CPSC 259 UNIT TESTING

September 2014

Testing and Debugging

- These are crucial skills
- Testing searches for the presence of errors
- Debugging searches for the source of errors
- The manifestation of an error may well occur some ‘distance’ from its source

What is Unit Testing

- **Primary goal**: Isolate the smallest pieces of testable software in an application and determine whether they behave exactly as you expect

  http://msdn.microsoft.com/en-us/library/aa292197(v=vs.71).aspx

- **Motivation**: Good unit tests give you the ability to verify that your functions work as expected and help you to identify failures in your algorithms

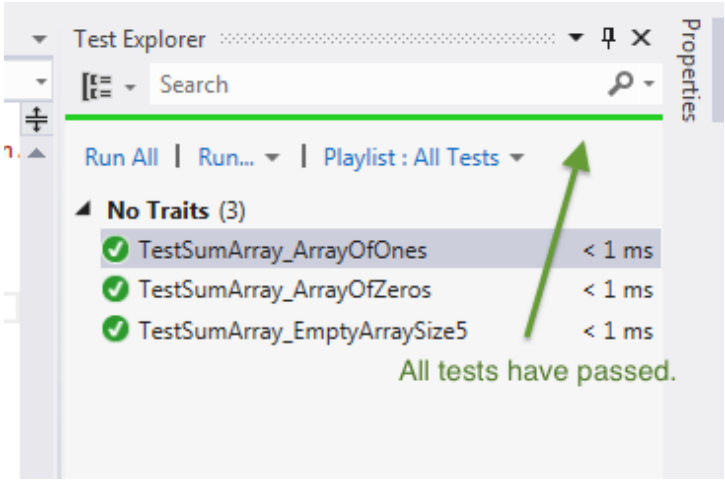
  http://wp.tutsplus.com/tutorials/creative-coding/the-beginners-guide-to-unit-testing-what-is-unit-testing/

- "A unit test is an automated piece of code that invokes a unit of work in the system and then checks a single assumption about the behavior of that unit of work."

  http://artofunittesting.com/definition-of-a-unit-test/

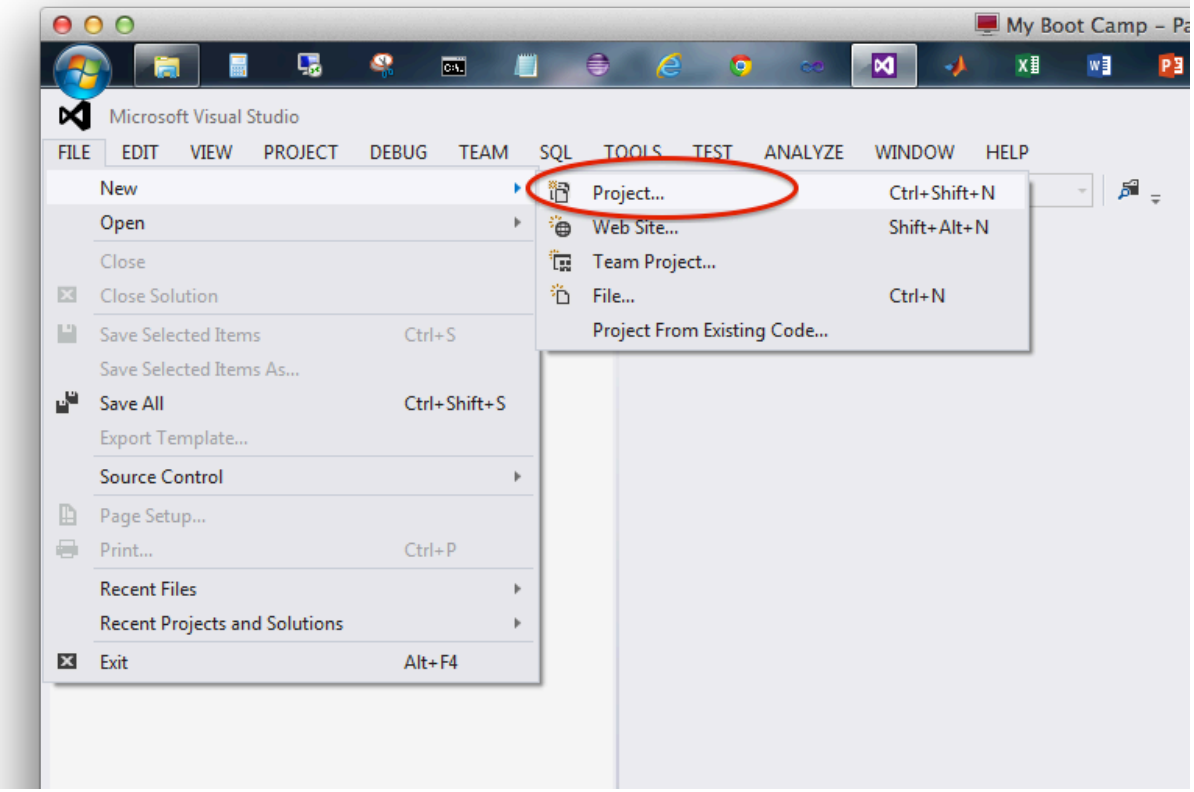
What is Unit Testing in CPSC 259?

- Write test cases for our functions
- Each test case executes a function with a unique assortment of parameters to test the function's correctness
- Write as many test cases as necessary to thoroughly test a function
- Employ Test-Driven Development when we write the tests first



Step-by-step: Unit Tests and VS2012

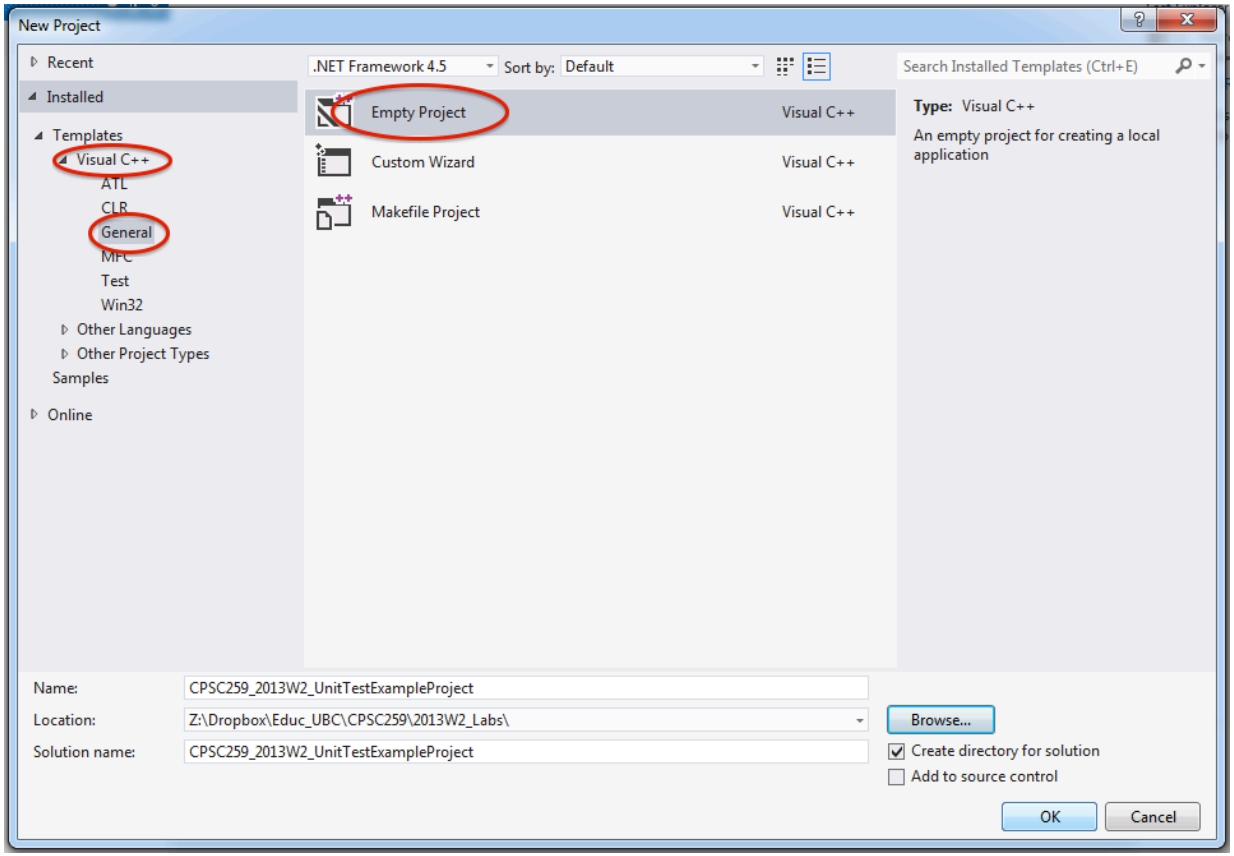
1. Start a new project: **Menu: File -> New -> Project**



Here's one way to create a Visual Studio 2012 Solution that contains a C-language project, and an associated Unit Test project.

Step-by-step: Unit Tests and VS2012

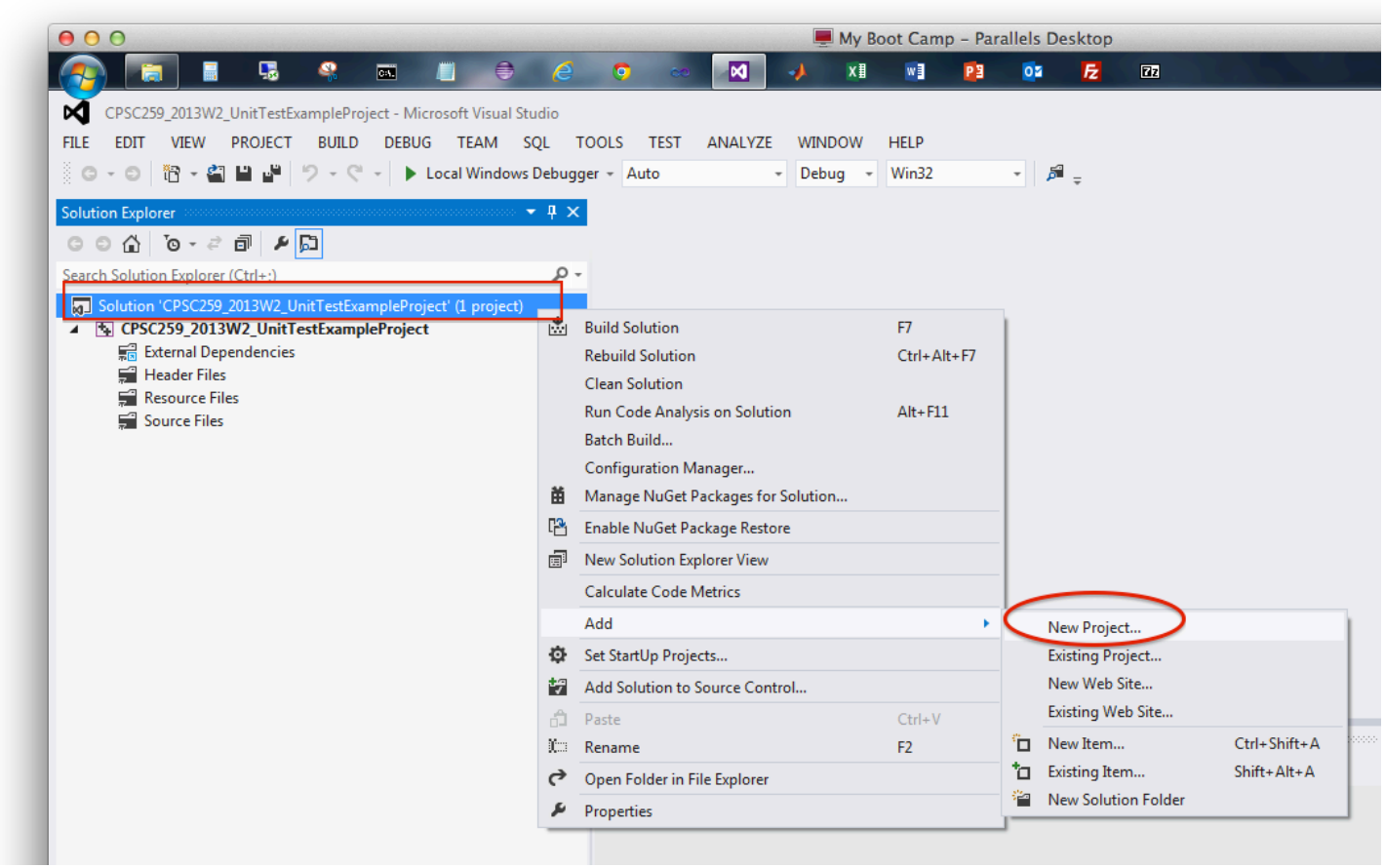
2. For project type select **Visual C++ -> General -> Empty Project**, choose a name, and select **OK**.



Ignore the warning message about file location

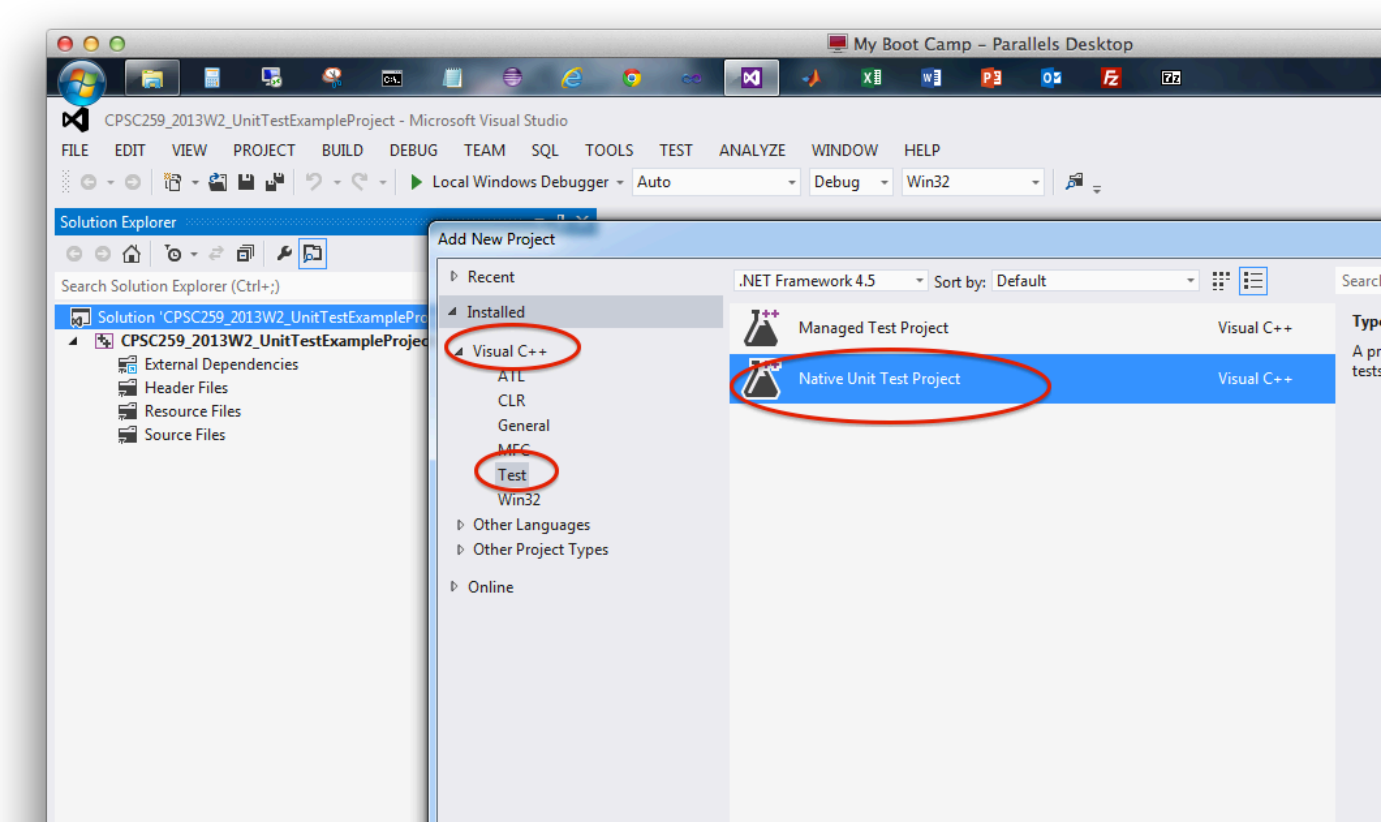
Step-by-step: Unit Tests and VS2012

3. In the Solution Explorer, select the Solution (make sure it's the Solution!) that contains the project we just created, right-click and choose **Add -> New Project**.



Step-by-step: Unit Tests and VS2012

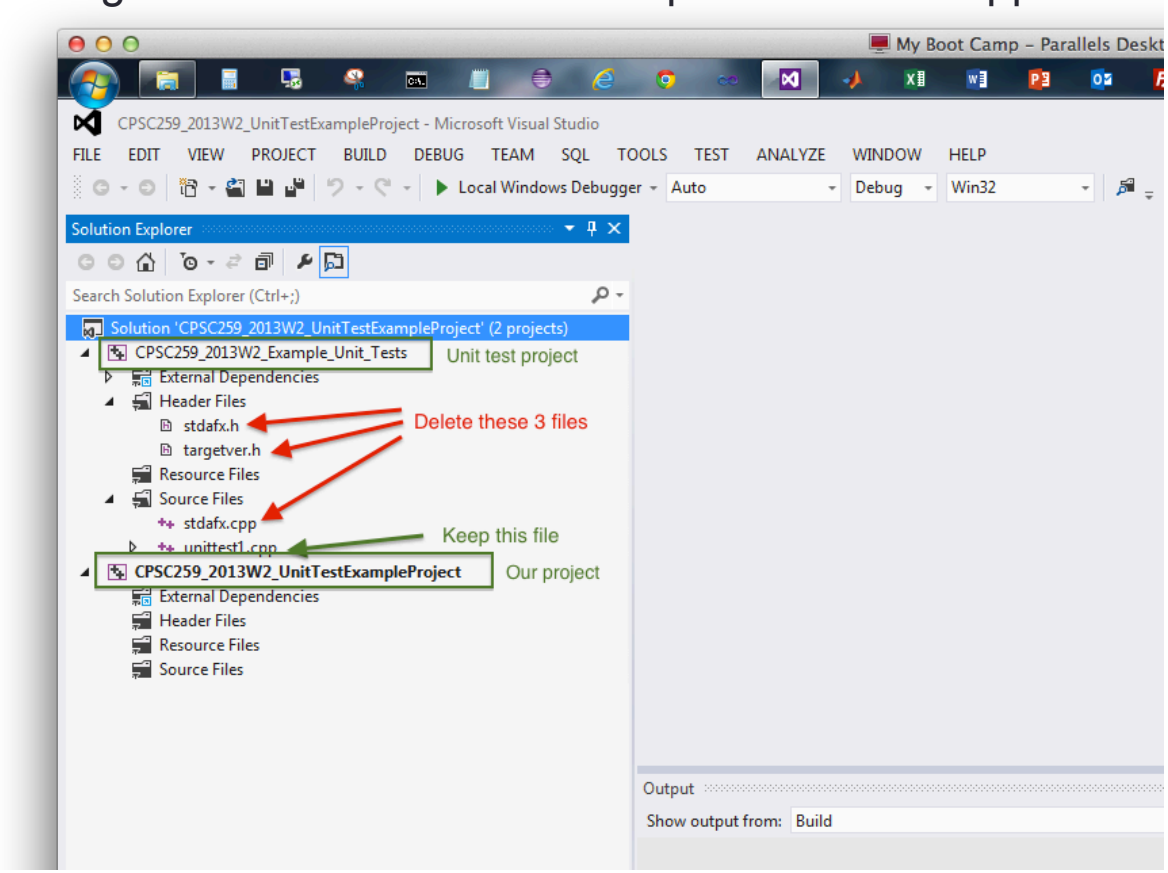
4. For the type, select **Visual C++ -> Test -> Native Unit Test Project**, choose a name*, and select **OK**.



* It usually makes sense to have UnitTest somewhere in the project name

Step-by-step: Unit Tests and VS2012

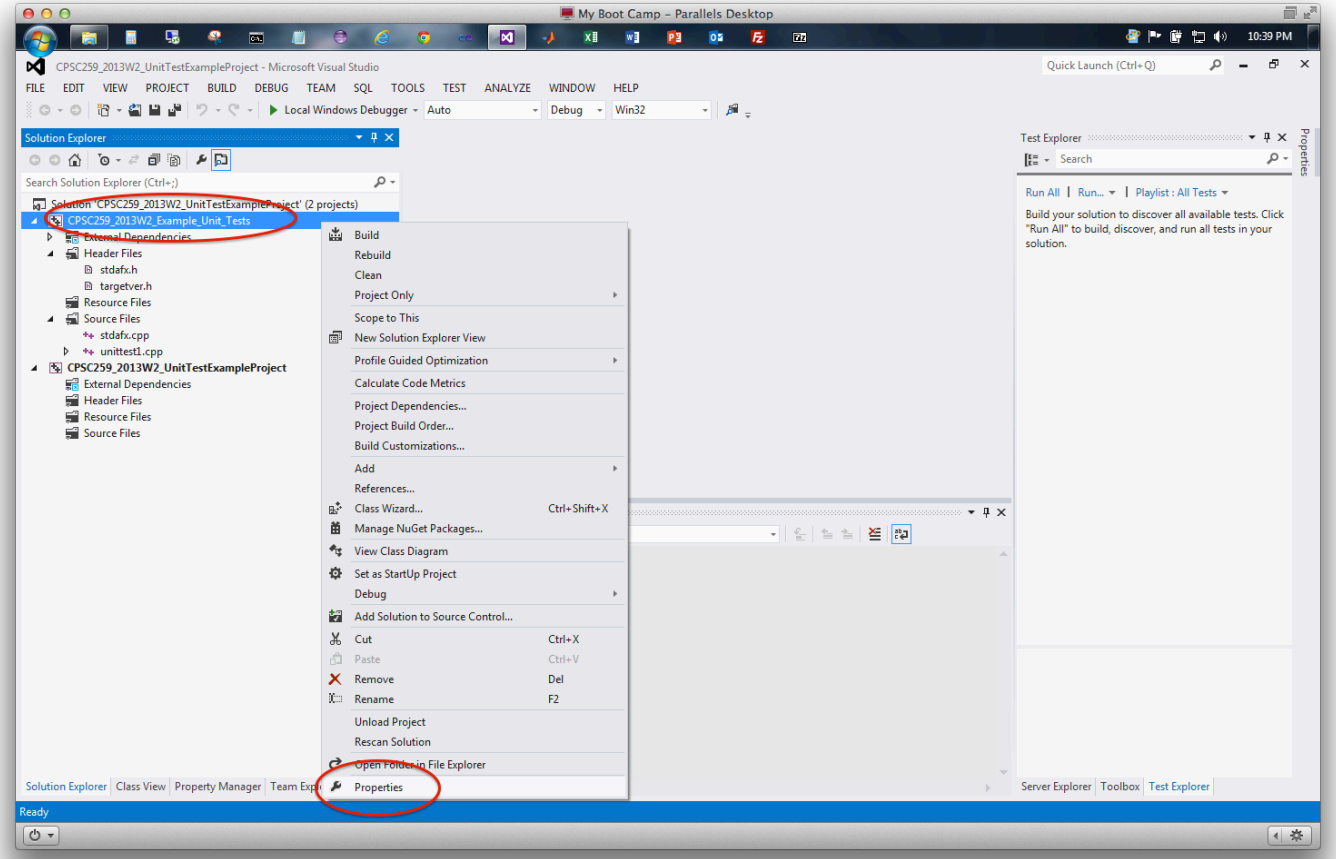
5. You should end up with something that looks like this: 2 projects contained in a single solution. Go ahead and delete the 3 files indicated by right-clicking them and choosing **Remove -> Delete** *. Keep the unittest1.cpp file:



* Those 3 files are needed for precompiled headers, which we won't use.

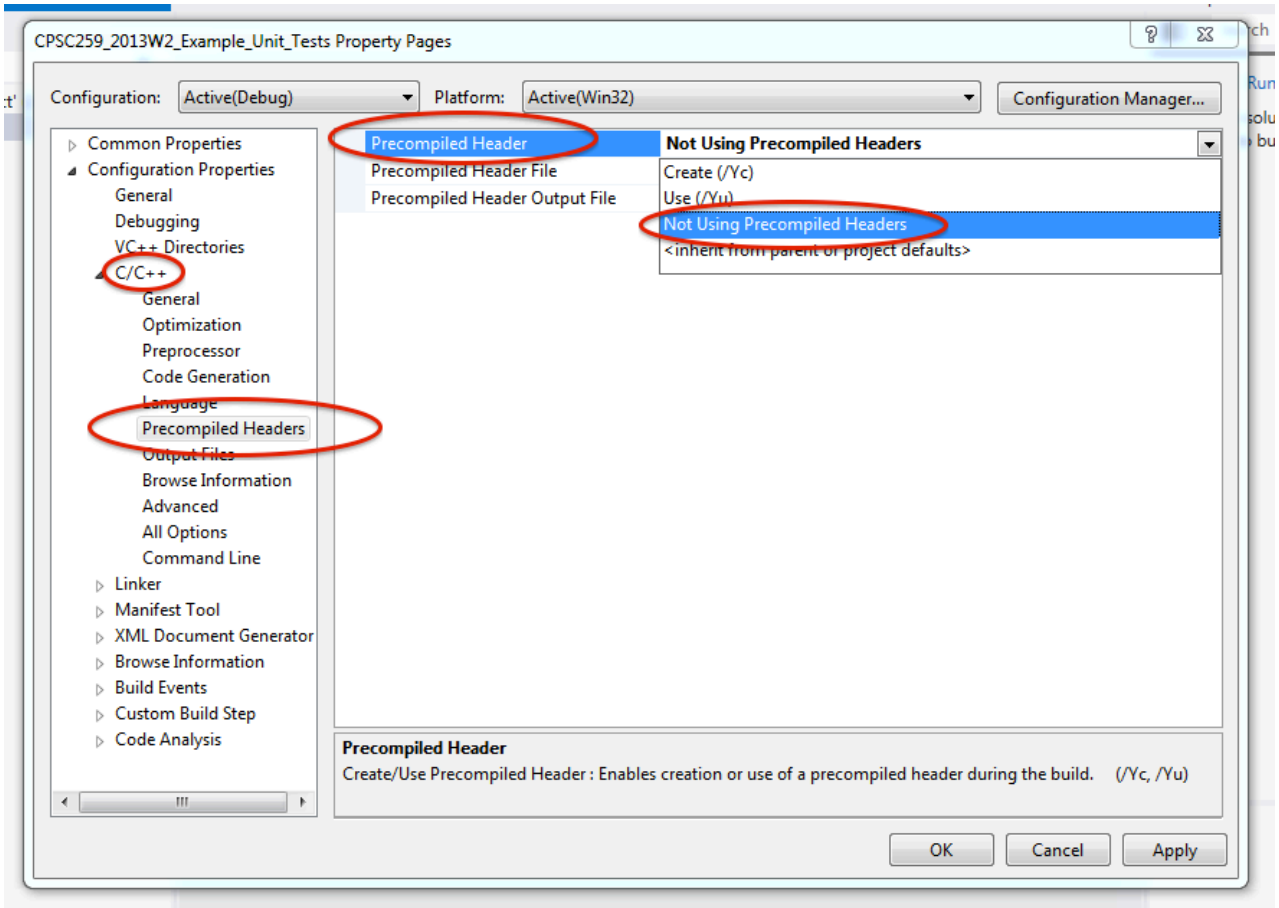
Step-by-step: Unit Tests and VS2012

6. Lets disable the use of precompiled headers in the Unit Test project, since we won't use them. Right-click the Unit Test project and choose **Properties**.



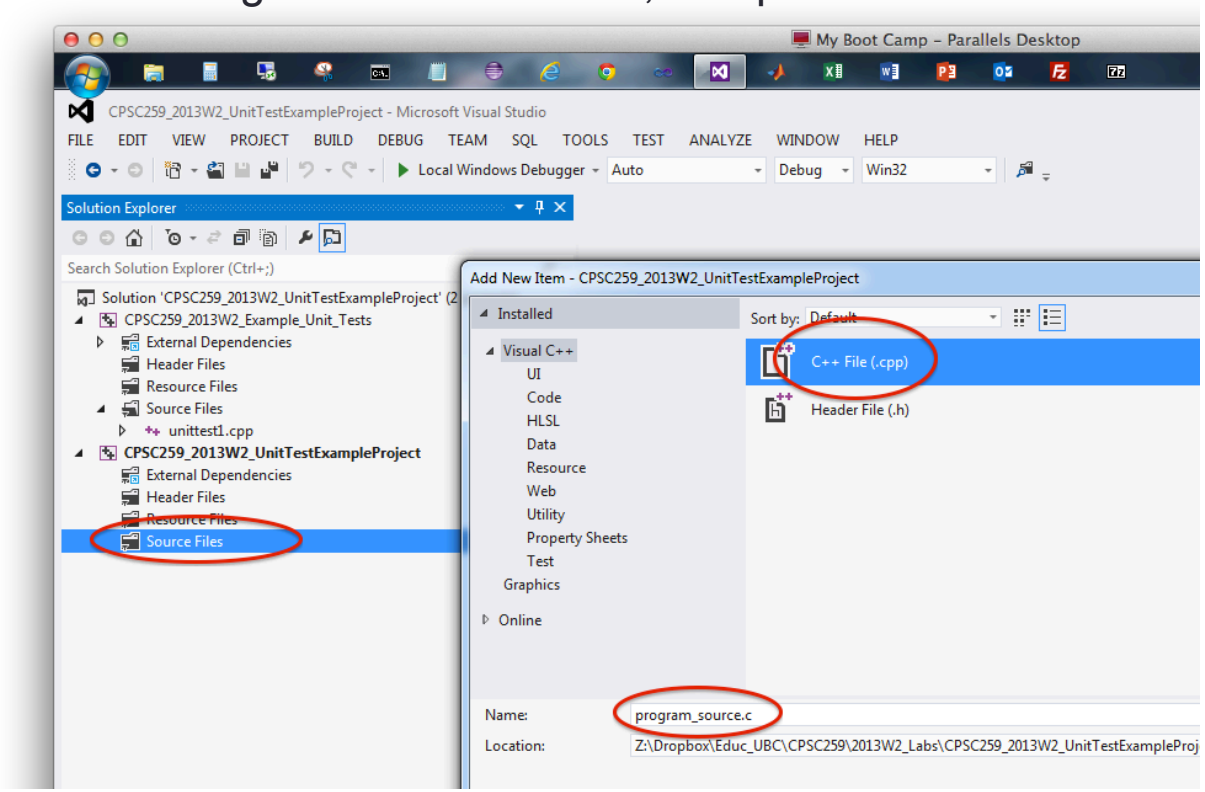
Step-by-step: Unit Tests and VS2012

7. Choose **Configuration Properties -> C/C++ -> Precompiled Headers -> Precompiled Header**. Select **Not Using Precompiled Headers**. Press **OK**.



Step-by-step: Unit Tests and VS2012

8. We need to create a project to test. Right-click the **Source Files** folder of the project to test, and choose **Add -> New** Item. Enter the name of your source file, remembering to use the .c suffix, and press **Add**.



Step-by-step: Unit Tests and VS2012

9. Inside the source file, let's write our main function and the skeleton for the function we want to test. That squiggly line means we have to create a header file too (see next step).

program_source.c

(Global Scope)

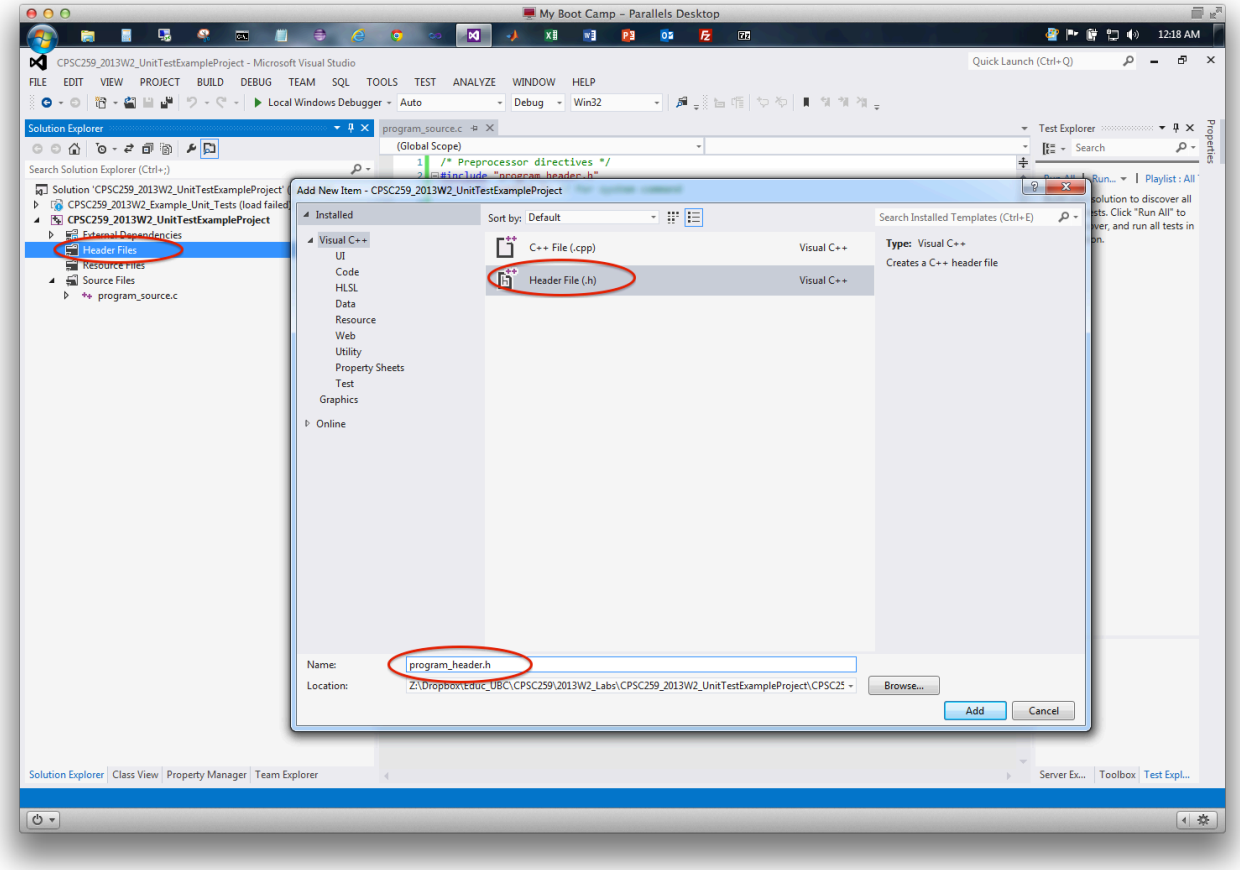
```c
/* Preprocessor directives */
#include "program_header.h"
#include <stdlib.h> // For system command

/*
 Main function drives the program.  Every C program must have one main function.
 PRE:       NULL (no pre-conditions)
 POST:      NULL (no side-effects)
 RETURN:    IF the program exits correctly THEN 0 ELSE 1
 */
int main (void)
{
    system("pause");
    return 0;
}

/*
 sum_array
 PURPOSE: returns the sum of the integers in the passed array
 PRE:     an_array is an array of integers
 PRE:     the_length_of_the_array is the correct length of an_array
 POST:    no side-effects, e.g., nothing is printed or changed
 RETURN:  the sum of the ints in the array, also an int
 */
int sum_array(int an_array[], int the_length_of_the_array)
{
    return 0; // We have to return something because the return value is an int!
}
```

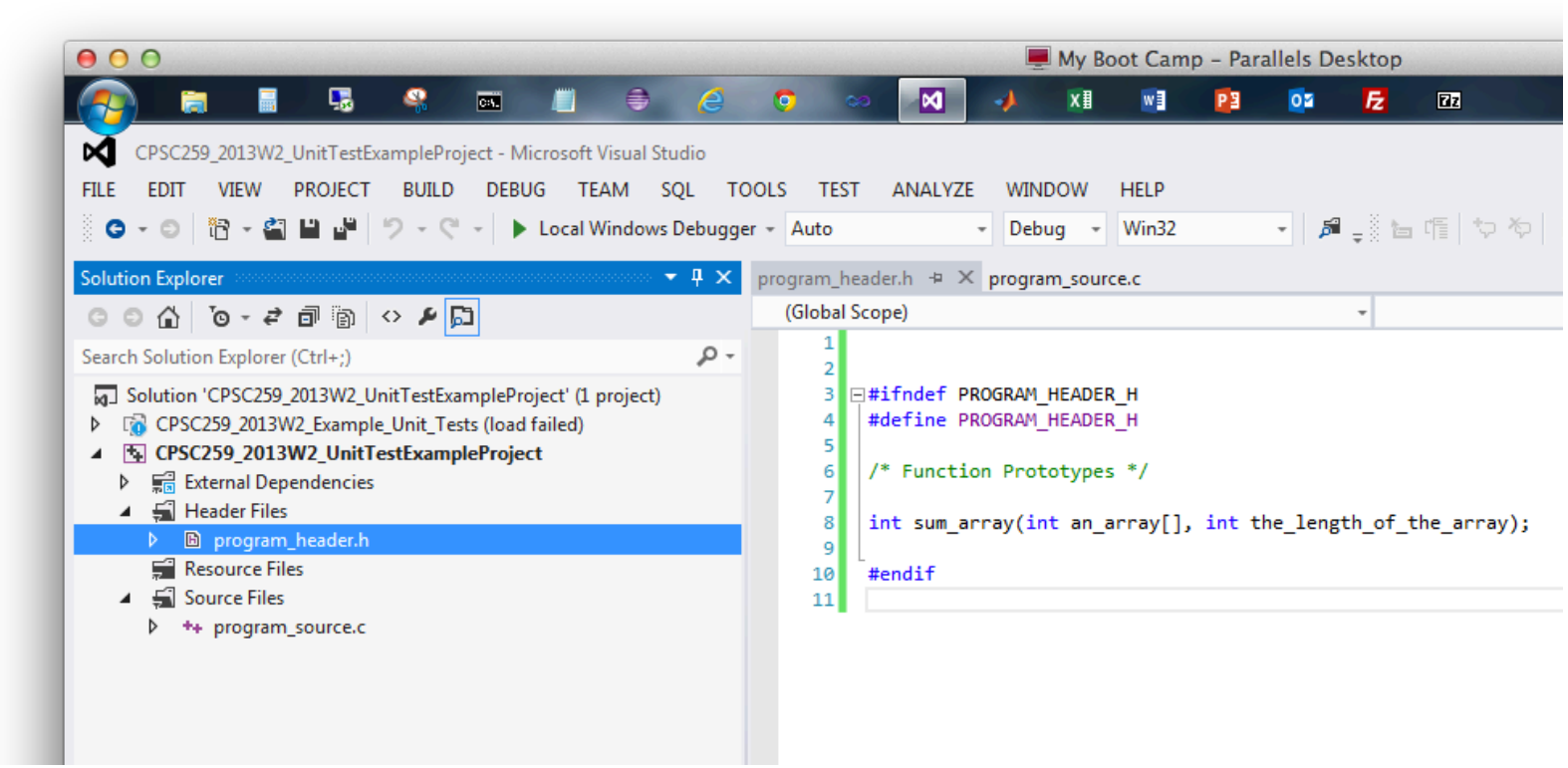
Step-by-step: Unit Tests and VS2012

10. Right-click the **Header Files** folder of the project to test, and choose **Add -> New** Item. Enter the name of your header file and press **Add**.



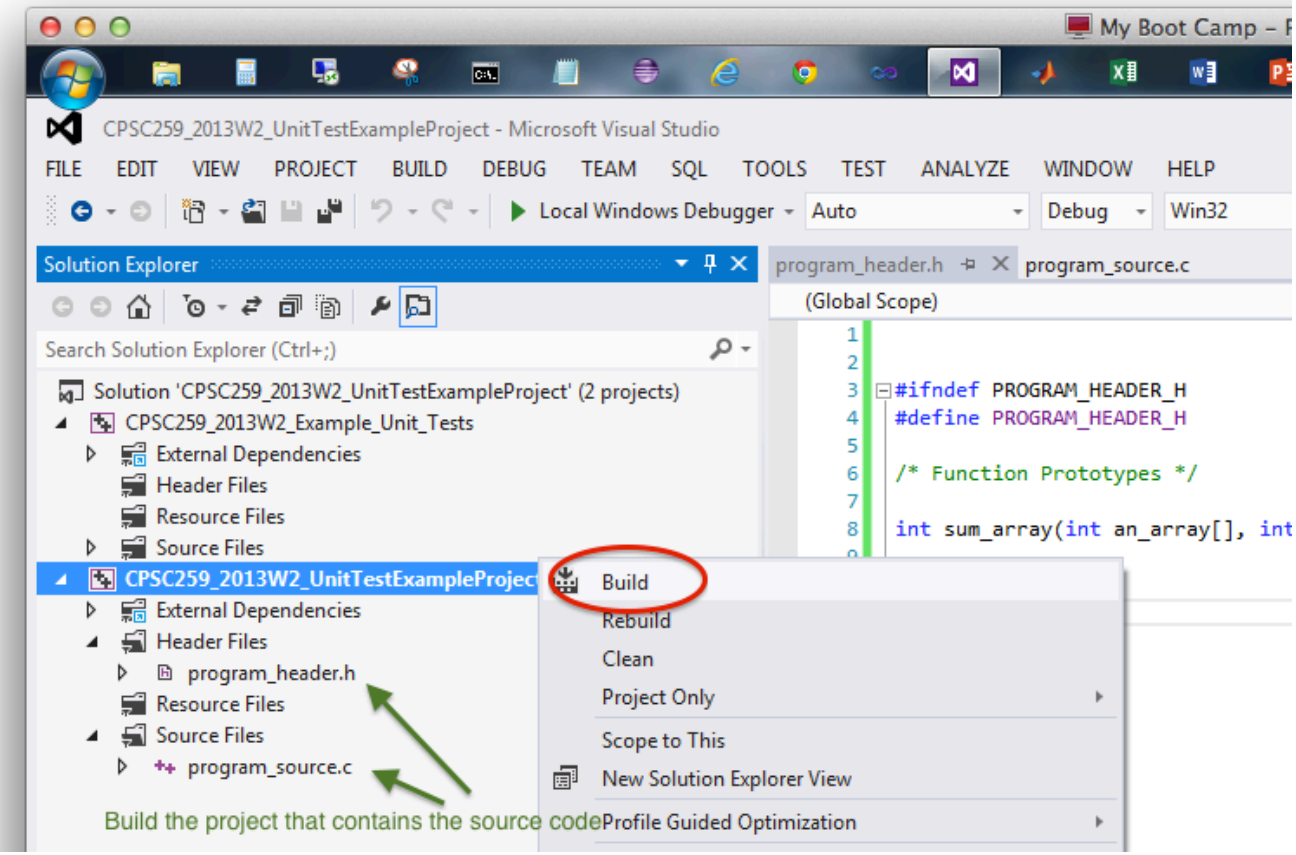
Step-by-step: Unit Tests and VS2012

11. Inside the header file, let's write our function prototype. Remember to surround your prototypes with include guards:



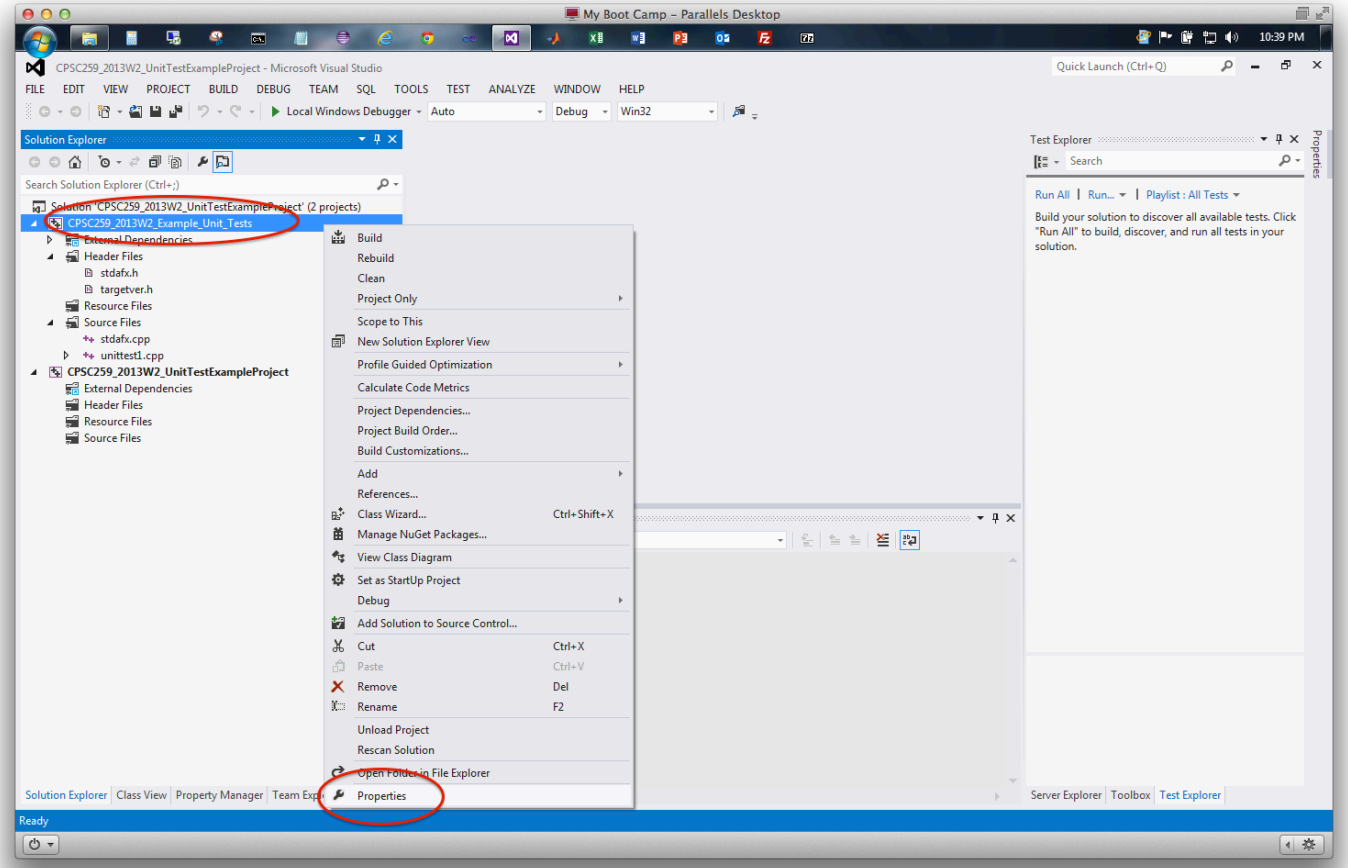
Step-by-step: Unit Tests and VS2012

12. Now that we've written our source code, we need to build the project. Right-click the project that contains the source code and select **Build**.



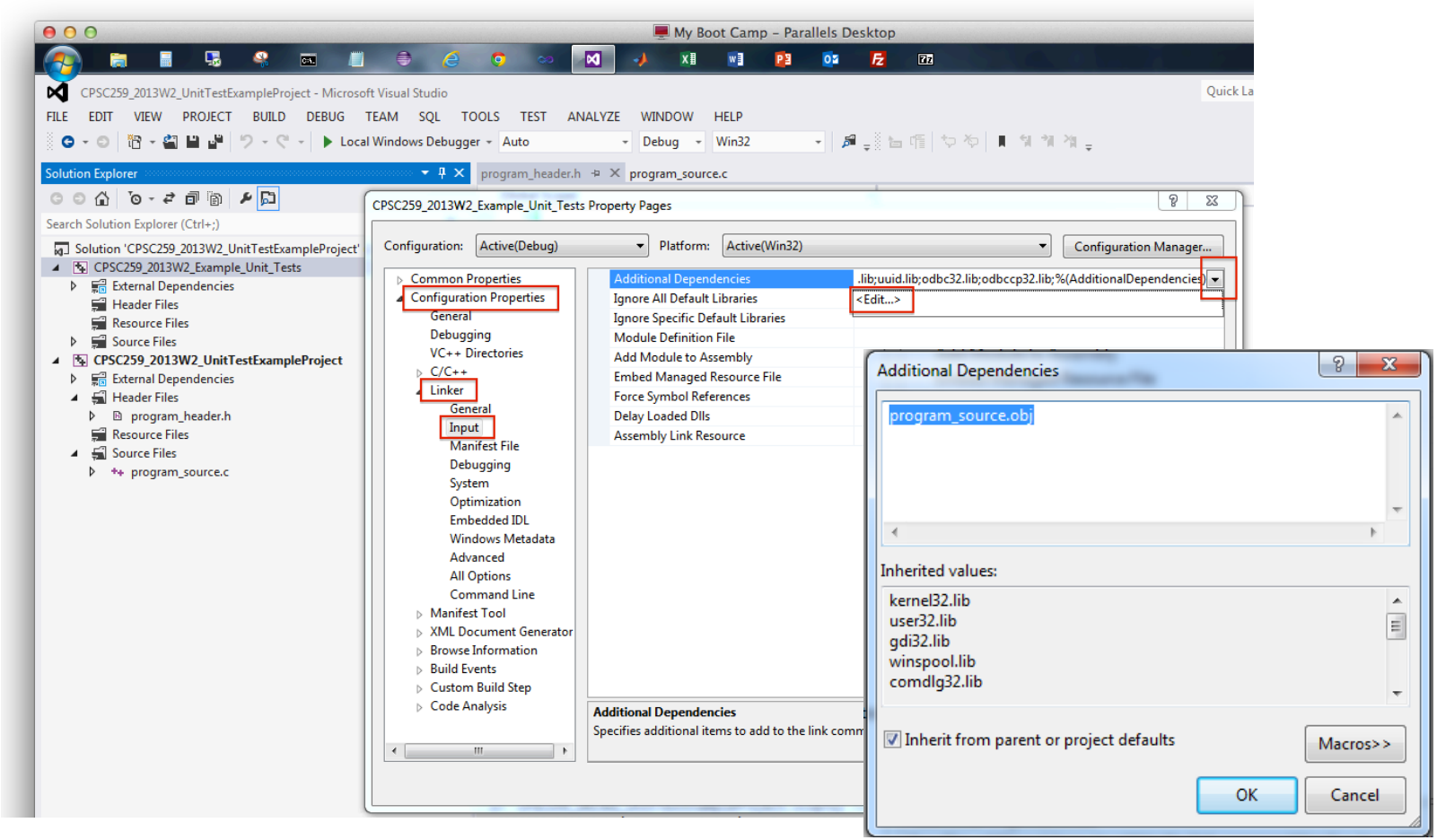
Step-by-step: Unit Tests and VS2012

13. Building the source code project generates an .obj file. The unit tests will execute against this .obj file. Let's configure the tests now. Right-click the Unit Test project and select **Properties**.



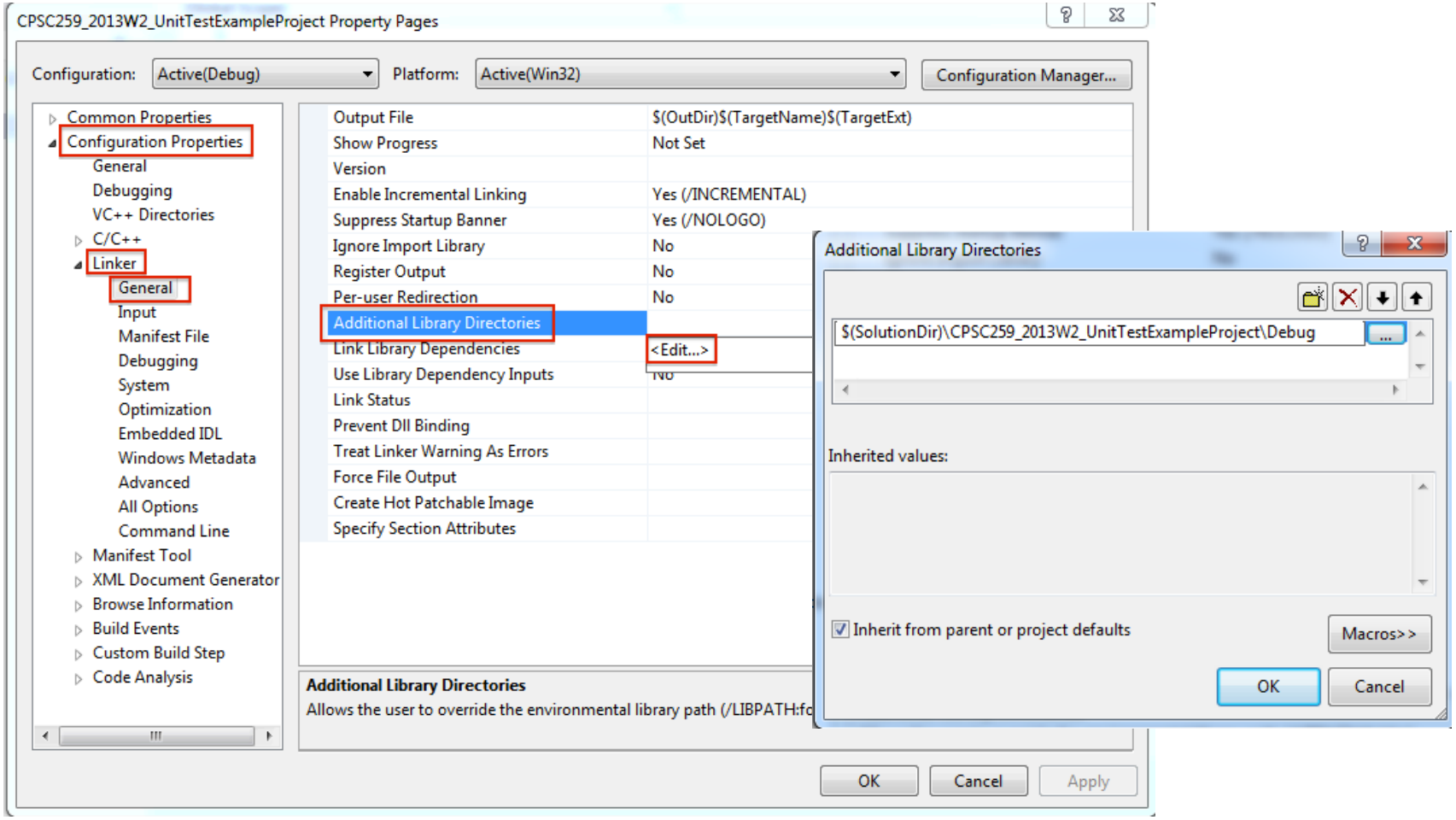
Step-by-step: Unit Tests and VS2012

14. Choose **Configuration Properties -> Linker -> Input -> Additional Dependencies**. Choose **Edit**, and add the names of the .obj file(s) to the list. They will have the form source_file_name.obj. Select **OK** once.



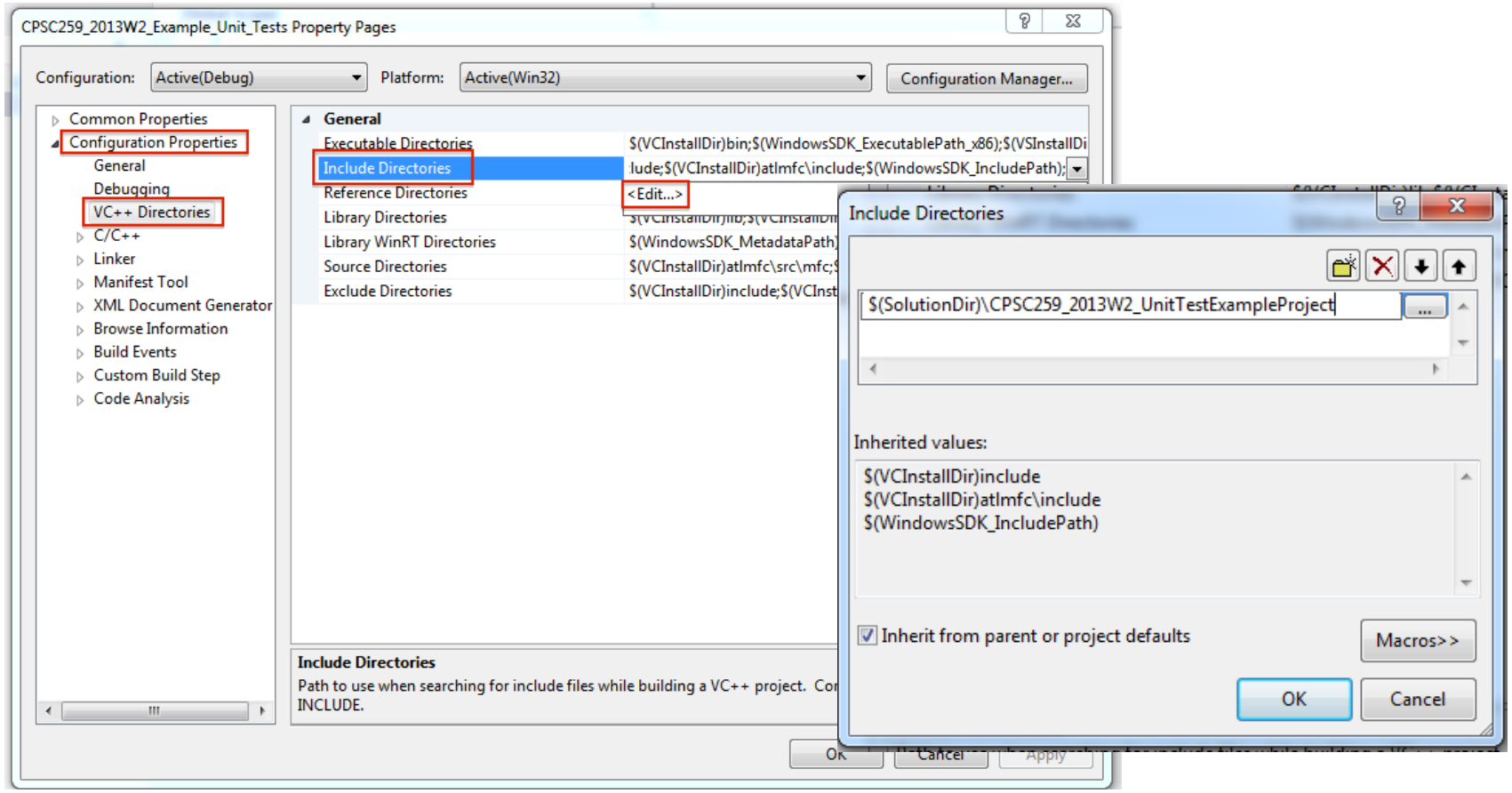
Step-by-step: Unit Tests and VS2012

15. Choose **Configuration Properties -> Linker -> General -> Additional Library Directories**. Choose **Edit**, and add the directory path of the .obj file(s). The path is typically within the build folder of the project under test. Use the Visual Studio macro for the solution file folder root, then choose **OK**:



Step-by-step: Unit Tests and VS2012

16. Choose **Configuration Properties -> VC++ Directories ->Include Directories**. Choose **Edit**, and then add the header directory of the project under test. You can use a Visual Studio source folder macro here, too:



Step-by-step: Unit Tests and VS2012

17. Only a few steps left. We need to edit the unittest1.cpp file inside the Unit Test Source Folders file. Make these two changes:

My Boot Camp – Parallels Desktop

CPSC259_2013W2_UnitTestExampleProject - Microsoft Visual Studio

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ANALYZE WINDOW HELP

Local Windows Debugger | Auto | Debug | Win32

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'CPSC259_2013W2_UnitTestExampleProject' (2 projects)
  - CPSC259_2013W2_Example_Unit_Tests
    - External Dependencies
    - Header Files
    - Resource Files
    - Source Files
      - unittest1.cpp
  - CPSC259_2013W2_UnitTestExampleProject
    - External Dependencies
    - Header Files
      - program_header.h
    - Resource Files
    - Source Files
      - program_source.c

unittest1.cpp | program_header.h | program_source.c

(Global Scope)

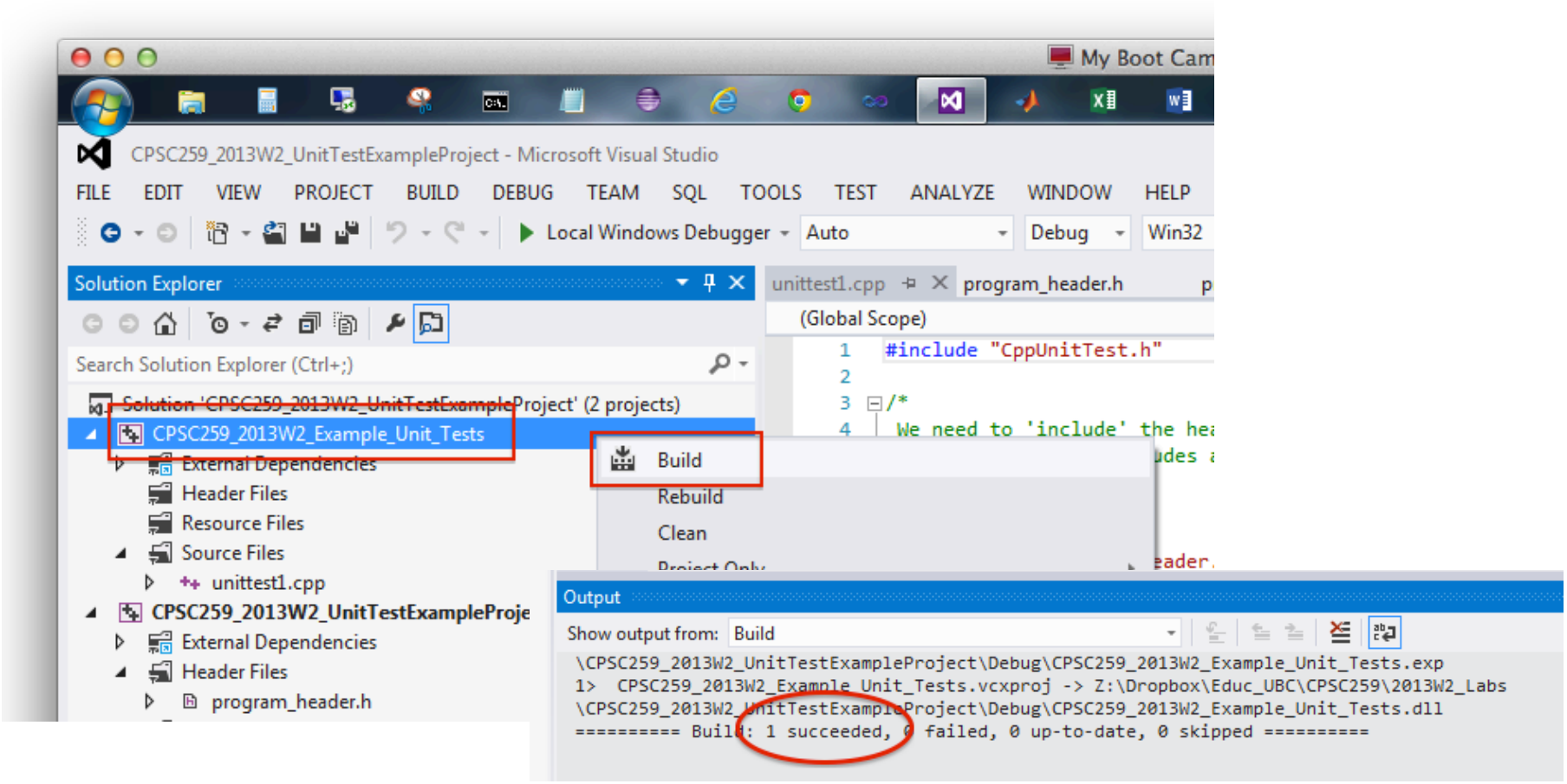
```
1  #include "stdafx.h"
2  #include "CppUnitTest.h"
3
4  /*
5    We need to 'include' the header file(s) for the project we are testing
6    Do this AFTER the includes and BEFORE the using namespace line.
7    */
8
9  extern "C" {
10     #include "program_header.h"
11 }
12
13
14 using namespace Microsoft::VisualStudio::CppUnitTestFramework;
15
16 namespace CPSC259_2013W2_Example_Unit_Tests
17 {
18     TEST_CLASS(UnitTest1)
19     {
20     public:
21
22         TEST_METHOD(TestMethod1)
23         {
24             // TODO: Your test code here
25         }
26
27     };
28 }
```

Delete this preprocessor directive

Add this code AFTER the include statements and BEFORE the using namespace statement

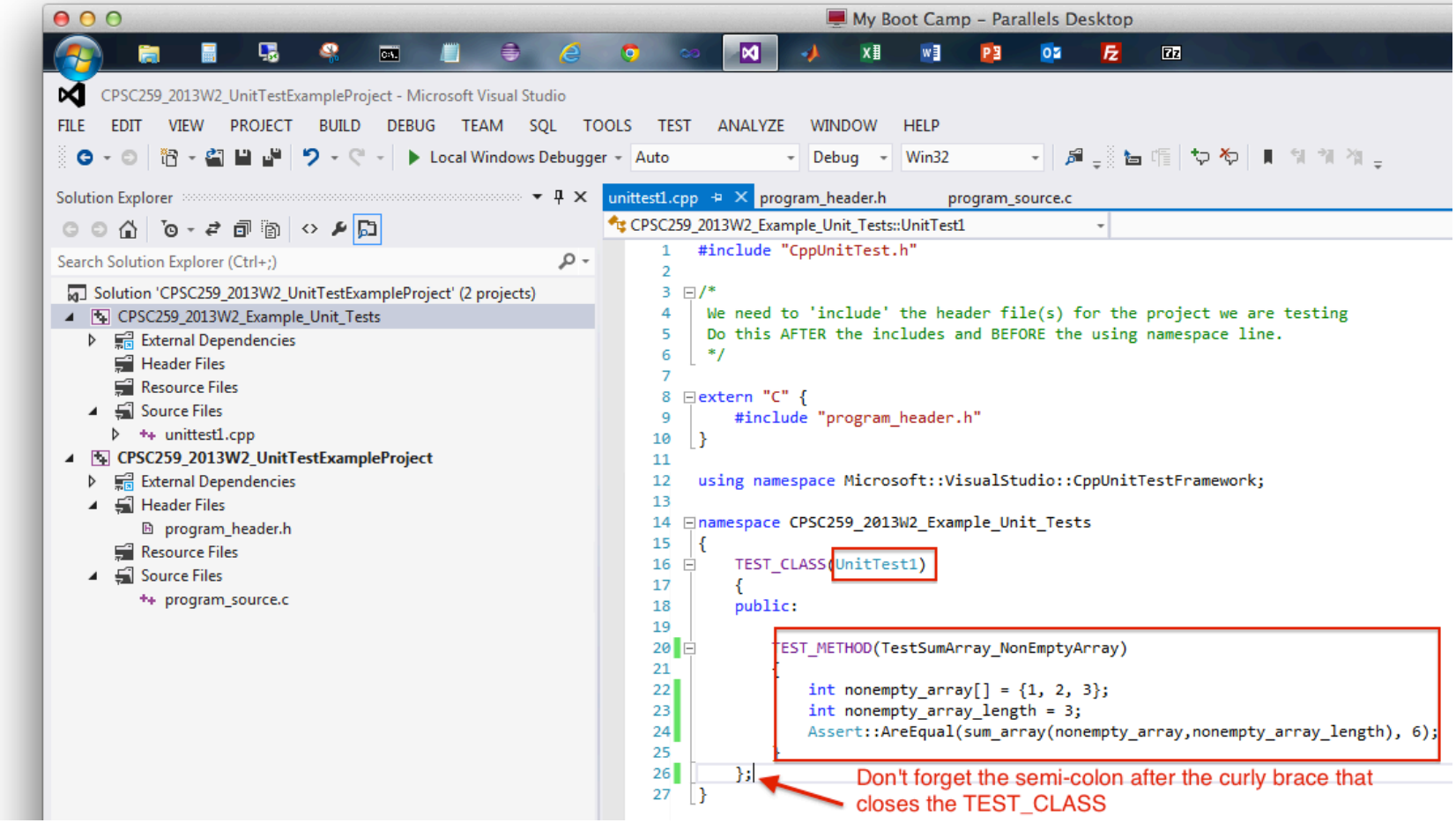
Step-by-step: Unit Tests and VS2012

18. Now we can build our unit tests. Right-click the Unit Test project and select **Build**. If it worked, your output will say ‘succeeded’:



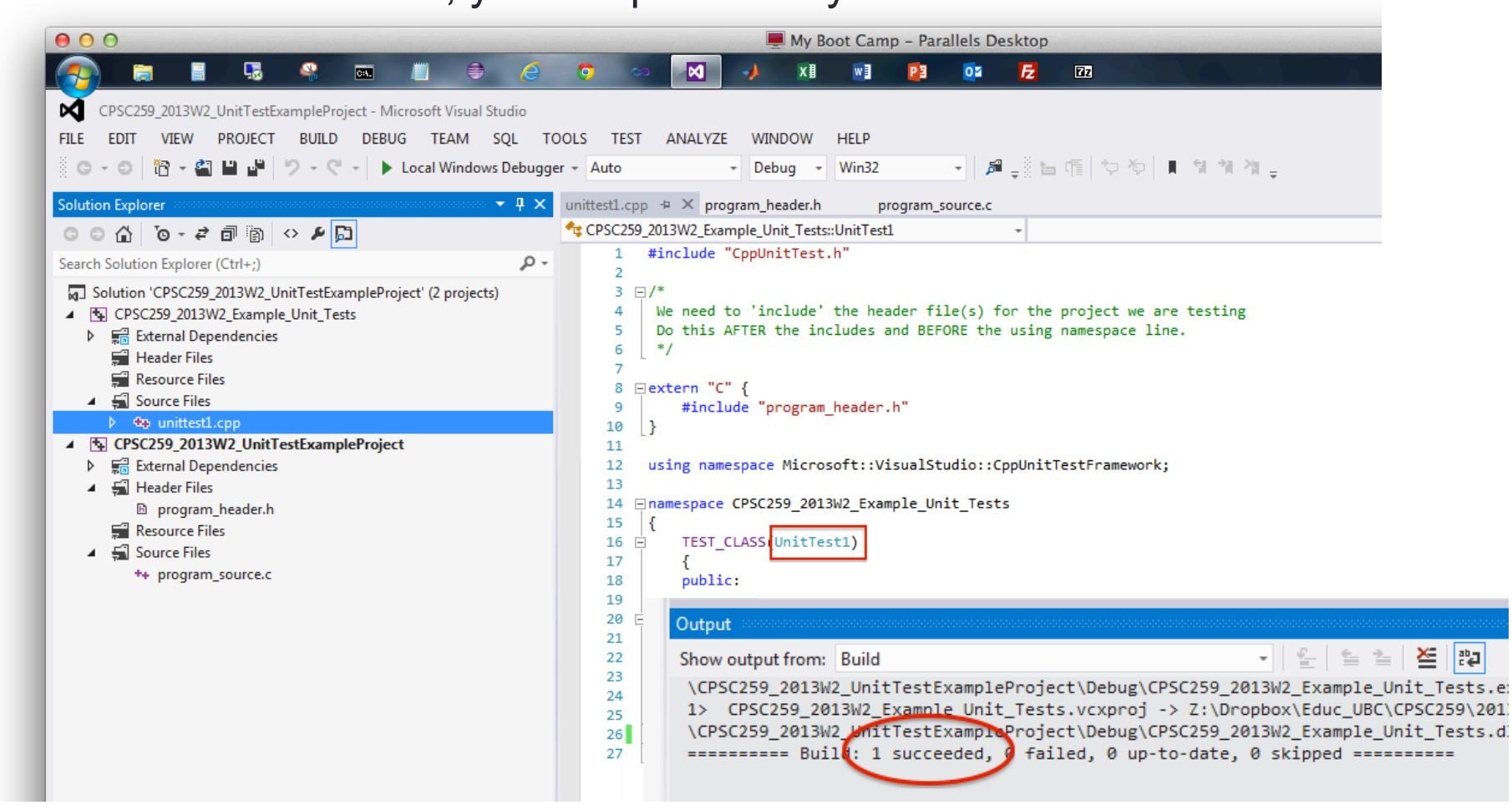
Step-by-step: Unit Tests and VS2012

19. Lets write a unit test now. Edit the unit test code to look like this. Can you guess what the Assert::AreEqual statement means?



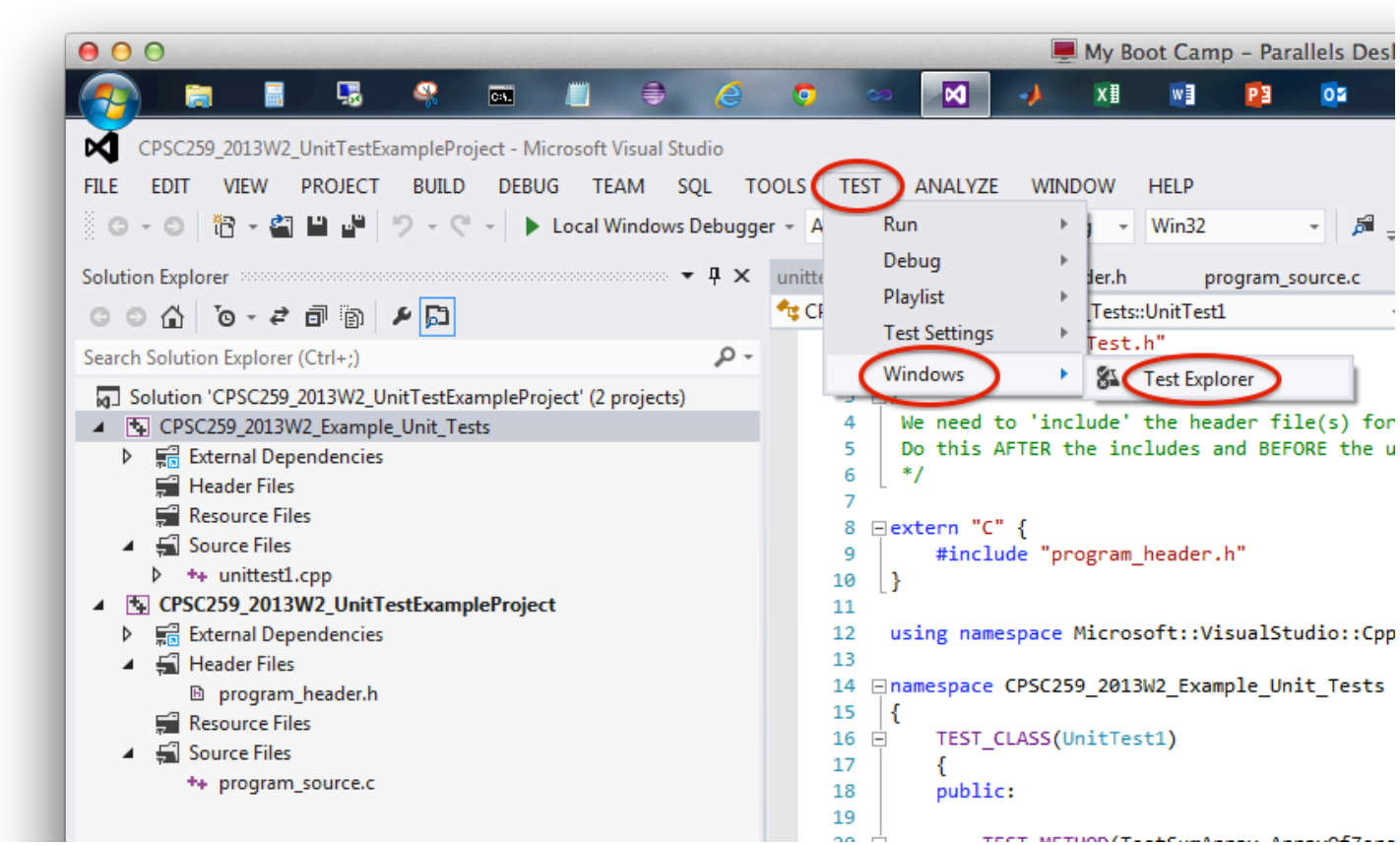
Step-by-step: Unit Tests and VS2012

20. Rebuild our unit tests. Right-click the Unit Test project and select **Build**. If it worked, your output will say 'succeeded':



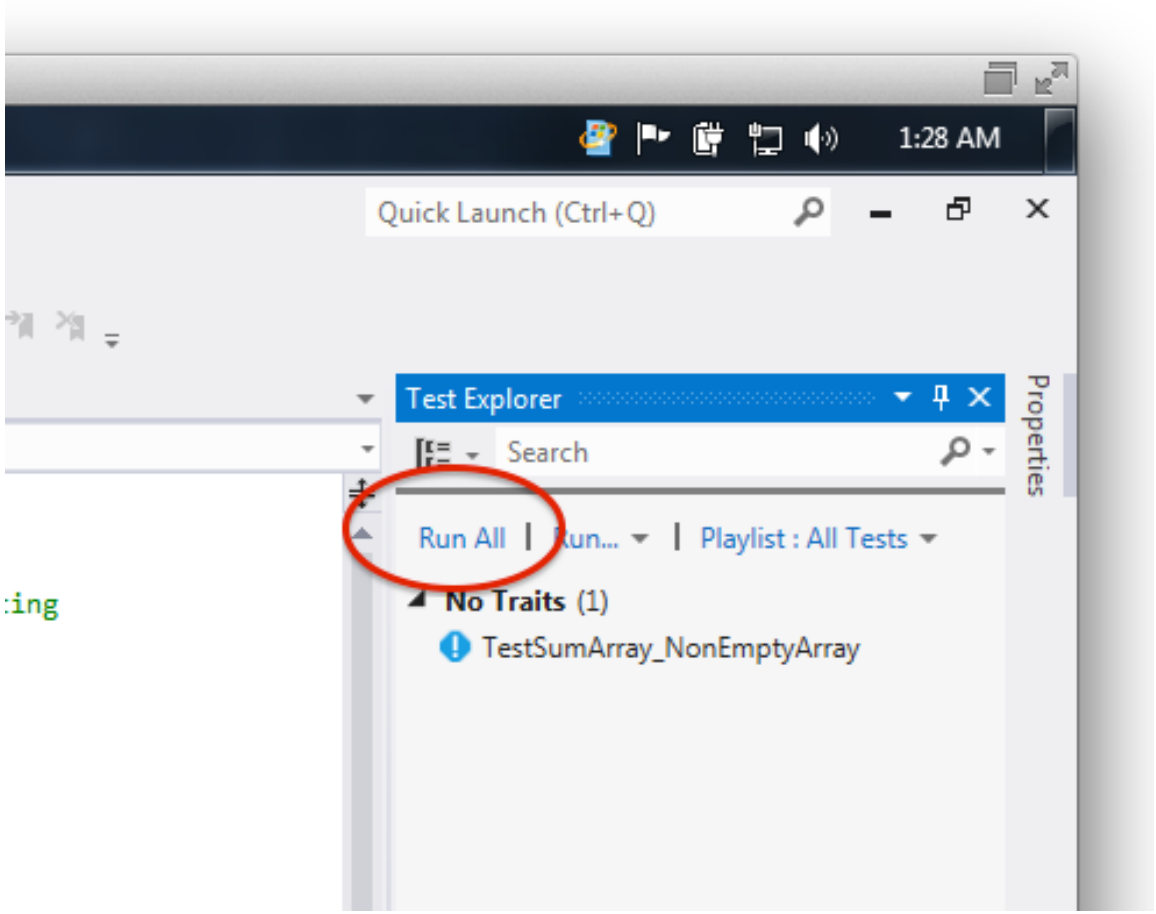
Step-by-step: Unit Tests and VS2012

21. If the Test Explorer isn't visible, make it visible by selecting **Menu: Test -> Windows -> Test Explorer**.



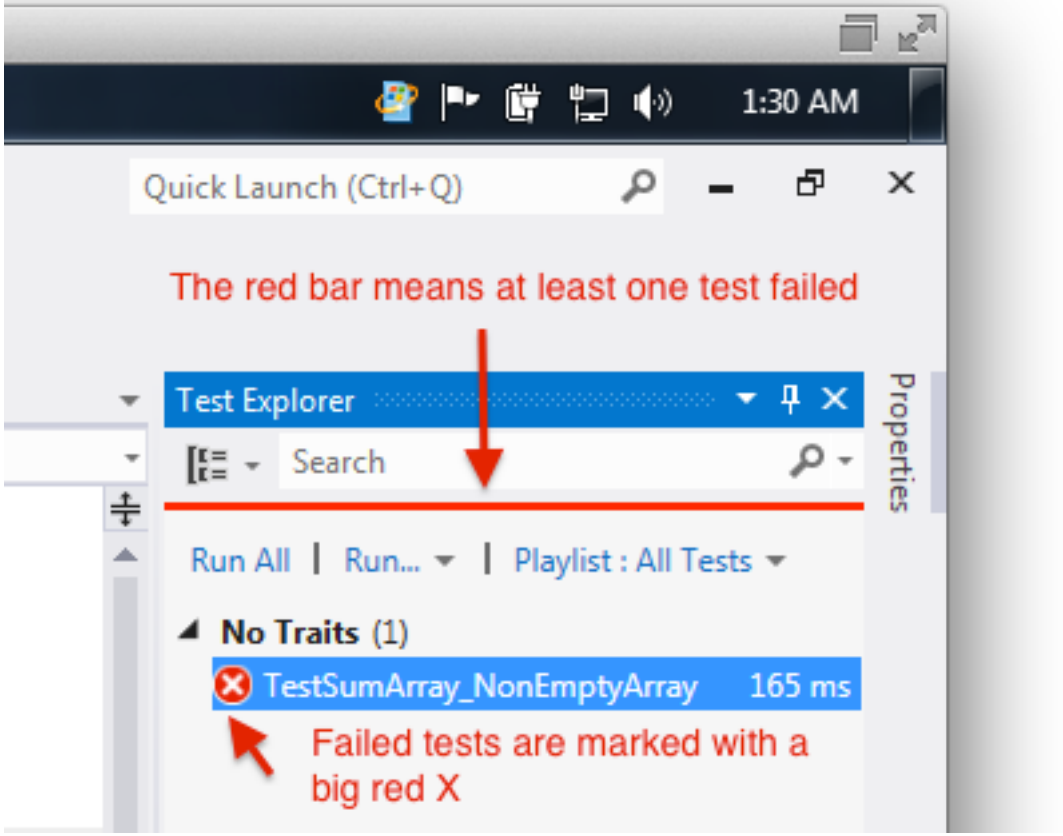
Step-by-step: Unit Tests and VS2012

22. In the top of the Test Explorer, click **Run All**.



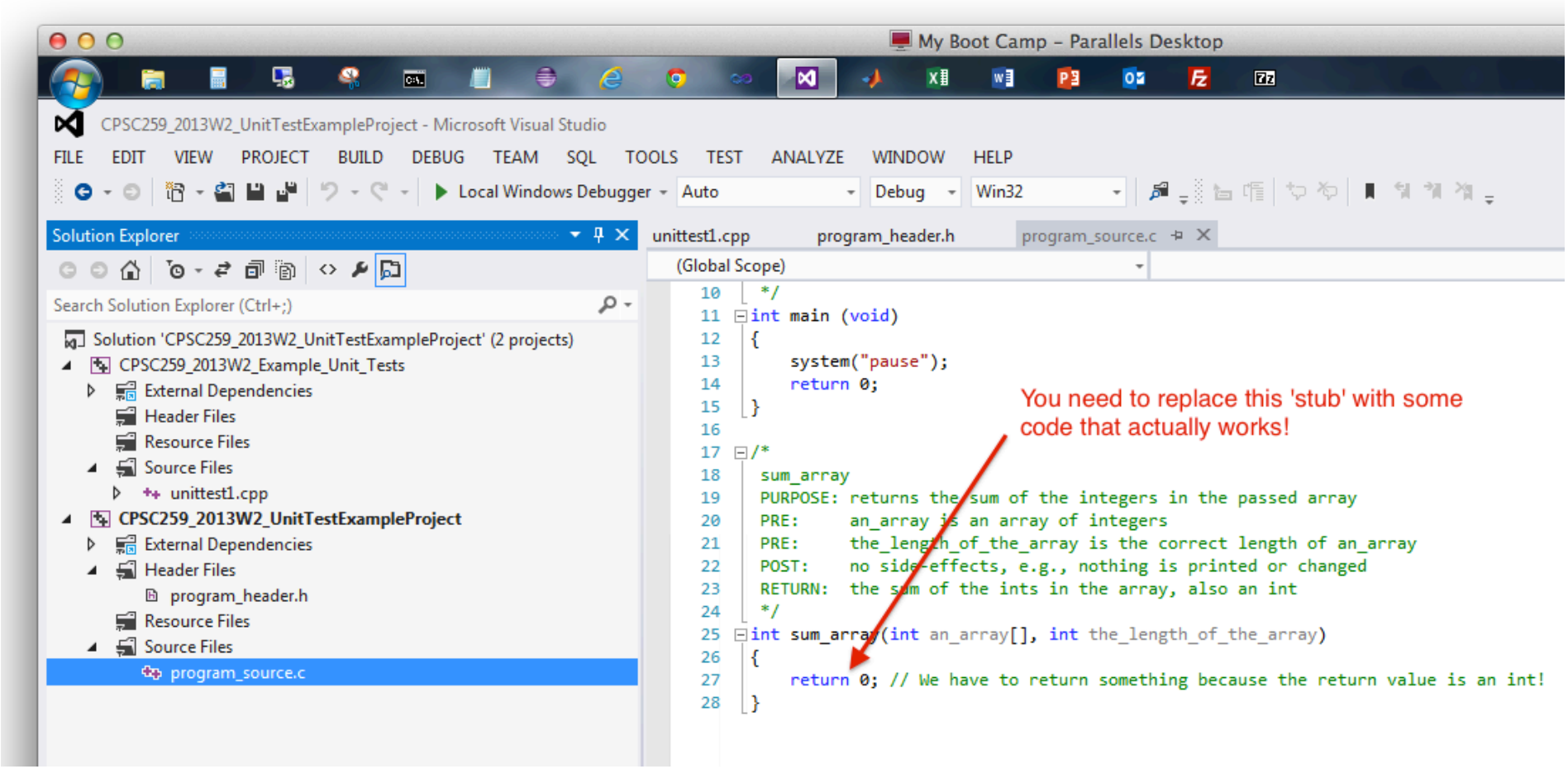
Step-by-step: Unit Tests and VS2012

23. It failed! Our test was basic, so the problem must be in our source code.



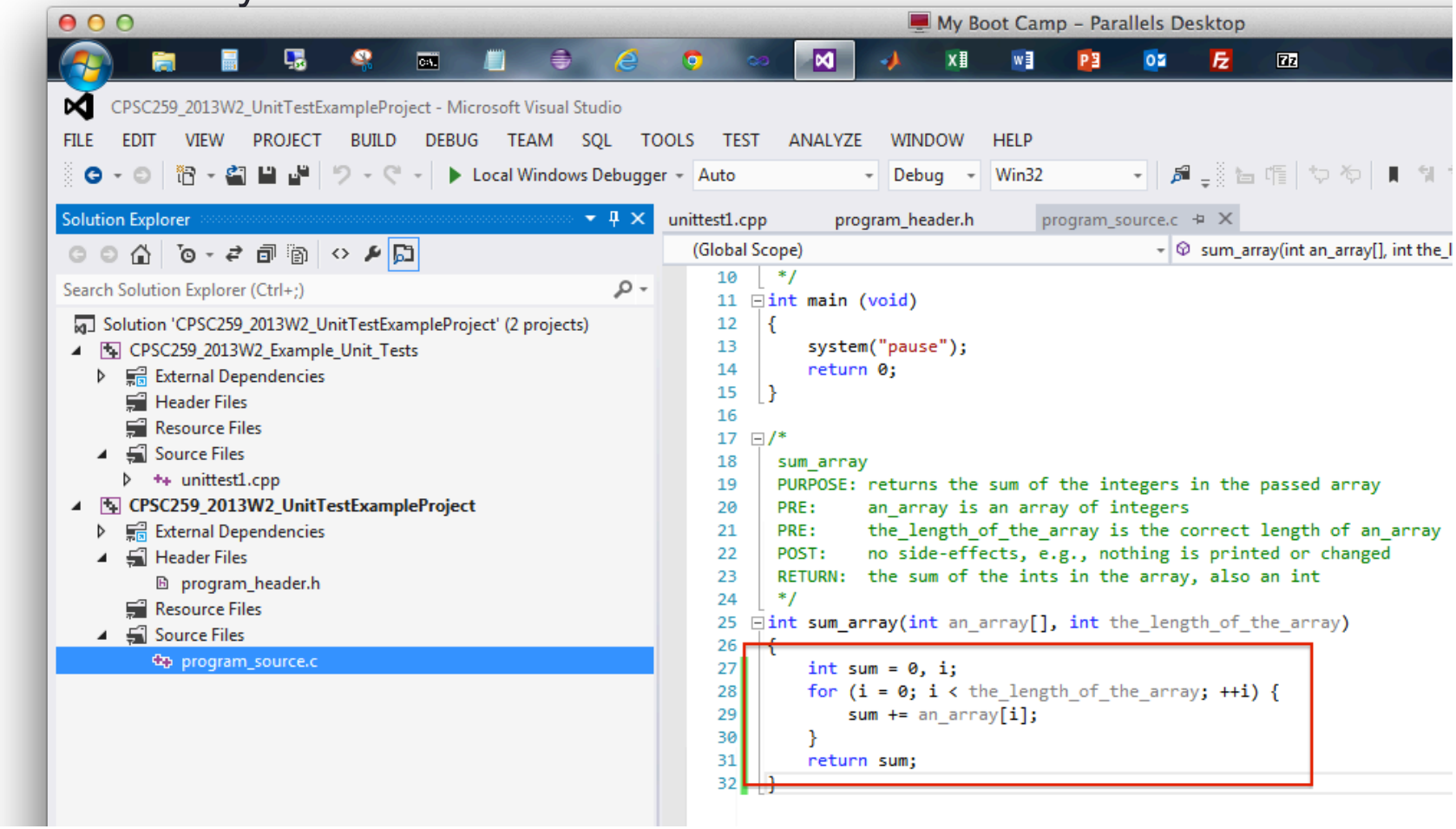
Step-by-step: Unit Tests and VS2012

24. Lets visit the source code. Of course! We forgot to implement it!



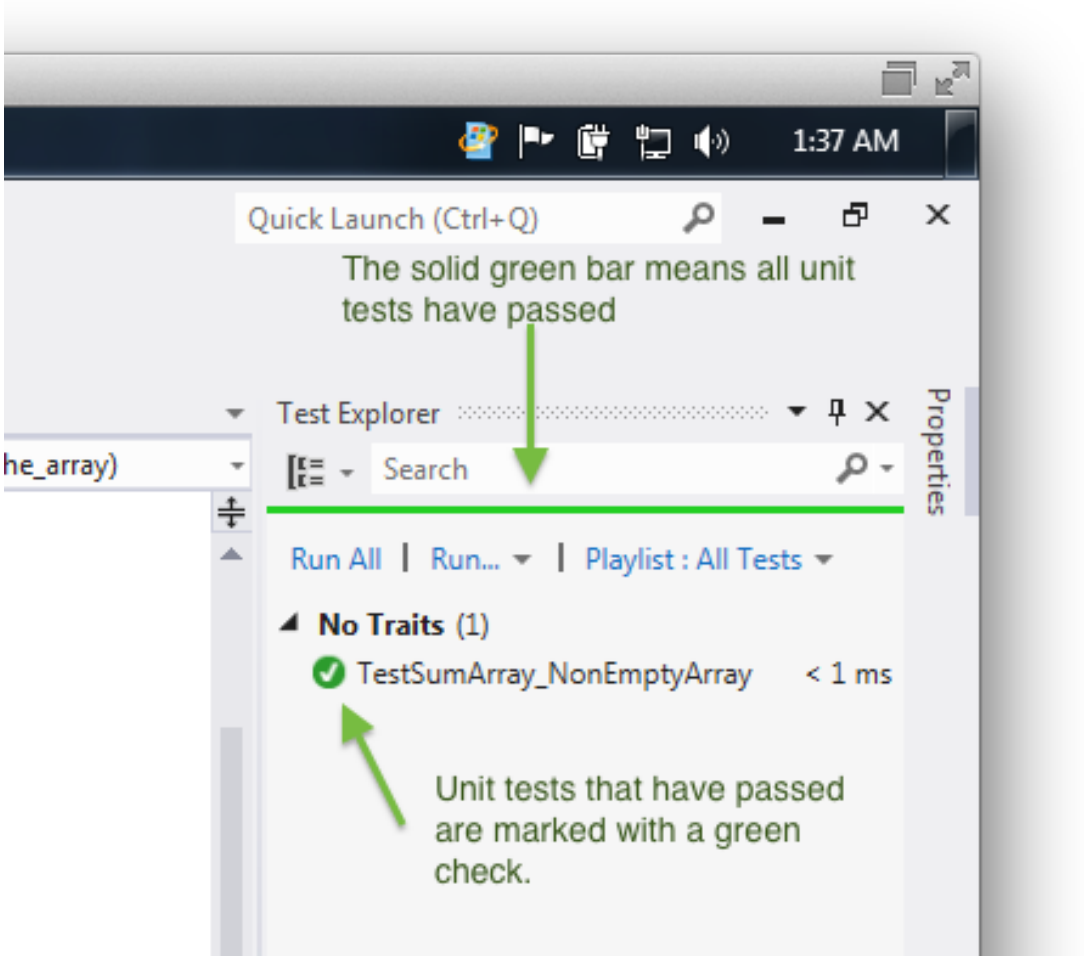
Step-by-step: Unit Tests and VS2012

25. Implement the sum_array function, rebuild your project, and then re-run your tests.



Step-by-step: Unit Tests and VS2012

26. Passed!



Step-by-step: Unit Tests and VS2012

27. We should write more than 1 unit test for each function. **Each time you add a test, re-build the Unit Test project, and re-run the tests**. Your goal is to test each function until you're sure each function executes correctly in response to all possible combinations of input. As the term progresses, we'll show you how to write more complicated unit tests. You can also learn from the tests we provide in your frameworks. Good luck!

unittest1.cpp | program_header.h | program_source.c

CPSC259_2013W2_Example_Unit_Tests::UnitTest1 | TestSumArray_ArrayOfZeros()

```
 4   We need to  include  the header file(s) for the project we are testing
 5   Do this AFTER the includes and BEFORE the using namespace line.
 6   */
 7
 8  extern "C" {
 9      #include "program_header.h"
10  }
11
12  using namespace Microsoft::VisualStudio::CppUnitTestFramework;
13
14  namespace CPSC259_2013W2_Example_Unit_Tests
15  {
16      TEST_CLASS(UnitTest1)
17      {
18      public:
19
20          TEST_METHOD(TestSumArray_NonEmptyArray)
21          {
22              int nonempty_array[] = {1, 2, 3};
23              int nonempty_array_length = 3;
24              Assert::AreEqual(sum_array(nonempty_array,nonempty_array_length), 6);
25          }
26
27          TEST_METHOD(TestSumArray_ArrayOfZeros)
28          {
29              int array_of_zeros[] = {0, 0, 0};
30              int array_of_zeros_length = 3;
31              // TODO: Your test code here
32              Assert::AreEqual(sum_array(array_of_zeros,array_of_zeros_length), 0);
33          }
34      };
35  }
```

A second unit test