**Lab 3: Introduction to Google Test (gtest)**

# Overview

In this lab you will:

1. Complete a program by writing a function
2. Test your code using gtest unit tests
3. Write simple gtest unit tests

## Set-up

To begin, you must copy the lab files to your working directory:

cp /afs/umbc.edu/users/c/m/cmarron/pub/cmsc202/lab3.zip .

Then decompress the file

unzip lab3.zip

You should now have three files in your directory, lab3.cpp, lab3.h, and Makefile:

* lab3.cpp is the implementation file. It contains main() and a function ChooseTwo().
* lab3.h is the interface file. It contains protototypes for two functions, ChooseTwo() and Factorial().
* Makefile is used to build the program lab3.

## Task 1

Note that the function Factorial() has not been implemented in lab3.cpp. Your first task is to write this function to complete the program. For positive arguments n ≤ 12, the function must return the product of all positive integers ≤ n, i.e. the factorial function you are familiar with from mathematics. For n = 0, the function should return 1, and for n < 0 or n > 12, the function should return -1 to indicate an invalid argument.

**Note:** you may not modify the interface file lab3.h. Implement the function Factorial() in lab3.cpp, after the main()function.

Once you have added the Factorial() function to the implementation file, save your work and run the make command to build the program.

# Testing with Google Test

Google Test (gtest) is a unit-testing framework for C++ programs. Unit tests are used to test individual units of source code, in this case, functions.

## Google Test Set-up

You will need to copy the gtest files for the lab to your working directory:

cp /afs/umbc.edu/users/c/m/cmarron/pub/cmsc202/lab3\_test.zip .

Then decompress the file:

unzip lab3\_test.zip

You should now have two additional files in your directory: lab3\_test.cpp and Makefile\_test.

## Running the Tests

To build and run the tests, you simply run make with Makefile\_test:

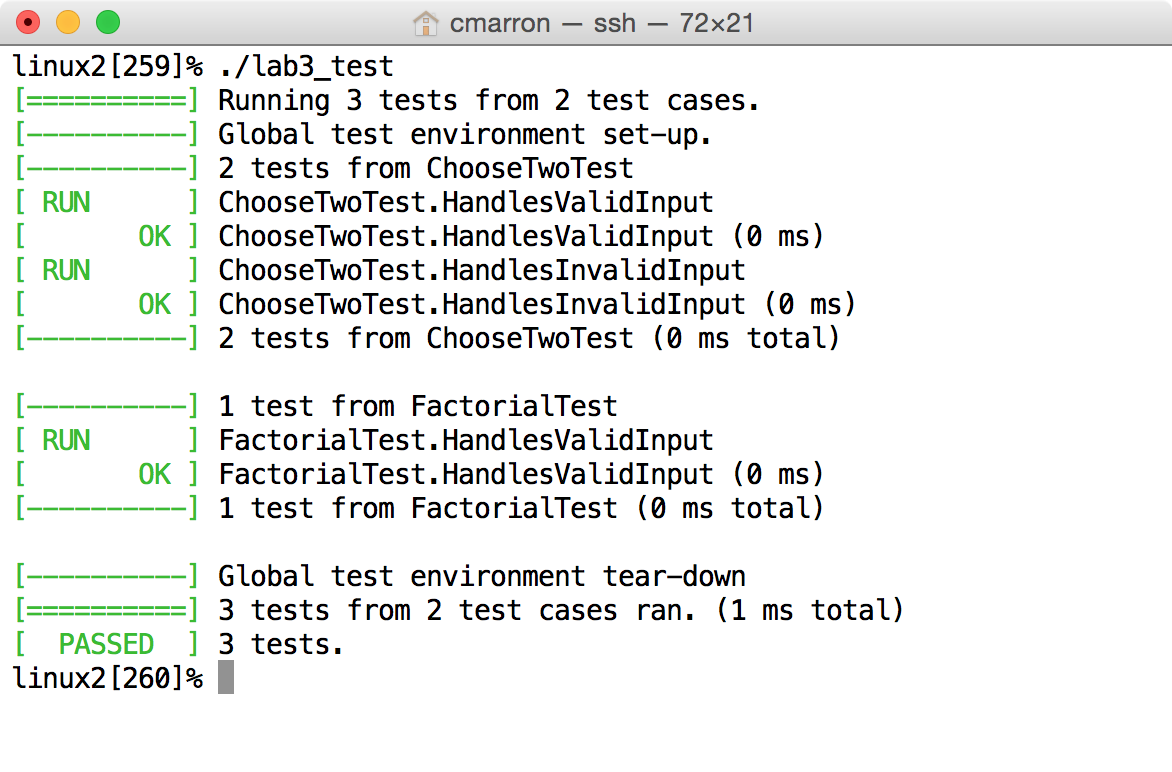
make -f Makefile\_test

You may see a number of warning messages, but that is okay — so long as there are no errors, the test program will build correctly. If the build is successful, you will now have an executable program named lab3\_test.

Run the test program:

./lab3\_test

You should see output similar to the following:



The program runs three tests; if you have implemented the Factorial() function correctly, they should all complete with status "OK." The three tests check that the functions are producing correct outputs for certain inputs:

1. ChooseTwoTest.HandlesValidInput checks that the ChooseTwo() function returns the correct value for several valid inputs.
2. ChooseTwoTest.HandlesInvalidInput checks that the ChooseTwo() function returns the correct value for invalid inputs.
3. FactorialTest.HandlesValidInput checks that the Factorial() function returns the correct value for several valid inputs.

Look at the file lab3\_test.cpp in a text editor and locate the the implementation of ChooseTwoTest.HandlesValidInput:

TEST(ChooseTwoTest, HandlesValidInput) {

EXPECT\_EQ(1, ChooseTwo(2)); // lower edge case

EXPECT\_EQ(45, ChooseTwo(10));

EXPECT\_EQ(190, ChooseTwo(20));

EXPECT\_EQ(536854528, ChooseTwo(32768)); // upper edge case

}

The arguments to TEST(), in this case ChooseTwoTest and HandlesValidInput are chosen by the user to help group and organize the tests: they appear as the test names in the output of the test program.

The EXPECT\_EQ() function is provided by gtest, and provides a simple way to check whether a function returns the correct value: the test passes if the two arguments are equal, and fails otherwise.

Although there are four tests in this function, two of them are special. The "lower edge case" checks that the function works properly with the smallest valid input, and the "upper edge case" checks that it works with the largest valid input. It is important to check edge cases, as it is easy to make mistakes on the edges: a bug can be introduced through a simple typo such as using < rather than ≤, or being off by one in an array index.

# Writing a Google Test

Scroll down in lab3\_test.cpp to locate the implementation ofFactorialTest.HandlesValidInput:

TEST(FactorialTest, HandlesValidInput) {

EXPECT\_EQ(1, Factorial(0)); // lower edge case

EXPECT\_EQ(1, Factorial(1));

EXPECT\_EQ(2, Factorial(2));

/\* TEST FOR Factorial(12) GOES HERE -- UPPER EDGE CASE \*/

}

## Task 2: Add a Test Case

Note that the implementation of Factorial.HandlesValidInput does not include a test for the upper edge case, n = 12. Your second task is to add the upper edge case test. The correct value of Factorial(12) is 479001600.

Once you have added the test, build and run the test program. If your program fails your test, determine whether it is the test or the Factorial() function that is incorrect.

## Task 3: Write a Google Test

Locate the tests that check that the ChooseTwo() function returns the correct value (-1) for invalid input. Using these tests as a model, write a new TEST() function that implements invalid input tests for your Factorial() function. Your TEST() function must check upper and lower edge cases.

**Note:** there is a comment in lab3\_test.cpp indicating where you should write the new TEST() function. It is essential that you not alter any of the code below the comment.

When your tests are ready, build and run the test program. If your program fails your tests, determine whether it is the tests or the Factorial() function that are incorrect.