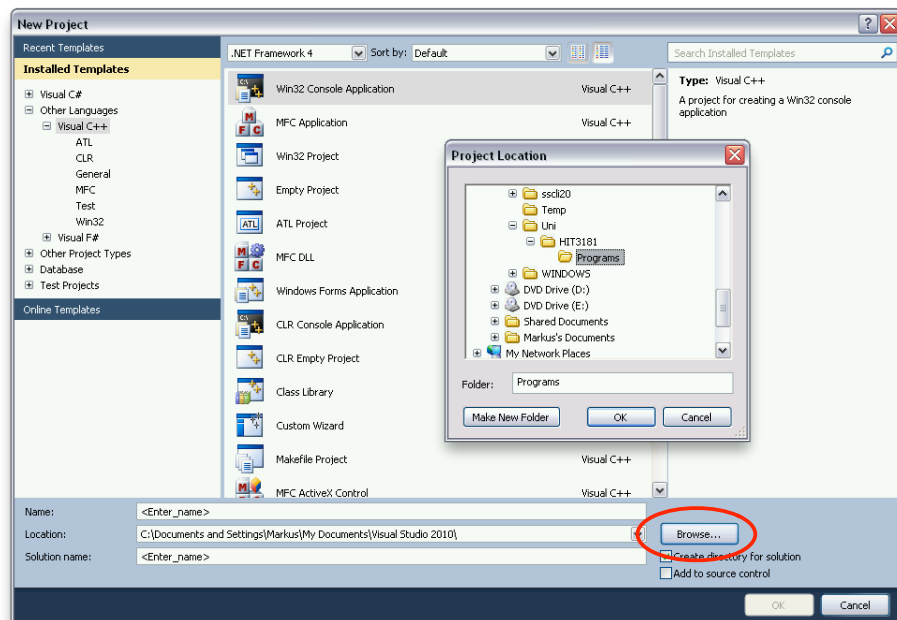


As a result, the New Project dialog window will appear. This is a so-called modal dialog window and it will stay on top of Visual Studio to force you to choose a proper project

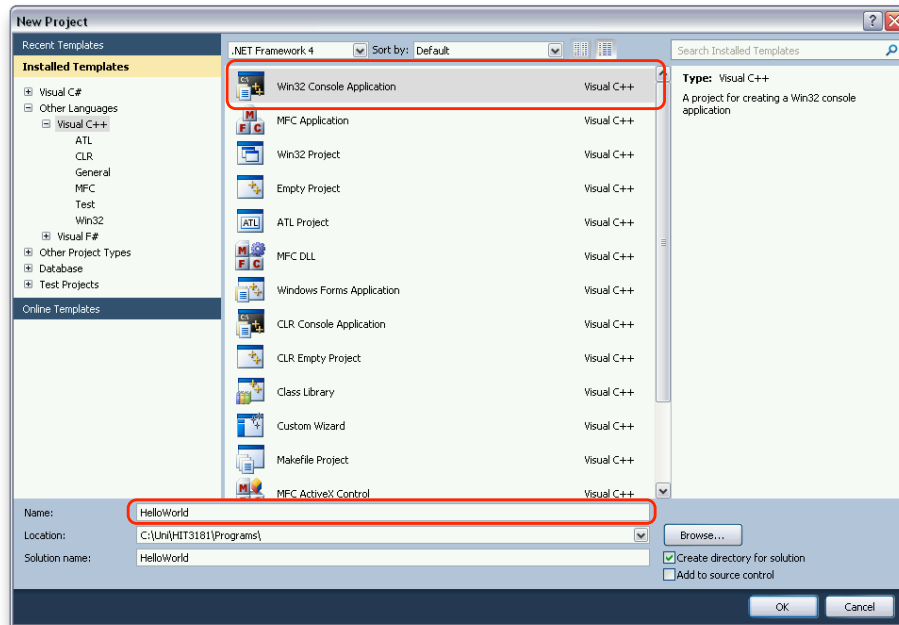
template. That is, you cannot do anything else before you have selected the type of project that you want to work on and you have completed the associated project-specific configuration steps:



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"](#):

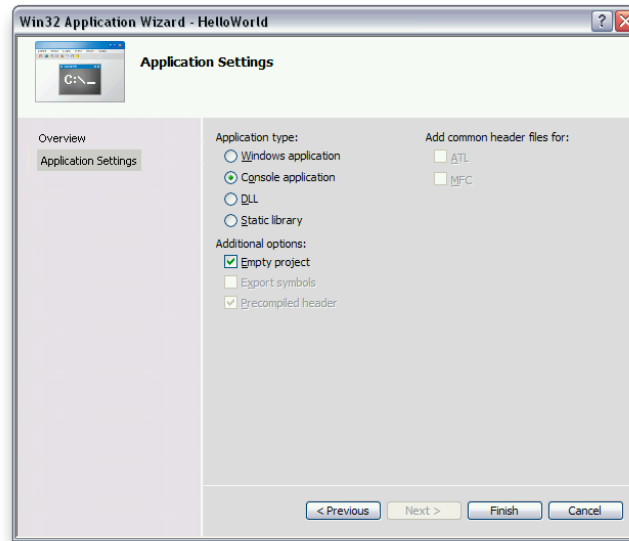


You should notice that the **OK** button is now enabled. That means, the basic settings have been completed. Press OK. Visual Studio will create the project. Yet, there is another step before we can proceed:

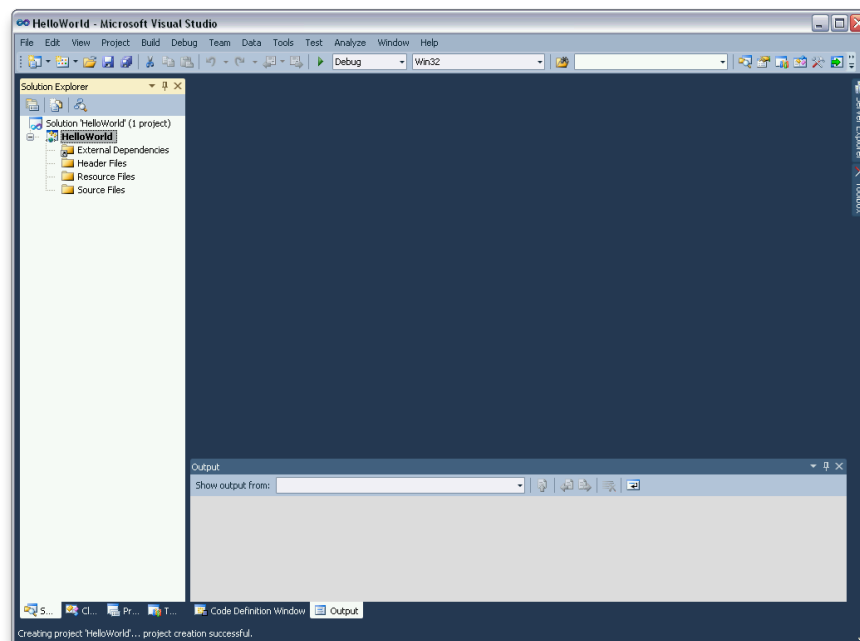


We see the **Win32 Application Wizard**. This wizard allows for fine-tuning the project settings. We could just press the **Finish** button to accept the default, but this would be a bad idea. We should always inspect and control what Visual Studio is assuming about our intentions. Hence, we press the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

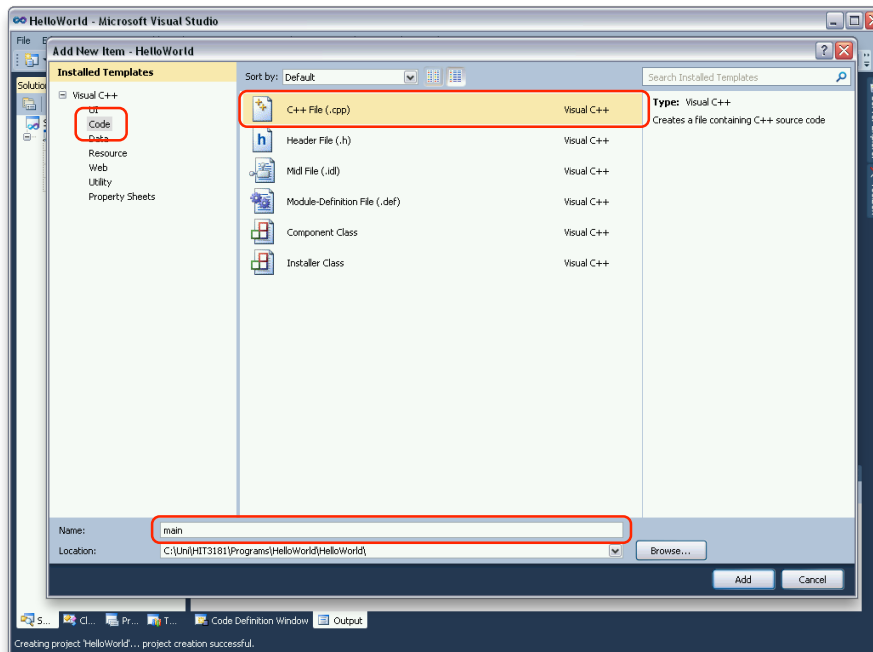
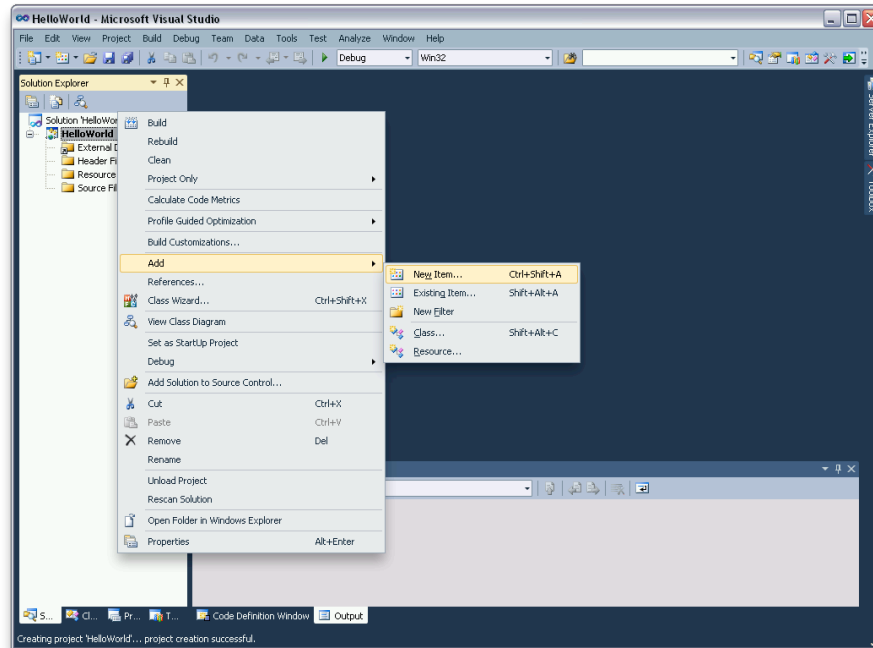
To keep things simple, we select **Empty project**. This way we create an empty project – we are in complete control!



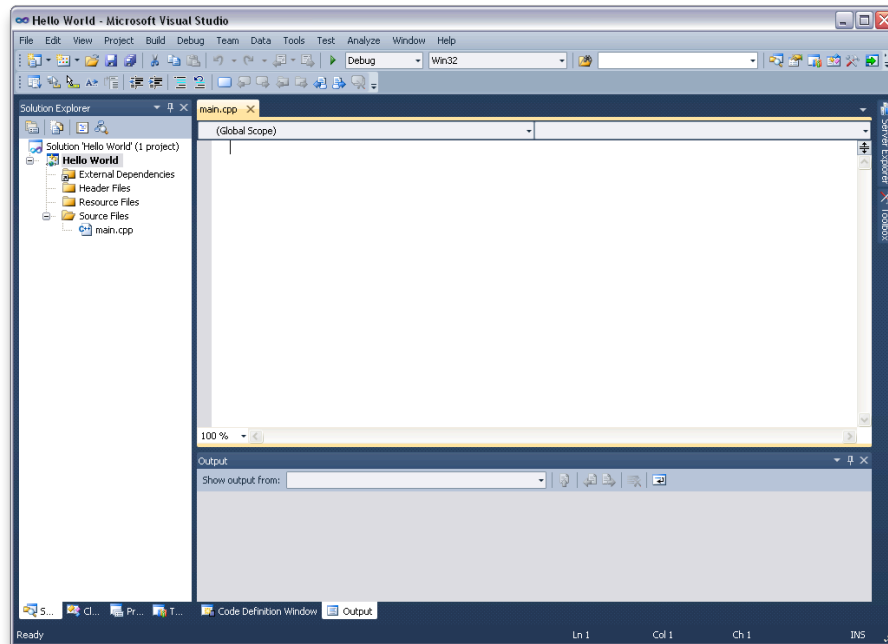
Configuration complete. There is just one option left: press the **Finish** button. The result should look like this:



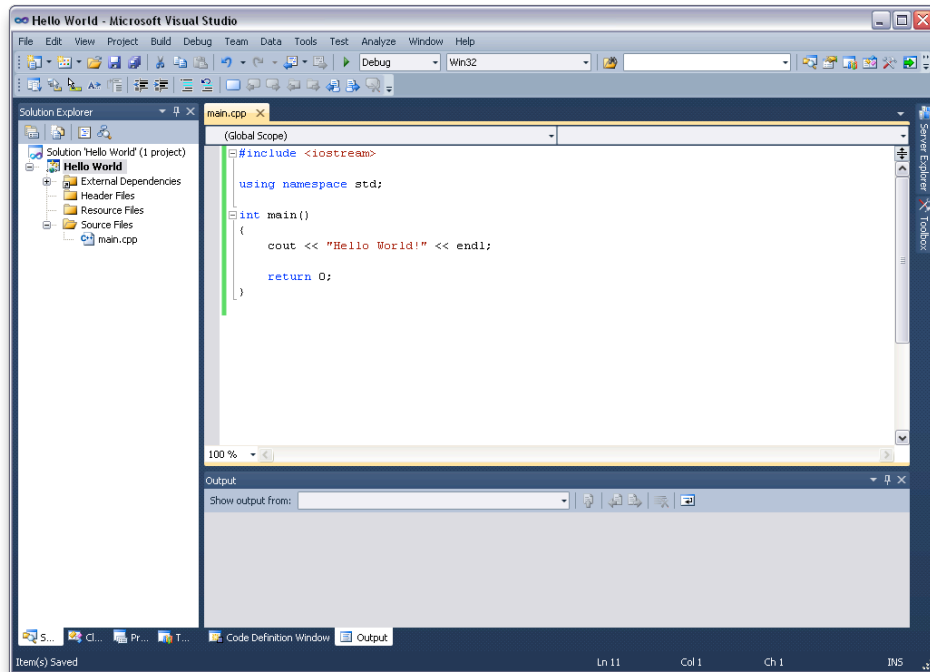
Now we add a **New Item**. We select **Code** of **Visual C++**, select **C++ File (.cpp)**, and type **main** as the name for the file:



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.



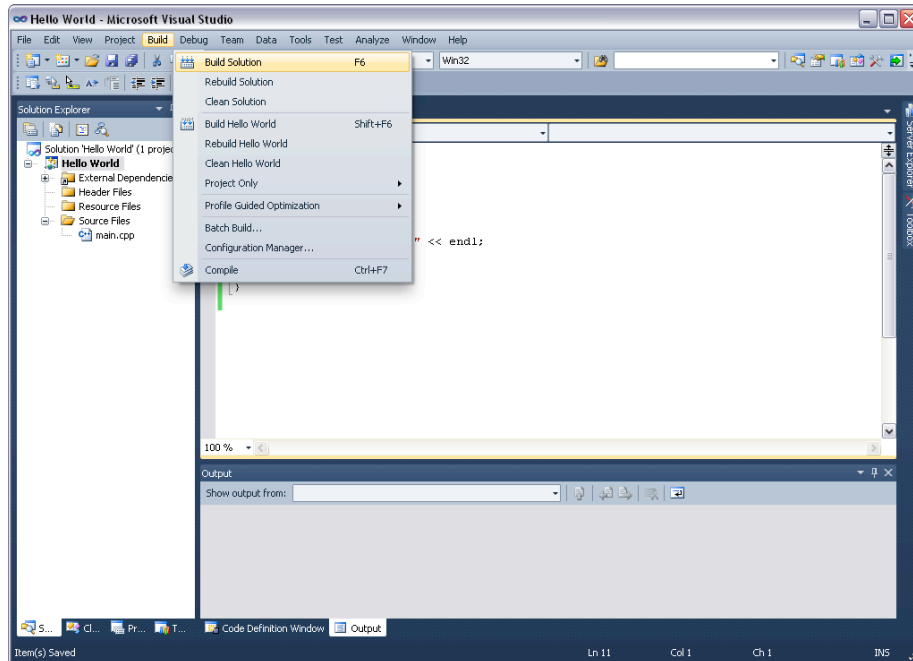
Third Step: Our first program.



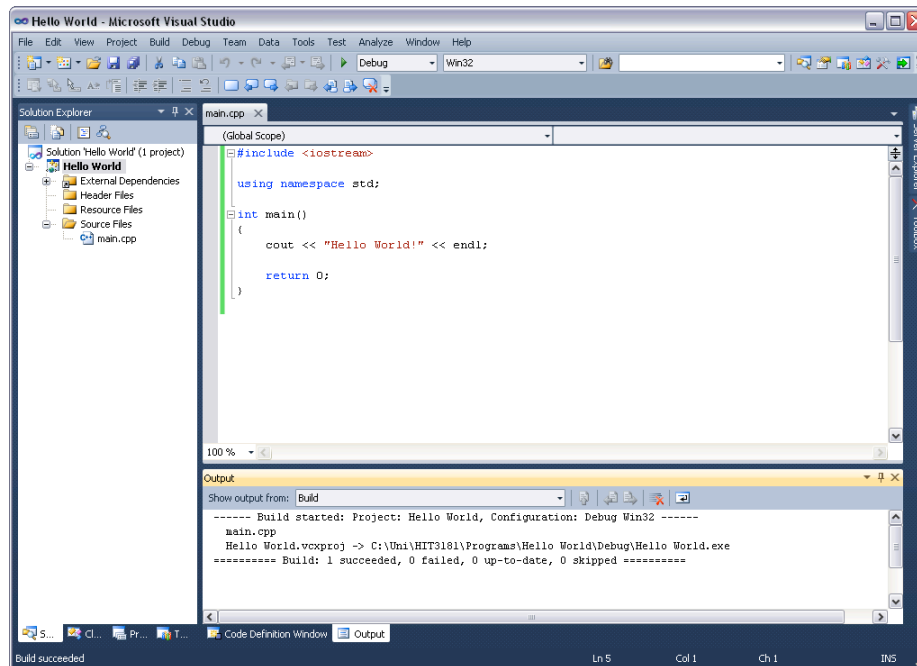
In particular, we write:

<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> .
<pre>int main() { cout << "Hello World!" << endl; return 0; }</pre>	<p>Our program just prints the string "Hello World!" 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>

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



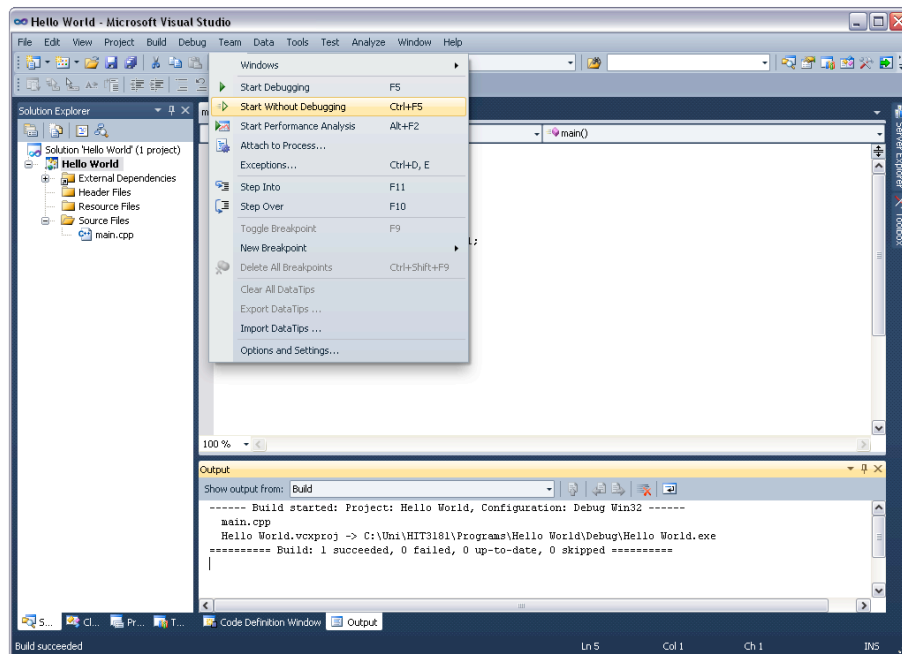
If everything goes well, the output should look like this:



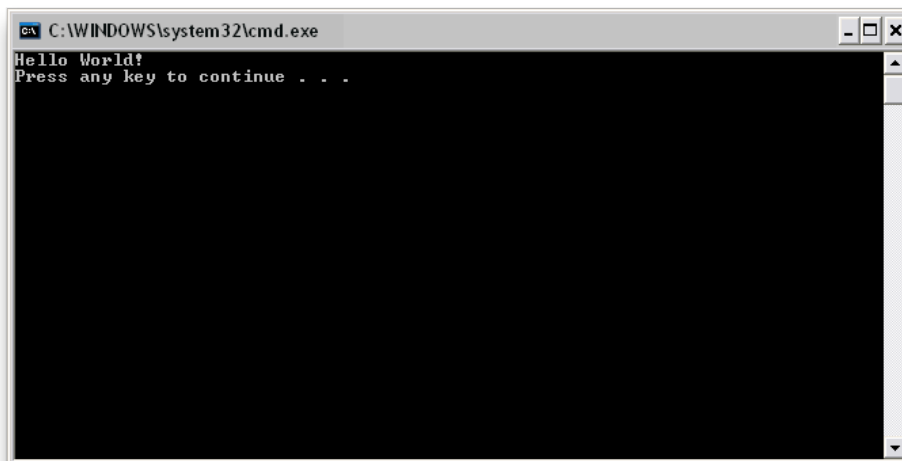
If there are any errors, check the source code, try to correct them, and rebuild your project until the build process succeeds.

Fourth Step: Running the program.

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



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



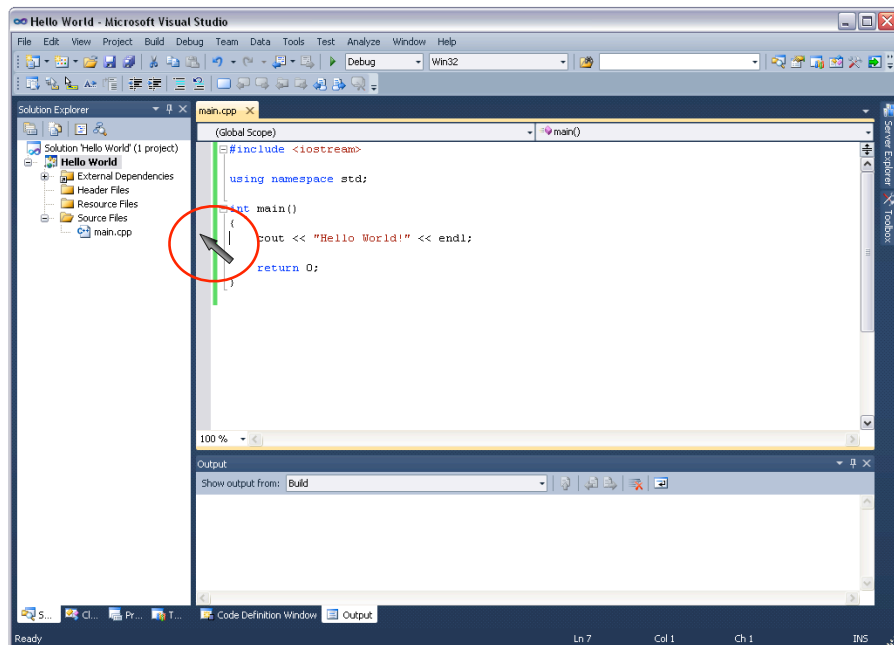
You have successfully built your first C++ program with Visual Studio.

Fifth Step: Debugging.

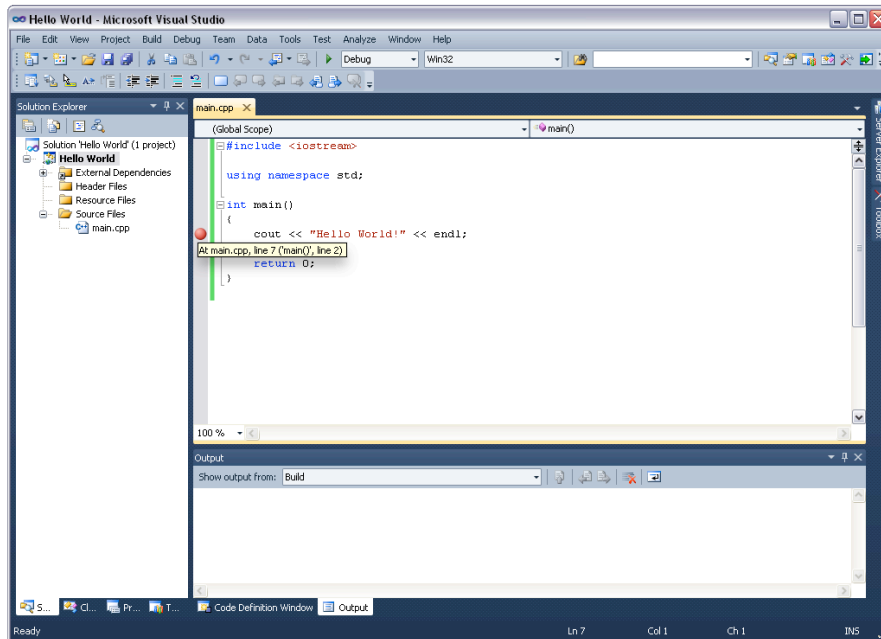
Unfortunately, it happens often that our program contains bugs. To locate them we need to test our program. The simplest form of testing is debugging, a step-wise procedure to monitor how the program evolves. Visual Studio is equipped with a built-in debugger. You can activate a debugging session with **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.

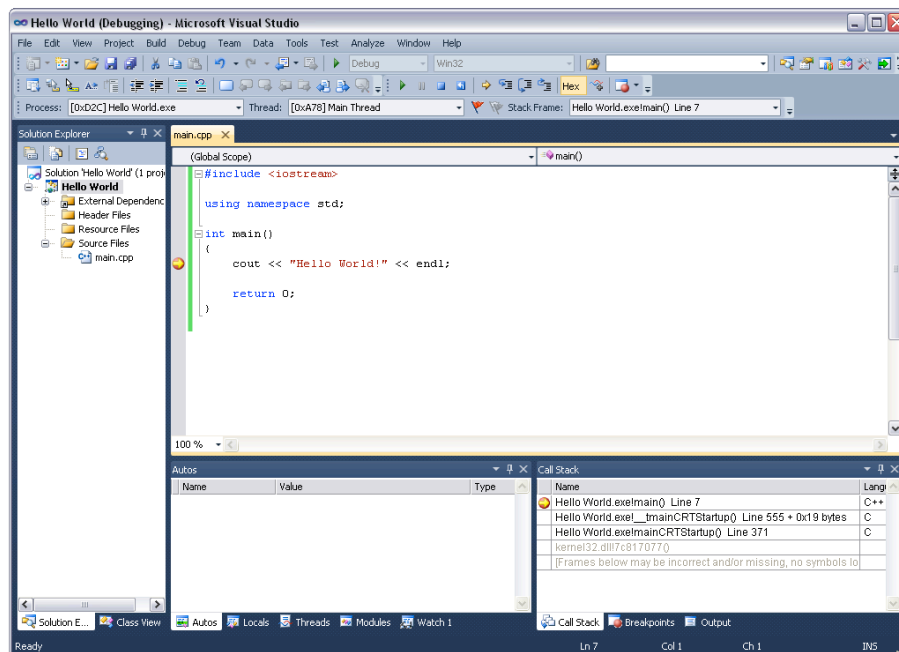
To specify a breakpoint, we need to position the mouse pointer on the gray area left of the editor window:



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 and you will see that the debugger stops at this line:



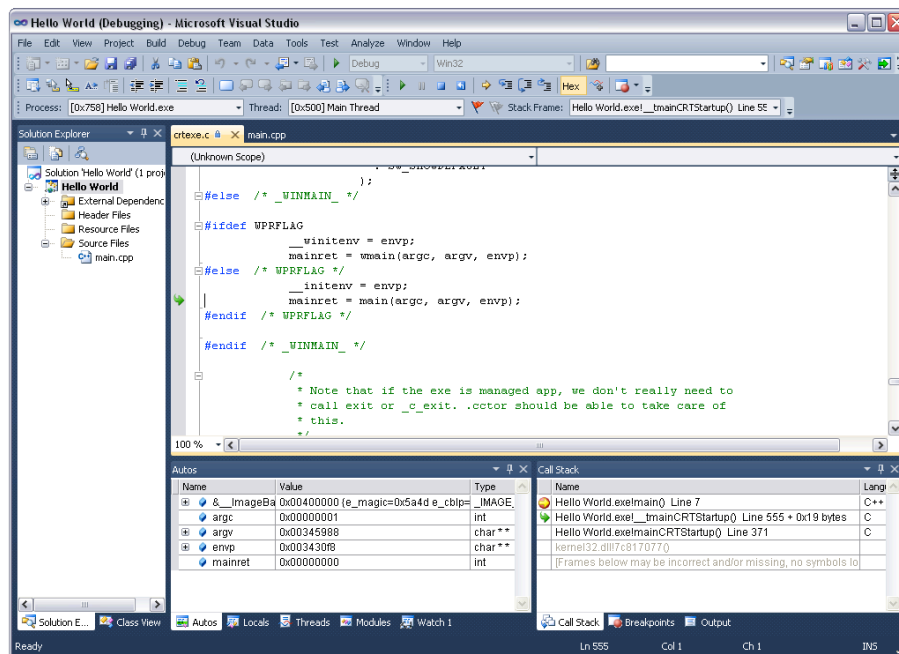
You have four options now:

- 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.

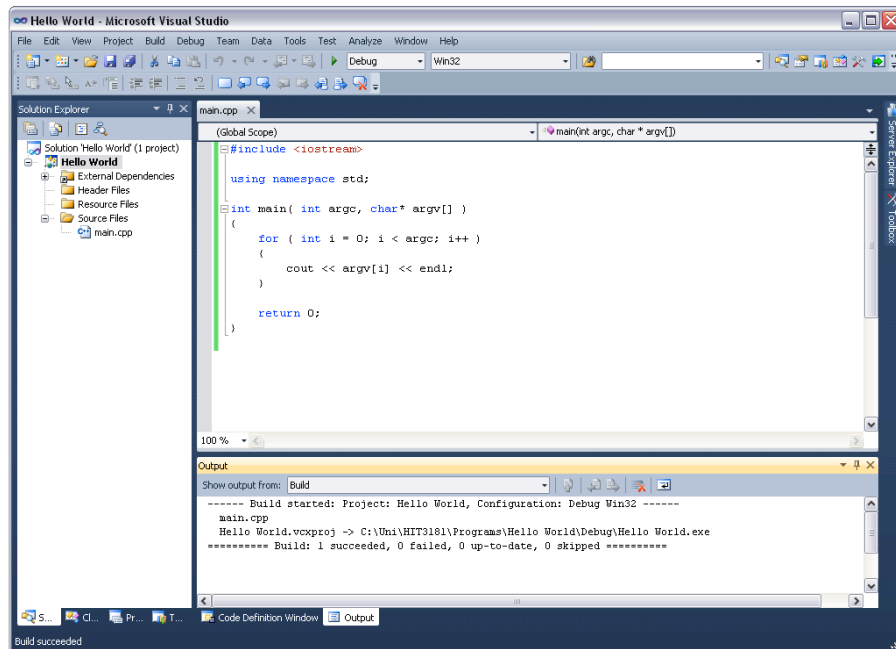
When you debug your program we can access and alter values of variables (to temporarily fix problems) and also switch between the different surrounding scopes using the call stack. For example, you can inspect the C++ runtime environment of main by clicking on the second entry in the call stack window.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



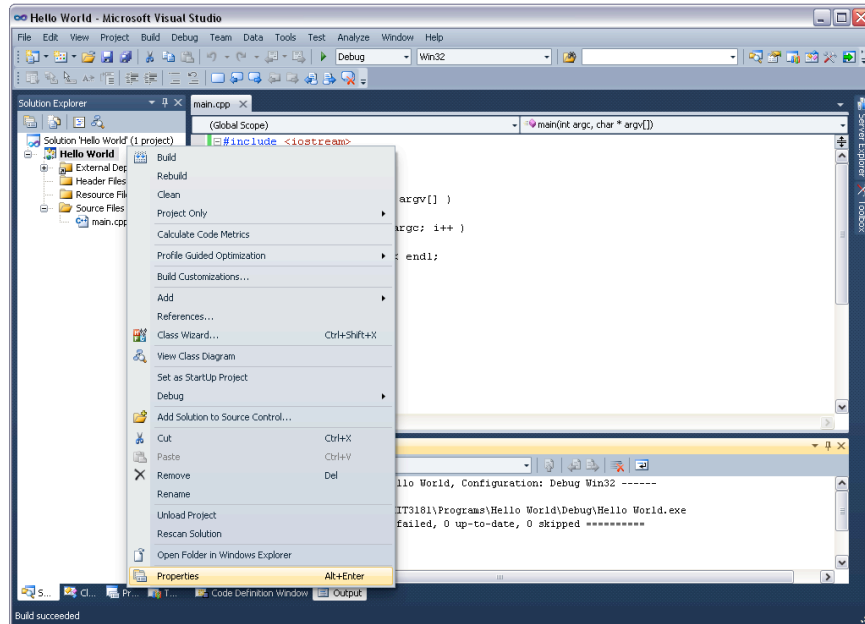
The debugger will show you the line where main has been called and will display the values of the activate variables in the C++ runtime environment. We are not going to change anything here. It is just a demonstration of the abilities of the Visual Studio debugger. Later in the semester, we can use this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

Sixth Step: Working with command line arguments.

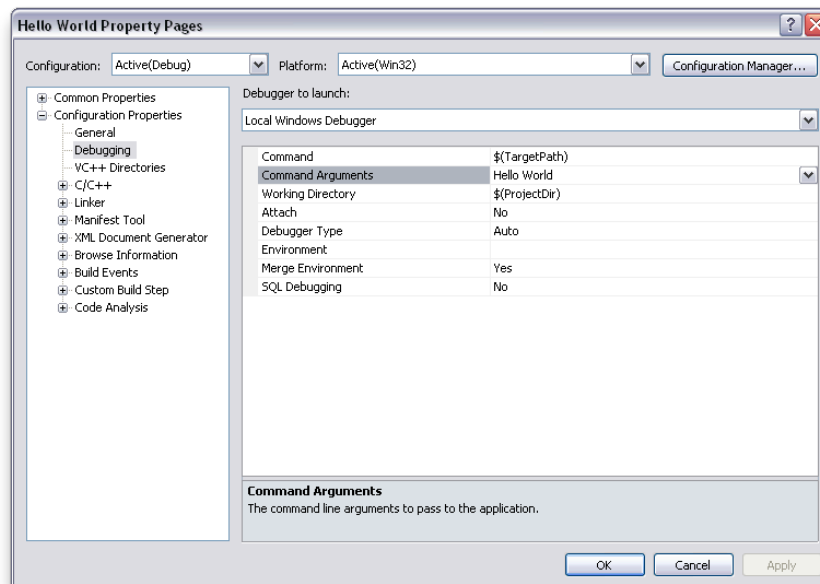
We change our main function to print the command line arguments to the screen. For this reason, we add `int argc` and `char * argv[]` as arguments to `main`. By convention, `argc` contains the number of arguments, that is, the number of elements in the array `argv`. The parameter `argv`, on the other 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.



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](#):

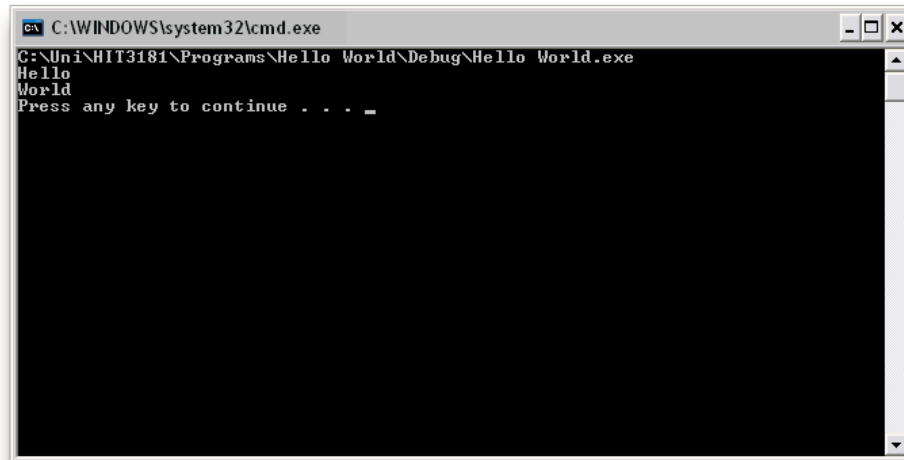


This brings up the following dialog window:



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.



```
C:\WINDOWS\system32\cmd.exe
C:\Uni\HIT3181\Programs\Hello World\Debug\Hello World.exe
Hello
World
Press any key to continue . . . _
```

The output should consist of three lines: the full path of the program, the line "Hello", and the line "World".

You have completed successfully this tutorial session.