Software Testing

Unit Testing

Unit Testing

- unit testing is the process of testing the code in small units.
- Usually these units correspond to functions
- Each unit test feeds the function under test specific values and then gathers a return value and compares it to expected output.
- If the value produced by the function under test is identical to that that is expected, then the test has passed.

How Unit Tests Work

- We usually use a unit test framework which
  - Allows us to set up the test,
  - Call the function being tested and store the result,
  - Compare the result to the expected result
  - Use an assertion to throw an exception if the test fails
  - Tear down the test

Test Math Functions

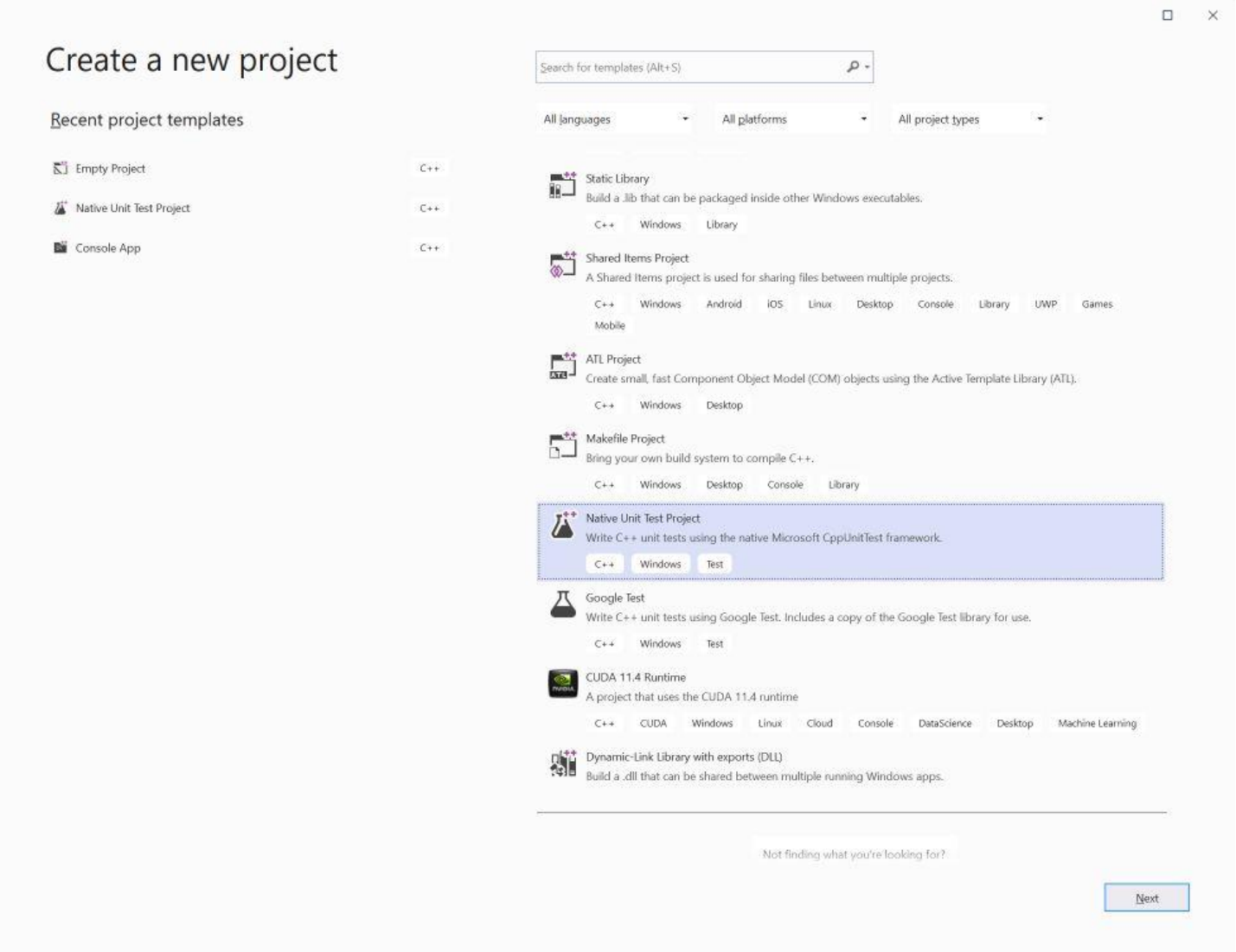
mathfuncs.h

```
#pragma once

#ifndef MATHFUNCS_H

#define MATHFUNCS_H



double square(double n);

double cube(double n);



#endif
```

mathfuncs.c

```
#include "mathfuncs.h"

double square(double n)
{
return n * n;
}

double cube(double n)
{
return n * n * n;
}
```

Create a Test Project



Write Test Code

```
#include "pch.h"
#include "CppUnitTest.h"
#include "mathfuncs_r.h"

using namespace
Microsoft::VisualStudio::CppUnitTestFramework;

namespace MathTestSuite
{
    TEST_CLASS(MathTest)
    {
    public:

        TEST_METHOD(SquareTest)
        {
            double d = square(8.0);
            Assert::AreEqual(64.0, d);
        }

        TEST_METHOD(CubeTest)
        {
            double d = cube(3.0);
            Assert::AreEqual(27.0, d);
        }
    };
```

```
TEST_CLASS(MathIntegrationTest)
    {
    public:

        TEST_METHOD(AdditionTest)
        {
            double d = square(8.0);
            double d1 = cube(3.0);
            Assert::AreEqual(91.0, d + d1);
        }
    };
}
```

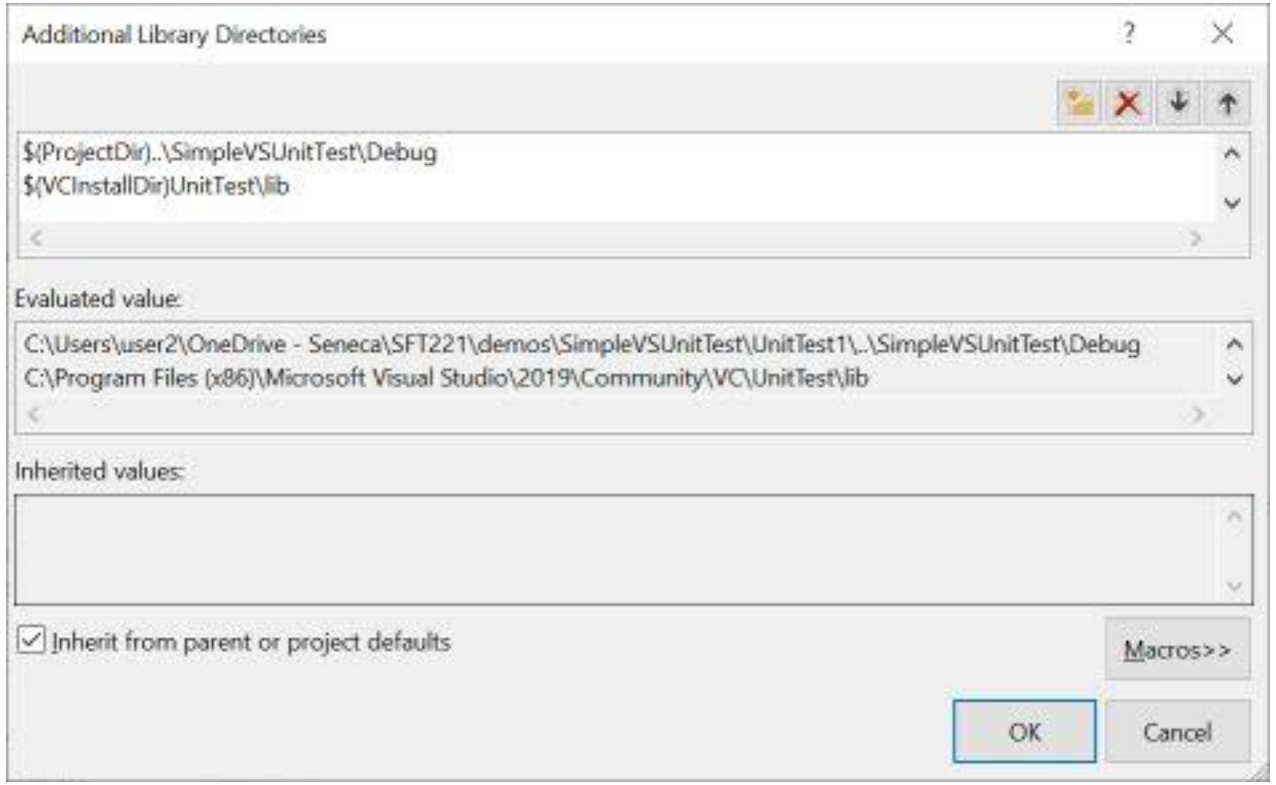
Access C Include File

```
#pragma once
#ifndef MATHFUNCS_R_H
#define MATHFUNCS_R_H

extern "C"
{
#include <mathfuncs.h>
}

#endif
```

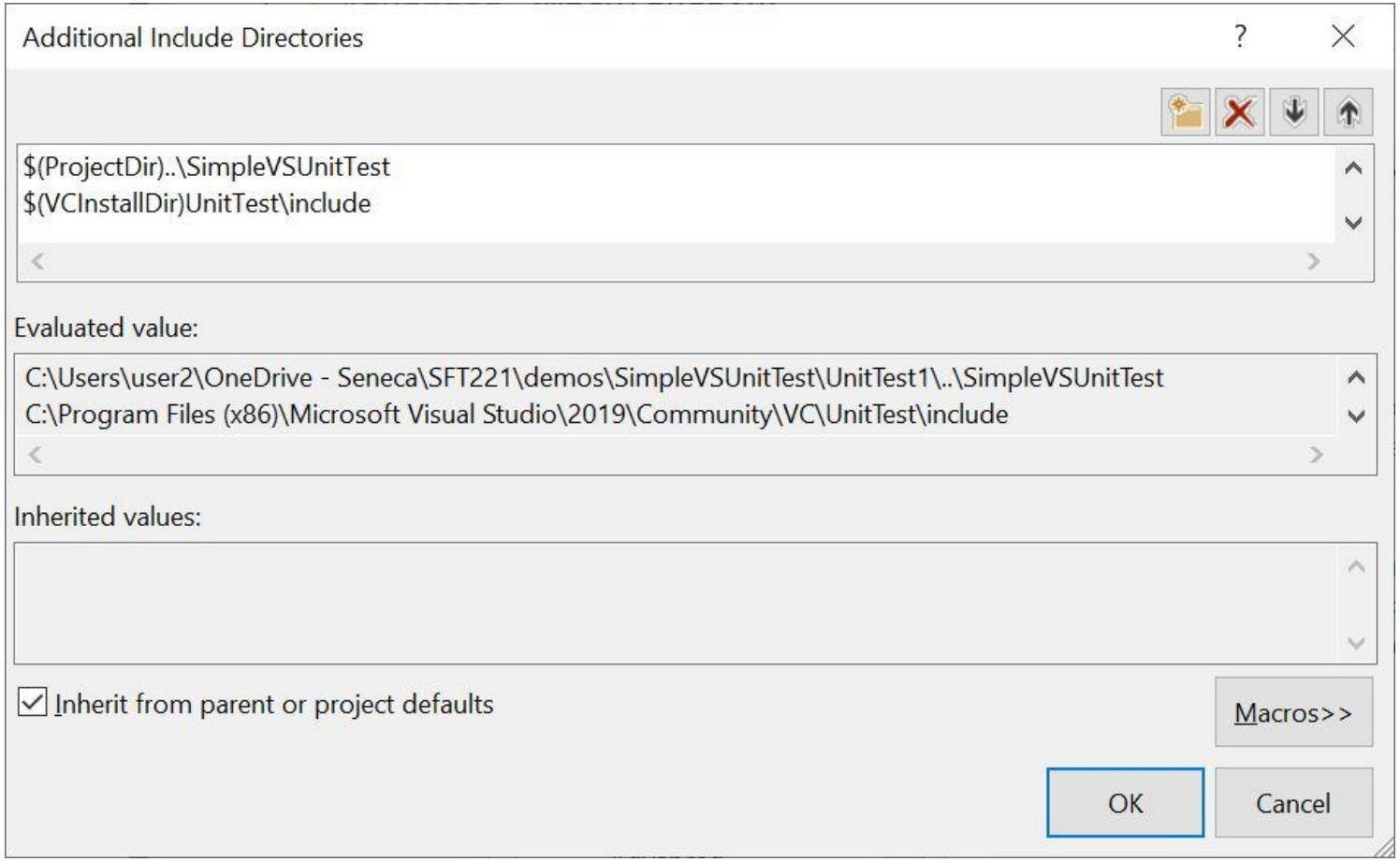
Assertions

Method	Description
AreEqual(v1, v2 [, "error message"])	Compares v1 to v2 and throws an exception if they are not equal. If they are not equal, the optional error string will be displayed.
AreNotEqual(v1, v2 [, "error message"])	Compares v1 to v2 and throws an exception if they are equal. If they are equal, the optional error string will be displayed.
IsTrue(b1 [, "error message"])	If the Boolean b1 is not true it throws an exception. If not true, the optional error string will be displayed.
IsFalse(b1 [, "error message"])	If the Boolean b1 is not false it throws an exception. If not false, the optional error string will be displayed.
Fail(["error message"])	This causes the test to fail and throws an exception. If called, the optional error string will be displayed.

Setup Library Path

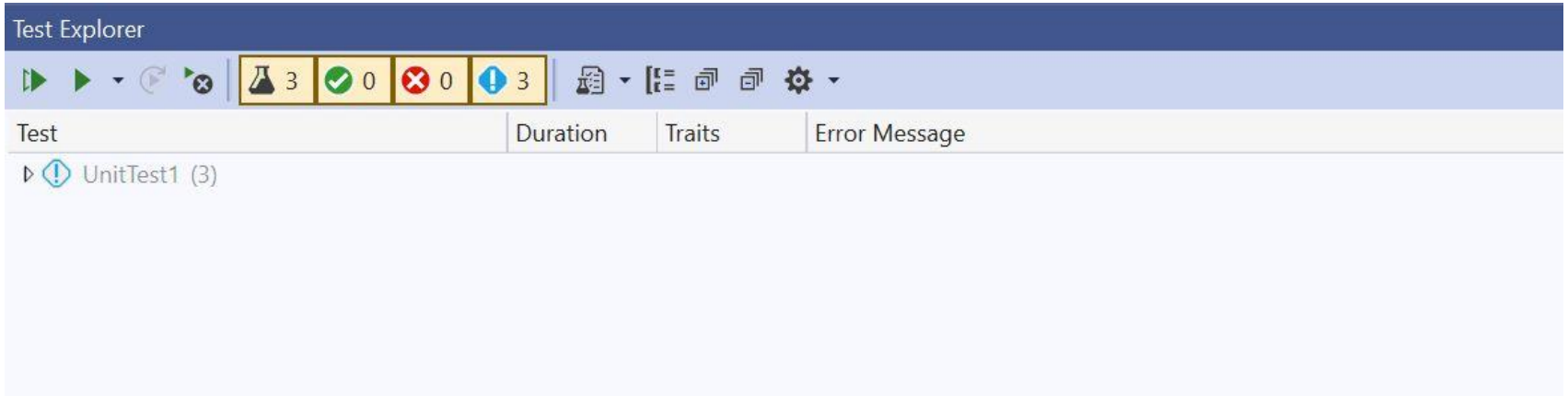


Setup Include Path

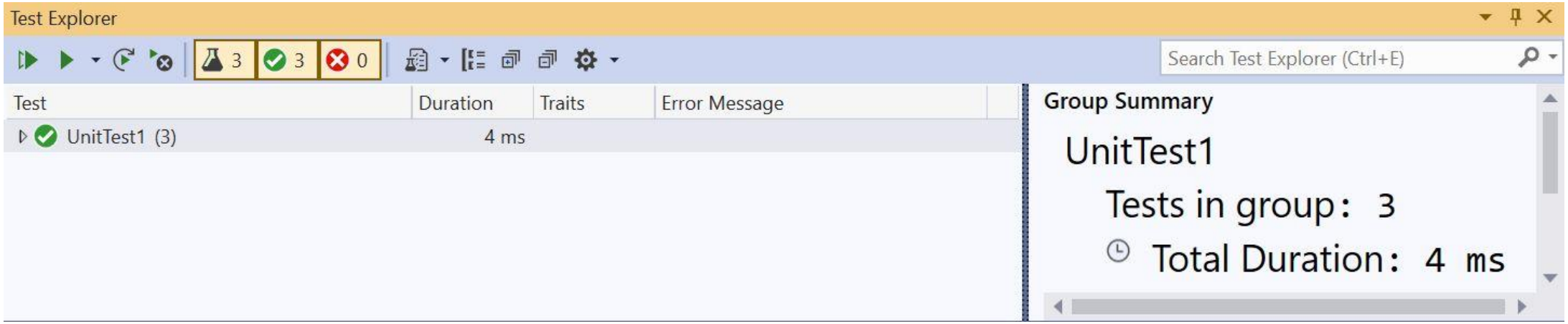


Running Tests in Visual Studio

- Go to the **Test** menu at the top and select **Test Explorer**.
- This will display a new window to control the tests, as shown below.



Test Results



Setup and Teardown Before/After Tests

```
TEST_MODULE_INITIALIZE(ModuleInitialize)
{
Logger::WriteMessage("In Module Initialize");
}

TEST_MODULE_CLEANUP(ModuleCleanup)
{
Logger::WriteMessage("In Module Cleanup");
}
```

Setup and Teaddown for Each Test

```
TEST_CLASS_INITIALIZE(ClassInitialize)
{
Logger::WriteMessage("In Class Initialize");
}

TEST_CLASS_CLEANUP(ClassCleanup)
{
Logger::WriteMessage("In Class Cleanup");
}
```

Test from Command Line

```
vstest.console.exe UnitTest1.dll
Microsoft (R) Test Execution Command Line Tool Version 16.11.0
Copyright (c) Microsoft Corporation. All rights reserved.

Starting test execution, please wait...
A total of 1 test files matched the specified pattern.
In Module Initialize
In Integration test
In Class Initialize
In Square test
In Cube test
In Class Cleanup
In Module Cleanup
Passed AdditionTest [< 1 ms]
Passed SquareTest [< 1 ms]
Passed CubeTest [< 1 ms]

Test Run Successful.
Total tests: 3
Passed: 3
Total time: 0.2394 Seconds
```

Test Coverage

- Test coverage measures which lines of code were tested
- The goal is to find the code which is not tested
- Tools are available to count how many times each line was executed
- More details are in the class notes.