

Visual Studio 2017 and Google Test

Introduction

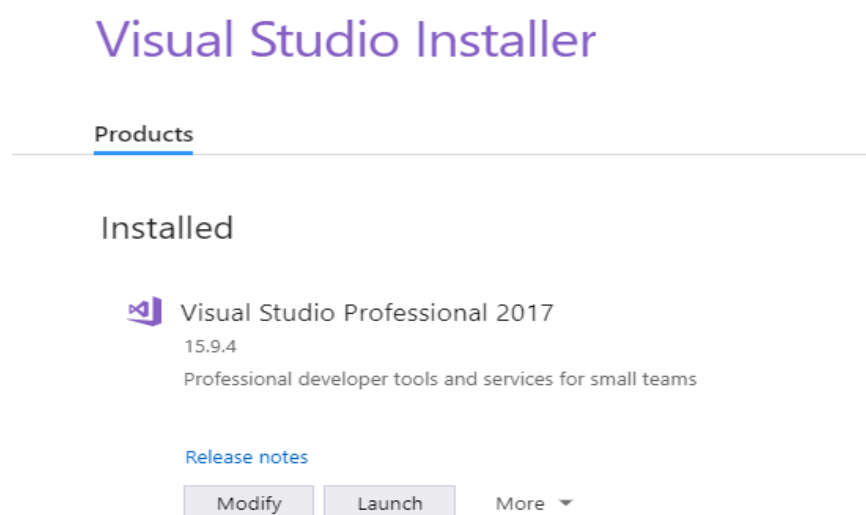
This is a primer that gets you setup in Visual Studio 2017 (VS17) to use Google Test (gtest) and in covering some of the basics to understanding and creating a google test. Through my research there are a few different ways to create a test and test application functions within a test file, or to create a test that tests external applications. In this document all examples are based around using VS17 to develop the tests and external applications.

As I begin to learn how to use the paradigm for testing it became clear very early that there was not a lot of good examples on the concepts and implementation of various methods of testing using gtest. So my journey began in an attempt to document and make it easier for the beginner to get started or those with some experience to maybe improve their skills. With that in mind, lets get started.

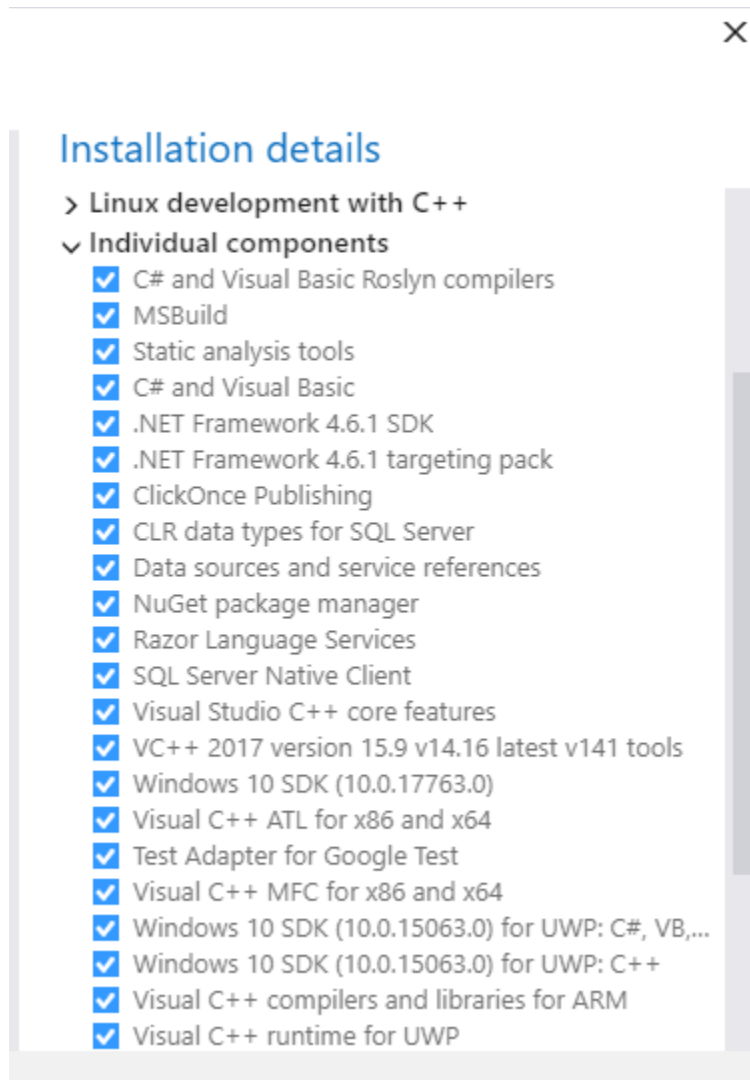
Installing Google Test into Visual Studio 2017.

Verify/Install gtest in VS17

Open the Visual Studio Installer to see something like this:



Click on the modify button to open the options for modifying VS17. Look at “Installation details” list on the right side and expand the list under “Individual components” if not already expanded. If need be, use the right slider bar to move the various options up and down. Under the “Individual components” section look for an entry called “Test Adapter for Google Test”. If it is listed and checked then you are good to go, you can now close the window without modifying anything and skip to the next section.



NOTE: If VS17 is running then you must exit the program before continuing with the installation. Otherwise you will get a message that certain options may be running, and you must close them before installation can continue.

If Test Adapter for Google Test is not shown, then click on the “Individual components” option shown across the top of the window next to the “Workloads” option, then scroll down the selection list to the “Debugging and testing” section. Locate option for the “Test Adapter for Google Test” as shown below and click the check box to select the option. Now click “Modify” and the installer will install the test adapter. Once installation is complete, and you encountered no errors, or resolved any you encountered, you can close the “Visual Studio Installer” and start VS17.

Workloads Individual components Language packs

- ☒ Visual C++ compilers and libraries for ARM64
- ☒ Visual C++ runtime for UWP
- ☒ Visual C++ tools for CMake
- ☐ Windows Universal CRT SDK
- ☐ Windows XP support for C++

Debugging and testing

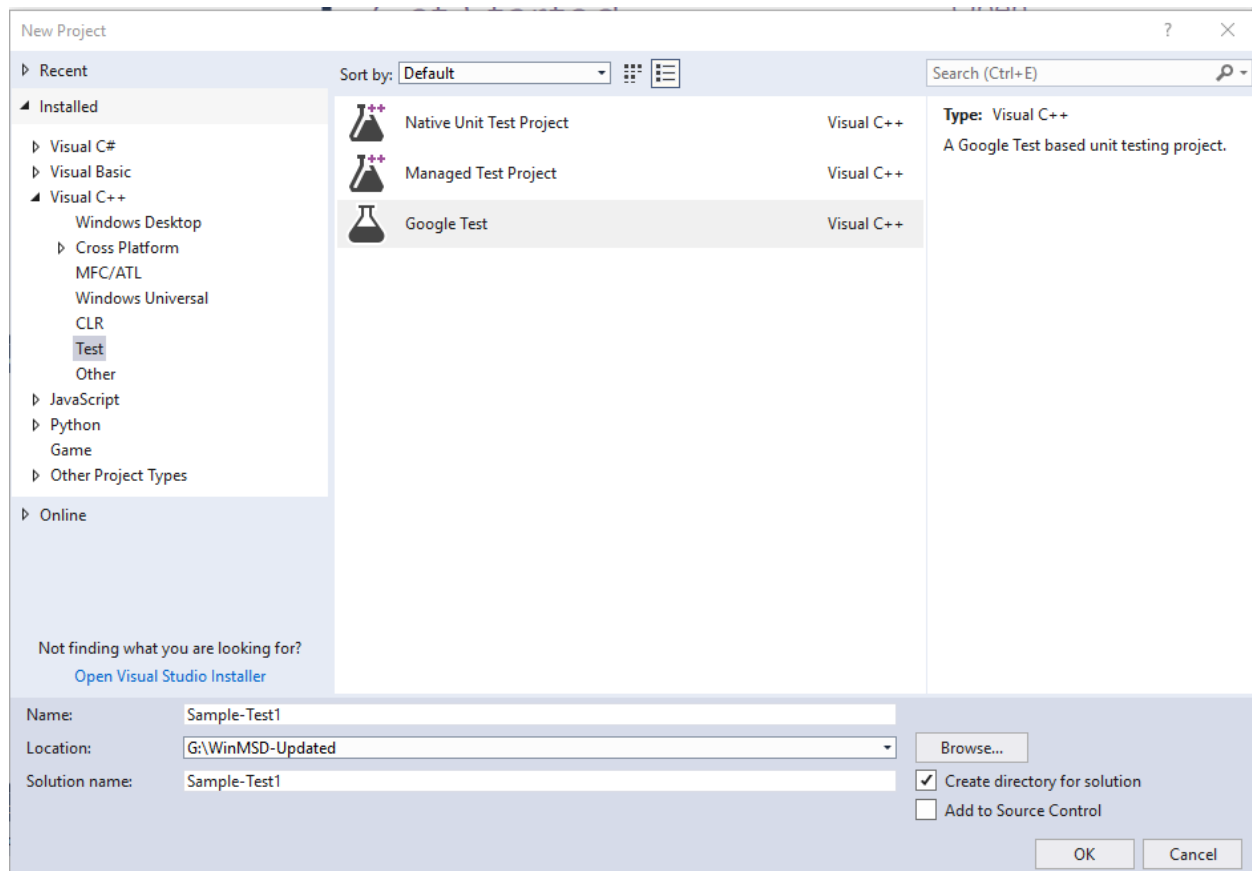
- ☒ .NET profiling tools
- ☒ C++ profiling tools
- ☒ JavaScript diagnostics
- ☒ Just-In-Time debugger
- ☐ Test Adapter for Boost.Test
- ☒ Test Adapter for Google Test
- ☐ Testing tools core features

Development activities

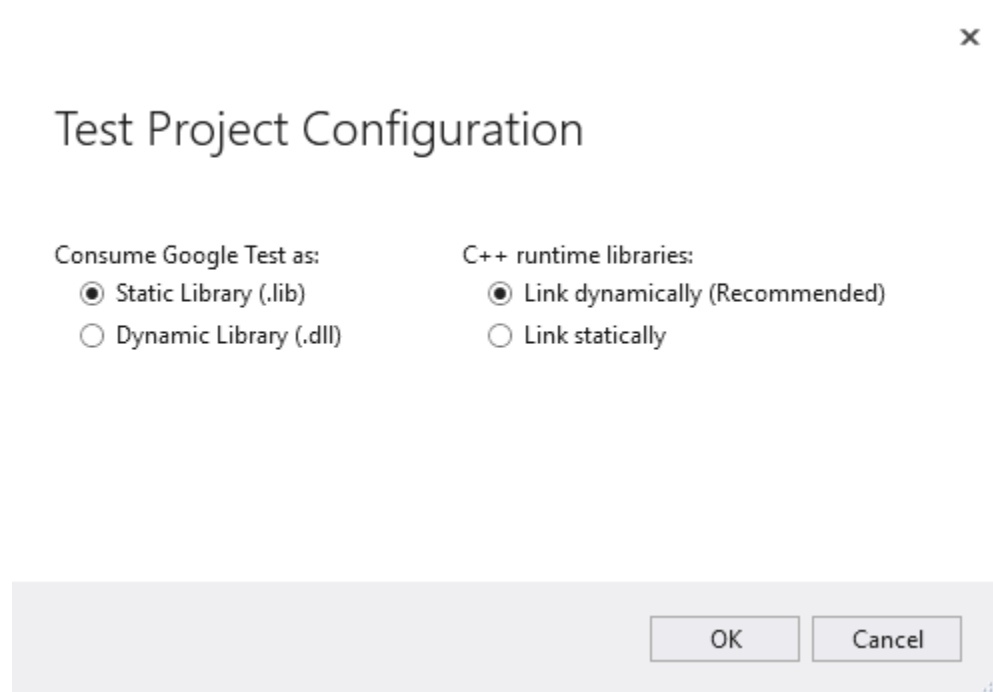
- ☐ ASP.NET and web development tools

Creating a Google Test Project in Visual Studio 2017

Start VS17. There are several ways to create a new Google Test. If you have a start page that is show, look under the “New Project” section for “Google Test C++”. If that option is not shown, then select “File, New, Project” from the file pulldown menu. If not already selected in the “New Project” window, then select “Test” under the “Visual C++” option and select the “Google Test” project. Your screen should look something like the following:

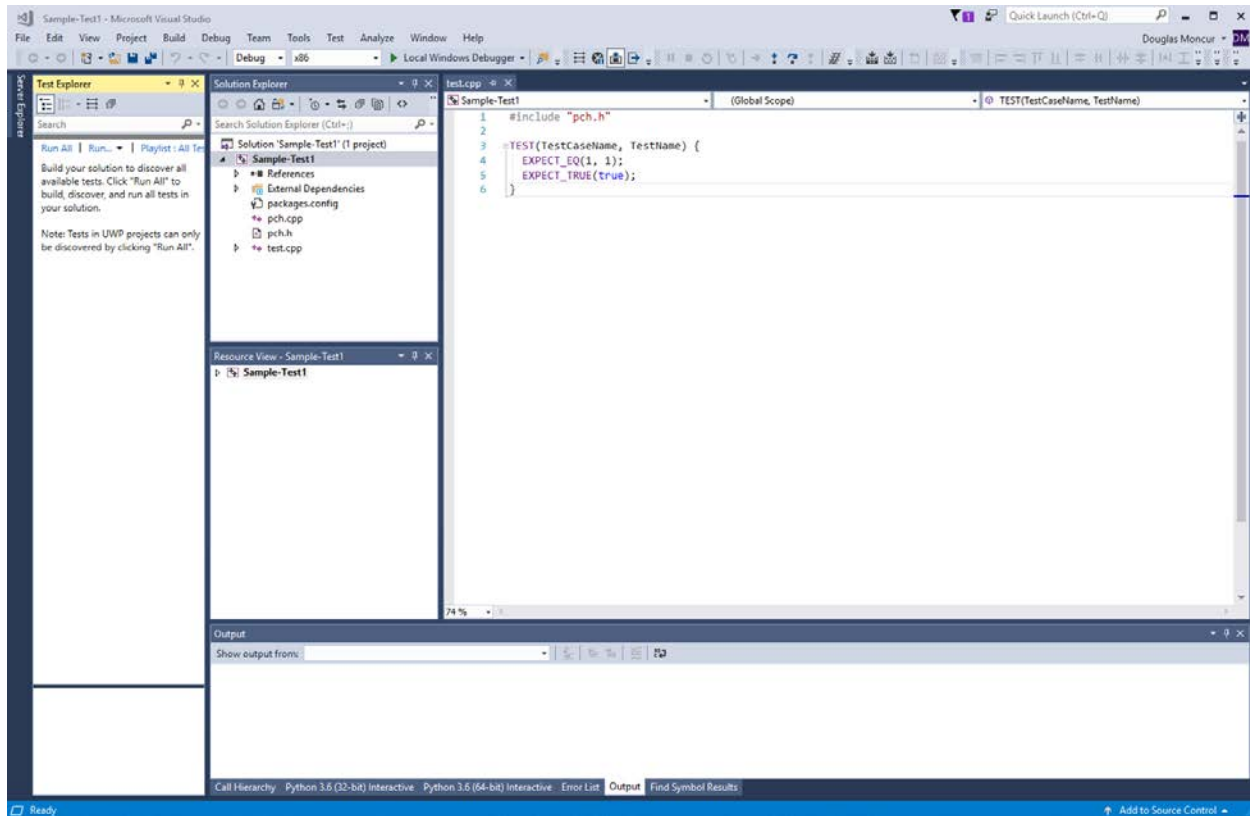


Select the location where you want your project created and enter the Name and Solution name as desired. This document will use the existing names as shown. Once you are satisfied with the names and location, you can use the default checkbox settings to the right or modify as required. Default settings will be used here. Click Ok and another window called “Test Project Configuration” will pop up. This has several options available. For this example, ensure that “Static Library (.lib)” and “Link dynamically (Recommended)” are selected and then click “OK”.

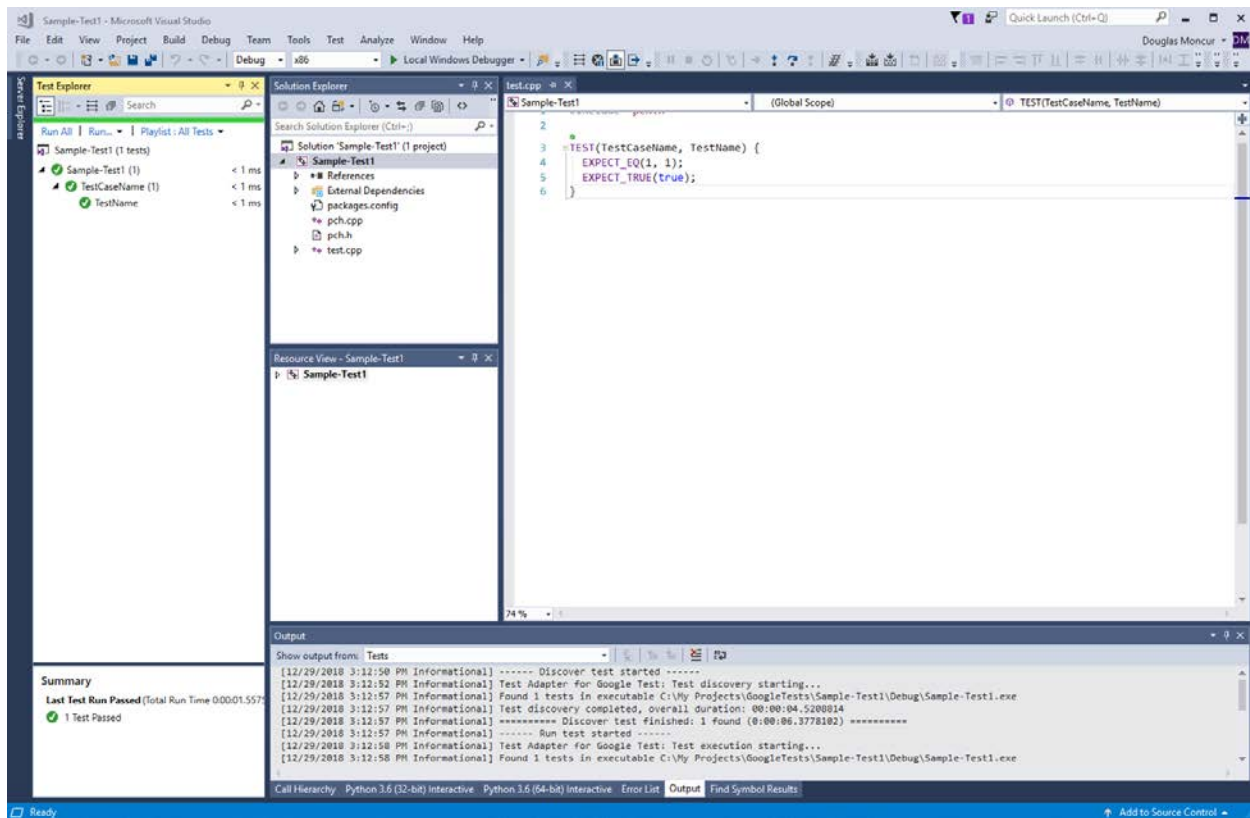


It is recommended that you open the “Test Explorer” window as one of your windows within VS17. To do so, go to the “Test” menu option and from the pulldown menu select “Windows” and then “Test Explorer” (Ctrl-E, T). This will show you the status of all tests that you run plus give you the options to run all tests or individual tests.

In the “Solution Explorer” window you should see the solution created for your first google test. You should also have a sample piece of code in the “test.cpp” file. It should all look something like this, depending on the windows you selected to have open.



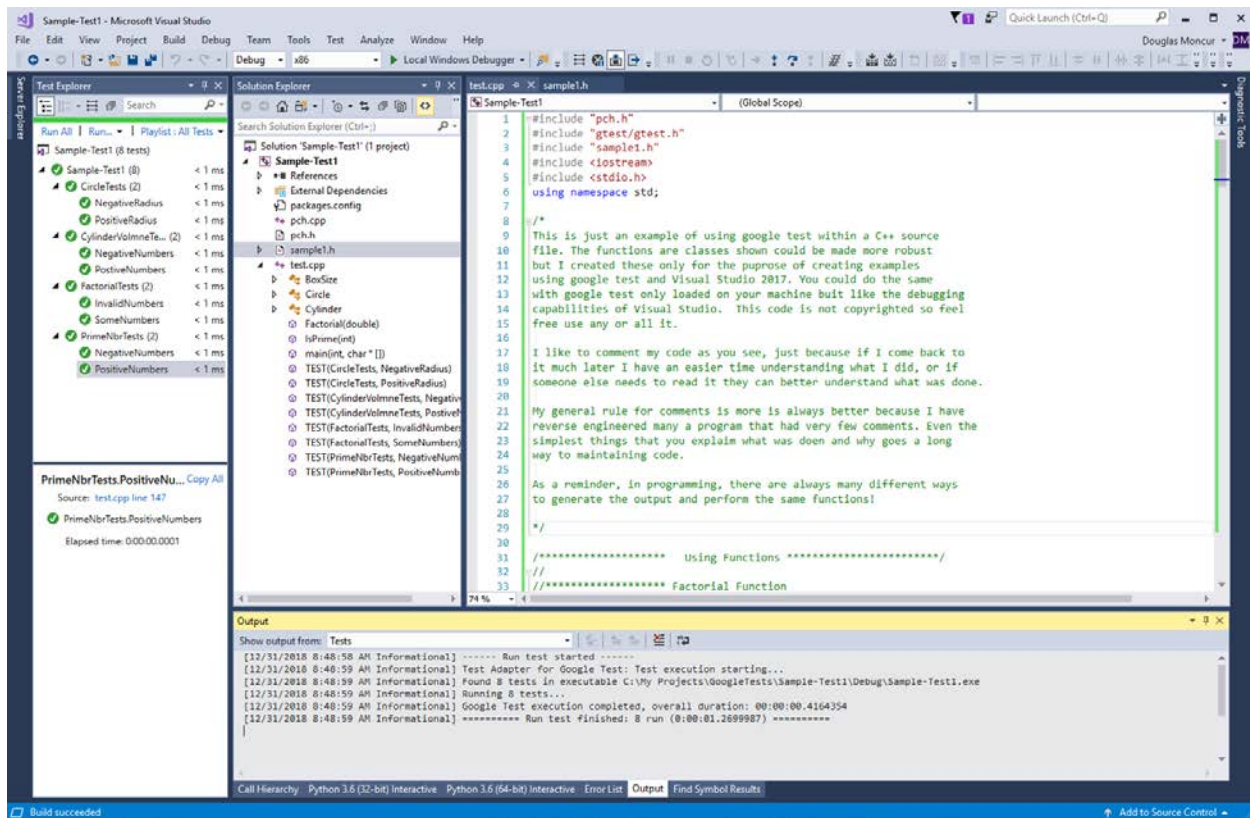
If you click “Run All” in the “Test Explorer” window, the project will compile and run the sample test and produce the output in the “Test Explorer” window. If you drilldown on each item you will see the “TestCaseName” and “TestName”. In this case, if nothing was modified it should have run successfully without errors. You can also view output in the “Output” window if that window is open. See the example below.



This completes the installation and setup of “google test” in VS17, and the creation of an initial project for your first google test without coding anything. The next section will go into creating a more useful test and explaining some of the google test commands as they are used. These examples will be built using this initial sample test project solution.

Creating an Application and Running Google Test

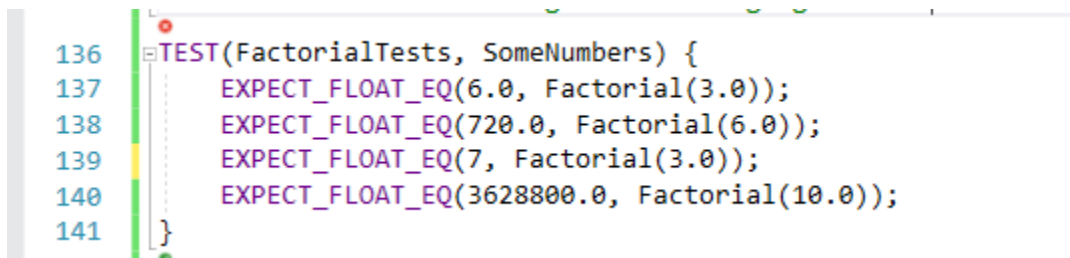
From the application created above in VS17, which was called “Sample_Test1”, a couple of structures and a few classes was created along with the main function. The small code shown above was removed as it was not needed, and the application and test cases were created. The complete listing of the application and test code is included in the appendix. A description of each area of VS17 is provided here to make sure the purpose of each area is understood, as someone may be new to VS17 or Visual Studio in general.



Test Explorer Window

The left most window, “Test Explorer” is used by the google test portion of the application. This window provides the status of each test case and unit test that was run. You can run only the tests by clicking on the “Run All” or from the “Run” pull down menu to select an individual test. The bottom portion with show you the results and any errors you encountered in running your test.

One test was modified to create an error as shown below. When you select (highlight) the unit test that failed the bottom portion of the “Test Explorer” window shows what was expected and what was received. In this case the test expected a value of 7 but got a value of 6 from the “FactorialsTests,SomeNumbers” test case. Double clicking on the failed test will take you to the failed testcase, but not the unit test line that failed. In this case, on line 139, there was a 7 that was entered as the expected result but should have been 6 as the factorial of 3 is 6. Notice also the red dot indicating a failure condition within the testcase.



Test Explorer

Search

Run All | Run... | Playlist : All Tests

Sample-Test1 (8 tests) 1 failed

Sample-Test1 (8) < 1 ms

CircleTests (2) < 1 ms

NegativeRadius < 1 ms

PositiveRadius < 1 ms

CylinderVolmneTests (2) < 1 ms

NegativeNumbers < 1 ms

PostiveNumbers < 1 ms

FactorialTests (2) < 1 ms

InvalidNumbers < 1 ms

SomeNumbers < 1 ms

PrimeNbrTests (2) < 1 ms

NegativeNumbers < 1 ms

PositiveNumbers < 1 ms

FactorialTests.SomeNumbers

Copy All

Source: [test.cpp line 136](#)

FactorialTests.SomeNumbers

Message:

Expected: 7

To be equal to: Factorial(3.0)

Which is: 6

Elapsed time: 0:00:00.0001

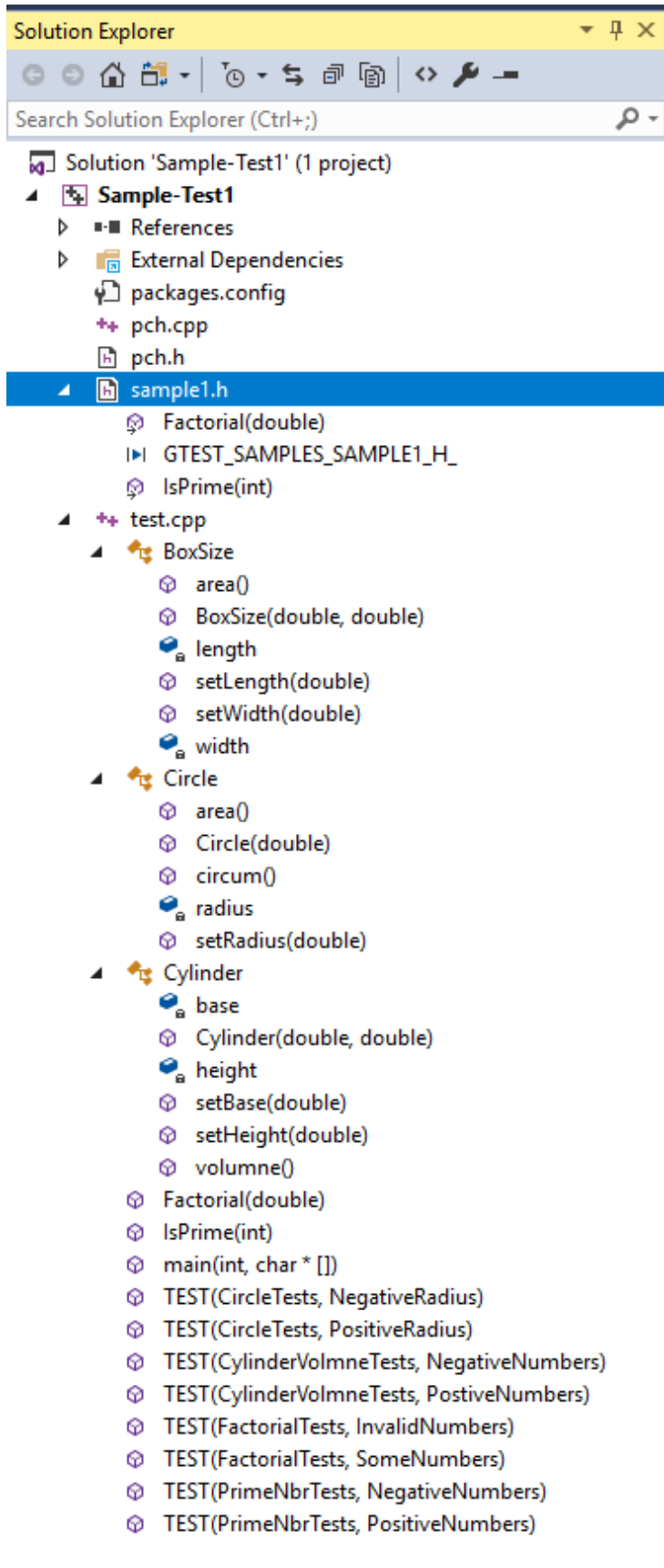
Stack Trace:

[test.cpp:139](#)

Solution Explorer

If you are familiar with Visual Studio, then the “Solution Explorer” window should be very familiar.

Notice that in this test solution it also shows the testscases that have been implemented, along with the classes and prototypes of the functions in the “.h” file.



Test.cpp file description

Again, there are many different ways to do the include files. This is just one, and some of it was part of the original project solution when VS17 created the google test project. All that was added was what was needed. There are many comments in the code and it should be easy to follow. It is not the intent of this document to describe the google test commands etc. There are articles in GitHub, and other google search's that can be of help in explaining google test commands. Here are a few links that might be useful.

<https://www.ibm.com/developerworks/aix/library/au-googletestingframework.html>

<https://github.com/google/googletest/tree/master/googletest/docs>

<https://docs.microsoft.com/en-us/visualstudio/test/how-to-use-google-test-for-cpp?view=vs-2017>

Summary

This example handles google test code with embedded application code. There are few ways to exclude it when you want the final build not to be bloated by the test code. One would be to put a conditional compile time flag around the test code. You probably want to use the same flag across all test files so nothing is forgotten.

The next project will focus on how google test can test existing applications using the external application files. This would allow developers to develop the code separate from the test code. Once this is figured it out, it will get posted as an update.

Hope this information was useful.

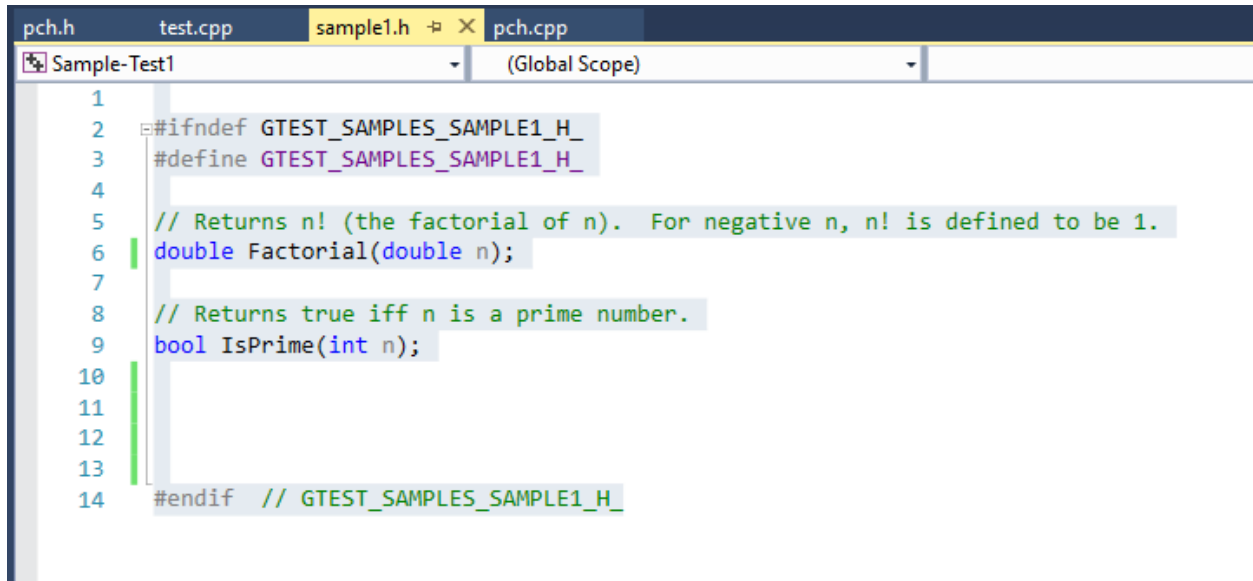
Appendix A – Acknowledgements and References

Here are some good references for the beginners.

1. Visual Studio Professional 2017 is copyrighted by the Microsoft Corporation.
2. cplusplus Tutorials Classes (I) : <http://www.cplusplus.com/doc/tutorial/classes/>
3. GitHub googletest documents:
<https://github.com/google/googletest/tree/master/googletest/docs>
4. Googletest primer:
<https://github.com/google/googletest/blob/master/googletest/docs/primer.md>

Appendix B – Source Code

Below is a pdf copy of the source code. It does not include the “.h” file which is shown below. This “.h” file was created by the project solution and the two prototype functions were added.



```
1
2 #ifndef GTEST_SAMPLES_SAMPLE1_H_
3 #define GTEST_SAMPLES_SAMPLE1_H_
4
5 // Returns n! (the factorial of n). For negative n, n! is defined to be 1.
6 double Factorial(double n);
7
8 // Returns true iff n is a prime number.
9 bool IsPrime(int n);
10
11
12
13
14 #endif // GTEST_SAMPLES_SAMPLE1_H_
```

Google Test Source file.

```
1  #include "pch.h"
2  // #include "gtest/gtest.h" // this is included in the pch.h file.
3  #include "sample1.h"
4  #include <iostream>
5  #include <stdio.h>
6  using namespace std;
7
8  /*
9   This is just an example of using google test within a C++ source
10  file. The functions are classes shown could be made more robust
11  but I created these only for the puprose of creating examples
12  using google test and Visual Studio 2017. You could do the same
13  with google test only loaded on your machine buit like the debugging
14  capabilities of Visual Studio. This code is not copyrighted so feel
15  free use any or all it.
16
17  I like to comment my code as you see, just because if I come back to
18  it much later I have an easier time understanding what I did, or if
19  someone else needs to read it they can better understand what was done.
20
21  My general rule for comments is more is always better because I have
22  reverse engineered many a program that had very few comments. Even the
23  simplest things that you explain what was doen and why goes a long
24  way to maintaining code.
25
26  As a reminder, in programming, there are always many different ways
27  to generate the output and perform the same functions!
28
29  */
30
31  /***** Using Functions *****/
32  //
33  /***** Factorial Function
34  //
35  // Returns nbr! (the factorial of nbr).
36  // For all negatives and non-facotorials, result returned as a 1.
37  // Note this could be put into class or made more robust but
38  // this is just an example to use for testing.
39  double Factorial(double nbr) {
40      double result = 1;
41      for (double ndx = 1; ndx <= nbr; ndx++) {
42          result *= ndx;
43      }
44      return result;
45  }
46
47  //
48  /***** Prime Number
49  //
50  // This function returns true if the number passed in is a prime number.
51  // Note this could be put into class but this is just an example.
52  bool IsPrime(int nbr) {
```

```
53     if (nbr <= 1) return false;           // No can do if a number is less than 1
54     if (nbr % 2 == 0) return nbr == 2;    // If and even number divisible by 2, not a
        a potential prime
55     // Now, we know that nbr is odd and nbr >= 3.
56     // Next action is to try and divide nbr by every odd number
57     // Try to divide nbr by every odd number oddNbr, starting from 3
58     for (int oddNbr = 3; ; oddNbr += 2) {
59         if (oddNbr > nbr / oddNbr) break; // do odd numbers only up to the sq.
            root of nbr
60
61         // the oddNbr is now <= nbr/oddNbr < nbr.
62         // and if nbr is divisible by oddNbr at this point, the nbr is not a
            prime
63         if (nbr % oddNbr == 0) return false;
64     }
65     // at this point nbr is a prime number so let them know.
66     return true;
67 }
68
69 //
70 /***** Using Classes *****/
71 //
72 /***** Circle Class
73 //
74 // This Class is used to calculate the circumference or area of a circle
75 // given the radius. The class is initialized with a radius value when
76 // instantiated (which could be done in different ways).
77 class Circle {
78     double radius;
79 public:
80
81     //Constructor with a value to initialize the radius
82     Circle(double r) { radius = r; }
83
84     // Public methods
85     double circum() { return 2 * 3.14159265* radius; }
86     double area() { return 3.14159265 * (radius * radius); }
87     void setRadius(double rad) { radius = rad; }
88
89 };
90
91 //
92 /***** Class Cylinder
93 //
94 // This class is used to calculate the volume of a cylinder
95 // given the radius and height variables
96 class Cylinder {
97     Circle base; // create a circle so we have a base to work from.
98     double height;
99 public:
100     // Constructor and initialization
101     Cylinder(double r, double h) : base(r), height(h) {}
```

```
102
103     // public methods
104     double volume() { return base.area() * height; }
105     void setHeight(double h) { height = h;}
106     void setBase(double b) { base = b;}
107 };
108
109 //
110 //***** Class BoxSize
111 //
112 // Initialized with the length and width it calculates
113 // the area of the box.
114 class BoxSize {
115     double length;
116     double width;
117 public:
118     // Constructor.
119     BoxSize(double l, double w) : length(l), width(w) {}
120
121     // public methods.
122     // note: You could adjust for negative numbers by using abs().
123     double area() { return (length*width); }
124     void setLength(double len) { length = len; }
125     void setWidth(double wid) { width = wid; }
126 };
127
128 /*****
129 /***** Test Cases *****/
130 /*****
131 // These are simple test cases and unit tests that test the
132 // various functions and class members. Kept them simple
133 // for example purposes only. There are probably different
134 // ways and many enhancements but I just wanted to show
135 // structure and how things worked in google test.
136 TEST(FactorialTests, SomeNumbers) {
137     EXPECT_FLOAT_EQ(6.0, Factorial(3.0));
138     EXPECT_FLOAT_EQ(720.0, Factorial(6.0));
139     EXPECT_FLOAT_EQ(6, Factorial(3.0));
140     EXPECT_FLOAT_EQ(3628800.0, Factorial(10.0));
141 }
142 TEST(FactorialTests, InvalidNumbers) {
143     EXPECT_FLOAT_EQ(1.0, Factorial(0.0)); // invalid number returns 1
144     EXPECT_FLOAT_EQ(1.0, Factorial(-3.0)); // invalid number returns 1
145 }
146 // Just an example of some primes and not so prime numbers
147 TEST(PrimeNbrTests, PositiveNumbers) {
148     EXPECT_EQ(true, IsPrime(2));
149     EXPECT_EQ(true, IsPrime(3));
150     EXPECT_EQ(true, IsPrime(5));
151     EXPECT_EQ(true, IsPrime(47));
152     EXPECT_EQ(true, IsPrime(167));
153 //Now non-prime numbers
```



```
154     EXPECT_EQ(false, IsPrime(0));
155     EXPECT_EQ(false, IsPrime(4));
156     EXPECT_EQ(false, IsPrime(18));
157     EXPECT_EQ(false, IsPrime(54));
158     EXPECT_EQ(false, IsPrime(141));
159
160 }
161 // Don't need to do all these but did some just
162 // as an example.
163 TEST(PrimeNbrTests, NegativeNumbers) {
164     EXPECT_EQ(false, IsPrime(-1));
165     EXPECT_EQ(false, IsPrime(-3));
166     EXPECT_EQ(false, IsPrime(-5));
167     EXPECT_EQ(false, IsPrime(-47));
168     EXPECT_EQ(false, IsPrime(-167));
169     //Now non-prime numbers
170     EXPECT_EQ(false, IsPrime(-0));
171     EXPECT_EQ(false, IsPrime(-4));
172     EXPECT_EQ(false, IsPrime(-18));
173     EXPECT_EQ(false, IsPrime(-54));
174     EXPECT_EQ(false, IsPrime(-141));
175 }
176 TEST(CircleTests, PositiveRadius) {
177     Circle TestCircle(0.0);
178     EXPECT_FLOAT_EQ(0.0, TestCircle.circum());
179     TestCircle.setRadius(6.0);
180     EXPECT_FLOAT_EQ(37.6991, TestCircle.circum());
181     TestCircle.setRadius(100.0);
182     EXPECT_FLOAT_EQ(628.3185, TestCircle.circum());
183
184 }
185 TEST(CircleTests, NegativeRadius) {
186     Circle TestCircle(0.0);
187     EXPECT_FLOAT_EQ(0.0, TestCircle.circum());
188     TestCircle.setRadius(-6.0);
189     EXPECT_FLOAT_EQ(-37.6991, TestCircle.circum());
190     TestCircle.setRadius(-100.0);
191     EXPECT_FLOAT_EQ(-628.3185, TestCircle.circum());
192 }
193
194 TEST(CylinderVolmneTests, PostiveNumbers) {
195     Cylinder myCylinder(0, 0);
196     EXPECT_FLOAT_EQ(0.0, myCylinder.volumne());
197     myCylinder.setBase(10.0);
198     EXPECT_FLOAT_EQ(0.0, myCylinder.volumne());
199     myCylinder.setHeight(10.0);
200     EXPECT_FLOAT_EQ(3141.5925, myCylinder.volumne());
201     myCylinder.setBase(32.5);
202     myCylinder.setHeight(12.9);
203     EXPECT_FLOAT_EQ(42806.164, myCylinder.volumne());
204 }
205
```

```

206 TEST(CylinderVolumneTests, NegativeNumbers) {
207     Cylinder myCylinder(-0, -0);
208     EXPECT_FLOAT_EQ(0.0, myCylinder.volumne());
209     myCylinder.setBase(-10.0);
210     EXPECT_FLOAT_EQ(0.0, myCylinder.volumne());
211     myCylinder.setHeight(-10.0);
212     EXPECT_FLOAT_EQ(-3141.5925, myCylinder.volumne());
213     myCylinder.setBase(-32.5);
214     myCylinder.setHeight(-12.9);
215     EXPECT_FLOAT_EQ(-42806.164, myCylinder.volumne());
216 }
217
218 /***** Main Function *****/
219 int main(int argc, char* argv[]) {
220
221     // Some variables to try different numbers with
222     double Factorialnbr = 3.0;
223     int primeNbr = 7;
224
225     // Class creatations and initializations
226     BoxSize myBox(10, -20.4); // Create the box size
227     Circle myCircle(35.0); // Create and Initialize using constructor.
228     myCircle.setRadius(6.0); // example to reset radius if needed.
229     Cylinder myCylinder(20, 45); // Create and Initialize myCylinder
230     // Now before all the tests are run below,
231     // show results using printf here.
232     printf("\n\n*****\n\n");
233     printf("1. MyBox area results %4.3f \n", myBox.area());
234     printf("2. MyCircle circumference %4.3f \n", myCircle.circum());
235     printf("3. MyCylinder Volumne %4.3f \n", myCylinder.volumne());
236     if (IsPrime(primeNbr) == true)
237         printf("4. This number is a prime number! %i \n", primeNbr);
238     else
239         printf("4. This number is NOT a prime number! %i \n", primeNbr);
240     printf("5. Factorilal of %4.2f is %f \n", Factorialnbr, Factorial
241           (Factorialnbr));
242     printf("\n\n*****\n\n");
243
244     // And as an example, you can use the cout i/o function as well
245     // to see the same thing.
246     cout << "6. MyBox area is: " << myBox.area() << '\n';
247     cout << "7. MyCircle circumference is: " << myCircle.circum() << '\n';
248     cout << "8. myCylinder volumne is: " << myCylinder.volumne() << '\n';
249     cout << "9. Factorial nbr is: " << Factorial(Factorialnbr) << '\n';
250
251     printf("*****\n\n");
252
253     // Now we are going to run all the tests. These too will be shown shown in
254     // the output screen when you run the program as an application and
255     // not running the tests only in the "Test Expolorer" window.
256     // If you run the tests only by using the "Run All" option
257     // in the Visual Studio Test Explorer window then the application

```

```
257     // items above are not run. This then allows you bypass all the
258     // executable code above in main.
259     testing::InitGoogleTest(&argc, argv);    // initialize test environment
260     return RUN_ALL_TESTS();                  // and run all the tests.
261
262
263     //hold the command window output display until a character is entered.
264     // char a;
265     //cin >> a;
266
267     return 0;
268 }
269
```