[**Premise**](#_i6i5kshjvk1r) **1**

[**Specifications**](#_e7m1e6qmqfes) **2**

[**Starter Code**](#_e0aq95kq7khh) **4**

[**Questions**](#_7mzs2tewc01f) **5**

[**Tips and Tricks**](#_ca4v04gezw3i) **6**

[C++11](#_247rcn96dwes) 6

[What cases should I check?](#_jd98j693tqh5) 6

[**Work in terminal**](#_6buca4r410ij) **8**

[**Work in Eclipse**](#_oxgcjqk2eeku) **9**

[Introduction](#_fkq934zqlie) 9

[Background](#_lhl38y37mq74) 9

[Setup Procedures](#_y5gj2pecfoz2) 10

[Step 1: Download googletest.](#_px89gmewflag) 10

[Step 2: Extract gtest-1.7.0.](#_yrh6fieyftte) 10

[Step 3: Setting up working environment for googletest in Eclipse](#_51laamg2zxfl) 11

[Step 3.1 Create unit test project](#_gt0of8xs3e3a) 11

[Step 3.3 adding googletest library, source and test code.](#_ckaksc8t0wpm) 15

[1. adding Googletest library to folder gtest\_src.](#_3rnd46tbkq5b) 15

[2. adding source code.](#_ftag0cvn7rn3) 16

[3. Adding test code.](#_oeoj65weqbso) 18

[Step 5: running Google Test.](#_ltk0bxv3u0bm) 22

Warm-up Project

January*, 2017* *CS22001*

# Premise

To warm you up to C++, software construction, and the class as a whole, you will complete the following project in a team of 2 students.

You are a software engineer looking to revolutionize the social networking industry. You have big ideas for your social networking application, but before you can take your project to that level, you have to lay the groundwork for it.

Your social network will center around *communities* and *people*. Communities are the collective structure of the application, and people represent individuals that are participating in your social network. The basic functionality of your social network will be as follows:

* Person objects have a username, firstname, lastname, age, and tagline associated with them
* the above five variables can be obtained and set via function calls, either individually, or all at once
* Person objects have a *private* inbox that stores messages (represented by strings)
* any Person can send a message to any other Person, which adds it to the back of their inbox
* Person objects can read messages from the front of their inbox, which prints them and then discards them from the front of the inbox.
* Community objects will contain a collection of Person objects within them
* Community objects have a name
* a Person can be added to a Community (and thus become *members*)
* a Person with a particular username can be retrieved from a Community
* Community objects can be searched for members by firstname
* all members of a Community can be obtained via a list of usernames
* all usernames of a Community can be printed out
* Community objects can distribute a message to a subset of their users

# Specifications

There are a lot of design choices to make in order to implement the above functionality! In order to make grading easier for us and to impose some structure that might make the implementation easier as well, here are some guidelines:

You will have exactly two classes: a Community class and a Person class (with those exact names). You will have a header file and a cpp (implementation) file for each class. *You may not implement any functions in the header files!*

You will use a C++ standard library (stl) map object to store the collection of Person objects from within the Community object. This object must be *private*, and it will map usernames (represented by strings, which are also stl objects) to Person objects. You may find the following documentation useful:

<http://www.cplusplus.com/reference/map/map/>

<http://www.cplusplus.com/reference/string/string/>

Note that the most painless way to read C++ documentation is to click on member functions that you want to use and to look at the example code snippets they provide. Some of the information on the main reference page for a class in the C++ standard library is extremely detailed and superfluous in all except the most nuanced cases.

Given that you are using a stl map, accomplishing some of the above functionality will *only be possible* if you use some of the member functions for the map object as specified by the documentation! In some cases, it may be possible to avoid using them, but your implementation will consequently be much longer and less elegant. Please don't cause yourself the pain of re-implementing the functions that are already part of the stl…unless, of course, you want the practice!

Do not create a class for messages. Messages should simply be stl strings.

A Person's username can have a maximum of 64 characters and must have alphanumeric characters only. Also, it can not start with a digit. Finally, it cannot be the empty string.

A Person's firstname and lastname can have a maximum of 64 characters, and must contain only alphabetic characters.

A Person's age must be between 0 (inclusive) and 128 (exclusive).

A person’s gender must be either “m” or “f”.

A Person's tagline is limited to 512 characters.

If a Person tries to read a message from their inbox but they have no messages, a print statement should be used to indicate this and the inbox should remain unchanged.

A Community's name can have a maximum of 128 characters and must have alphanumeric characters only. Also, it can not start with a digit. Finally, it cannot be the empty string.

If the user of your program tries to make a Person or Community object that violates the above specifications, a "null" instance of the class should be returned, defined as one with empty strings for all string fields, 0 for the age field, and a map object with no key/value pairs in it.

Both classes' default constructors should create "null" versions of those classes.

If the user of your program tries to *modify* a Person or Community object in a way that violates the above specifications, the object should remain unchanged and the function should return false.

If the user tries to get a Person by username from the Community and no such Person exists, the "null" Person (as described above) should be returned.

The user can query a list of Person by an age range, here the ages can be larger than 128 or smaller than 0, but the returned Person objects shall fall into the age range. Both the lower and upper bounds are inclusive.

The set\_info function need to set the user information by calling the each function 1) set\_username, 2) set\_firstname, 3) set\_lastname, 4) set\_age, and 5) set\_tagline, 6) set\_gender and need to return Boolean.

You should make an exception to the "usernames cannot be the empty string" rule in the case of the null Person

If the user tries to add a Person to a Community, but a Person with that username already exists in that Community, the function should return false and the Community should remain unchanged.

If a Community tries to send a message and some (at least 1) recipient usernames are not found within the Community, the message should still be sent to all usernames that are found within the Community...but the function should return false.

# Starter Code

We will provide the header files and parts of the implementation (cpp) files in your svn repositories inside the folder

To browse the code, please use this link:

https://phoenixforge.cs.uchicago.edu/projects/<cnetid>-cs22001-win-17/repository/show/warm-up-project

To check out code from the SVN repository, please use the link below:

https://phoenixforge.cs.uchicago.edu/svn/<cnetid>-cs22001-win-17/warm-up-project

*where cnetid is replaced by your actual cnetID*. You *must* use the exact prototypes as written in the header files, because we will test your code against our unit tests that will assume these prototypes! You may use any number of additional data members and helper functions as you want, and you may structure the internal logic of the implementations entirely as you want.

**Do not assume that our provided code is 100% correct! Treat the provided code as if it were untested.** Furthermore, some of the functions we wrote are not quite complete; if this is the case, a comment in the function will denote as such.

Unit Testing

You will be required to provide unit tests for your program. The unit tests should validate that all of the above specifications are met, and each function should be tested in isolation with at least 1 unit test. (This is, after all, the notion of unit testing!) This means that functions should be tested individually, unless a small set of functions relate to each other in a way that allows them to naturally be tested together (i.e., get\_username and set\_username). You will be graded not only on your own code's conformance to the specification, but on the absolute thoroughness of your unit tests. In fact, we will use a subtly broken version of the program to, effectively, test your unit tests!

We have provided some code to get you started on writing the unit tests as well.

Few Note for unit Testing

1. Unit test your constructors!
2. There are two things that need to be checked when performing unit testing:
   1. The actual data crunched/returned by various functions you are unit testing,
   2. The changes in state enacted by those functions. For example, if a setter function returns false, it needs to retain its previous state. If set\_info returns false, there may be valid changes in state that need to be tested.
3. A good example of non-obvious branch testing is that in add\_person, you need to test for failure because a) the person had invalid fields, and also, separately, because b) the person had valid fields, but another person with that username was already in the community.
4. Don't forget to test the "null" outputs of all functions! If a function can return an empty list, make sure that you test that particular scenario. Again, if there are many ways to achieve such a scenario that are independent from each other, they should all be tested.
5. There is a fine line between unnecessarily obsessive testing and insufficient testing. Try to achieve a balance across all functions: don't blast some and weakly test others, poorly distributing the time you invest in writing the tests. Although, testing more "fundamental" functions a bit more rigorously than higher-level ones might be wise.
6. You can create "fake" scenarios to perform unit testing. If your program requires some interactivity, you can write hooks that simulate that interactivity. If your program requires two devices to talk to each other, maybe you can simulate those two devices on one machine to simplify testing. Alternatively, you can make the unit tests interactive. Ultimately, do whatever achieves the goal of verifying the correctness/robustness of your code through the least painful means for you possible.

# Questions

The following questions are geared toward making you think about the structure of your code and not just the correctness of it. They are design questions and you should answer them in more than one complete sentence; there is no "correct" answer to any of the questions: nearly any answer will receive full credit if it is explained or justified properly. **There is a file inside the warm-up-project folder of your svn repository called written-answers.txt. You must place your answers to the below questions in this file.**

1. stl map
   1. Why do you think we chose to use an stl map object to store the collection of People within the Community class?
2. inbox
   1. The inbox of each Person is private. We made this design choice explicitly. Why do you think we did this?
   2. [Optional Bonus Question] Say you wanted to add privacy options to the Person class. Say that you wanted to provide a setting that allowed Person objects to restrict the messages they receive to messages from people within their Community. Explain in words (and code snippets if you prefer) how you would go about doing this.
3. design review
   1. What was the best or most clever part of the specification we provided, in terms of design?
   2. What was the worst or most structurally awkward part?

# Tips and Tricks

## C++11

C++11 is the newest widely-implemented standard of C++, and it comes with a lot of great bells and whistles. Here are a few:

<http://en.cppreference.com/w/cpp/language/auto>

<http://en.wikipedia.org/wiki/Anonymous_function#C.2B.2B_.28since_C.2B.2B11.29>

The auto keyword is in particular useful when you don't want to specify the type of a variable because it is very long, e.g., iterator types.

Anonymous functions are great in all languages, and now they're part of C++!

You may find the following documentation useful as well:

<http://www.cplusplus.com/reference/cctype/>

<http://www.cplusplus.com/reference/algorithm/find_if/>

Compiling within eclipse with C++11 is easy: <http://wiki.eclipse.org/CDT/User/FAQ#CDT_does_not_recognize_C.2B.2B11_features>

## What cases should I check?

Unit testing is all about checking edge cases and unique cases. What are the fundamentally unique cases for the functions you wrote? What are the edge cases for the inputs to your functions, i.e., the ones that bound the acceptable inputs?

GTest installation Guide

# Work in terminal

You can work in terminal without Eclipse. We provide a Makefile in the svn repository, so you can simply “make” to compile the starter code, and “./run” to run all the tests. For more information, please refer to the “README” file in the svn repository.

This is an expected outcome.

|  |
| --- |
| [==========] Running 14 tests from 2 test cases.  [----------] Global test environment set-up.  [----------] 7 tests from test\_community  [ RUN ] test\_community.get\_name  [ OK ] test\_community.get\_name (0 ms)  [ RUN ] test\_community.set\_name  [ OK ] test\_community.set\_name (0 ms)  [ RUN ] test\_community.add\_person  [ OK ] test\_community.add\_person (0 ms)  [ RUN ] test\_community.get\_all\_usernames  [ OK ] test\_community.get\_all\_usernames (0 ms)  [ RUN ] test\_community.find\_member  [ OK ] test\_community.find\_member (0 ms)  [ RUN ] test\_community.get\_member  [ OK ] test\_community.get\_member (0 ms)  [ RUN ] test\_community.send\_msg  [ OK ] test\_community.send\_msg (0 ms)  [----------] 7 tests from test\_community (0 ms total)  [----------] 7 tests from test\_person  [ RUN ] test\_person.test\_username  [ OK ] test\_person.test\_username (0 ms)  [ RUN ] test\_person.test\_firstname  [ OK ] test\_person.test\_firstname (0 ms)  [ RUN ] test\_person.test\_lastname  ./test/test\_person.cpp:22: Failure  Value of: person.set\_lastname("abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijkl")  Actual: false  Expected: true  ./test/test\_person.cpp:23: Failure  Value of: "abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcdefghijkl"  Expected: person.get\_lastname().c\_str()  Which is: ""  ./test/test\_person.cpp:26: Failure  Value of: person.set\_lastname("")  Actual: false  Expected: true  [ FAILED ] test\_person.test\_lastname (0 ms)  [ RUN ] test\_person.test\_age  ./test/test\_person.cpp:44: Failure  Value of: person.set\_age(0)  Actual: false  Expected: true  ./test/test\_person.cpp:46: Failure  Value of: person.set\_age(127)  Actual: false  Expected: true  ./test/test\_person.cpp:47: Failure  Value of: 127  Expected: person.get\_age()  Which is: 0  [ FAILED ] test\_person.test\_age (0 ms)  [ RUN ] test\_person.test\_tagline  [ OK ] test\_person.test\_tagline (0 ms)  [ RUN ] test\_person.test\_info  [ OK ] test\_person.test\_info (0 ms)  [ RUN ] test\_person.test\_msg  [ OK ] test\_person.test\_msg (0 ms)  [----------] 7 tests from test\_person (0 ms total)  [----------] Global test environment tear-down  [==========] 14 tests from 2 test cases ran. (0 ms total)  [ PASSED ] 12 tests.  [ FAILED ] 2 tests, listed below:  [ FAILED ] test\_person.test\_lastname  [ FAILED ] test\_person.test\_age  2 FAILED TESTS |

# Work in Eclipse

This gtest tutorial will be helpful for plugin the gtest to the eclipse. The test project is a factorial function, but you should be able to set up our warm-up project following the same procedures.

## Introduction

Software testing is a large and complex subject. Unit testing is the testing of an individual class in isolation from other classes. The goal of unit testing is to isolate each part of the program and show that the individual parts are correct.

## Background

Googletest is one of the most popular C++ unit test frameworks. It works on a variety of platforms (Linux, Mac OS X, Windows, Cygwin, Windows CE, and Symbian). Based on the xUnit architecture. Supports automatic test discovery, a rich set of assertions, user-defined assertions, death tests, fatal and non-fatal failures, value- and type-parameterized tests, various options for running the tests, and XML test report generation.

In this article, I will give a step-by-step description on how to set up Googletest in Eclipse Juno.

## Setup Procedures

### **Step 1: Download googletest.**

Download googletest from:

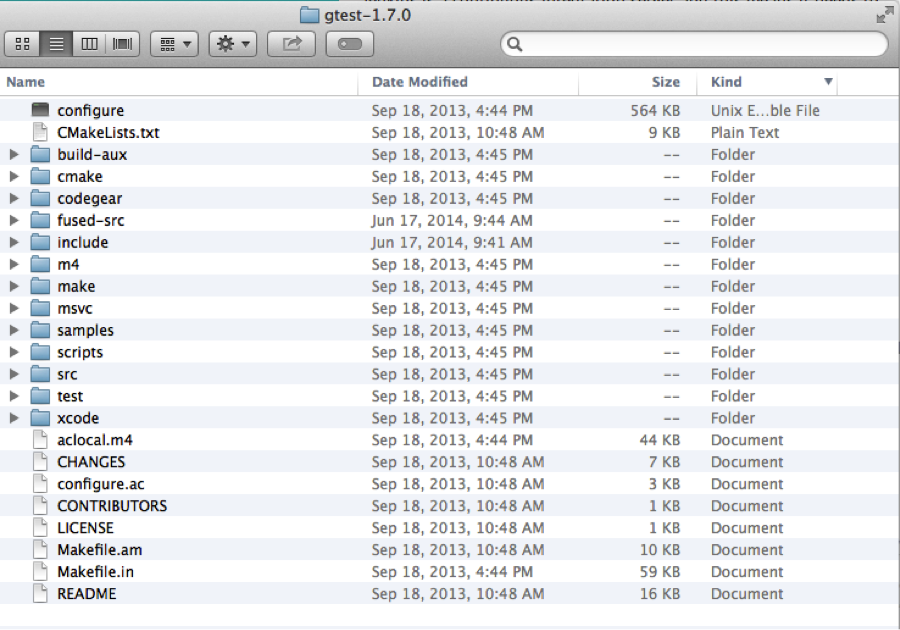
<https://github.com/google/googletest/tree/release-1.7.0>

We are using version 1.7.0

Choose “Clone or Download -> Download ZIP”, then unzip the files.

### **Step 2: Extract gtest-1.7.0.**

Extract gtest-1.7.0, and open the folder gtest-1.7.0. The folder contains several files. “samples” gives ten examples of googletest; “src” has the source code.

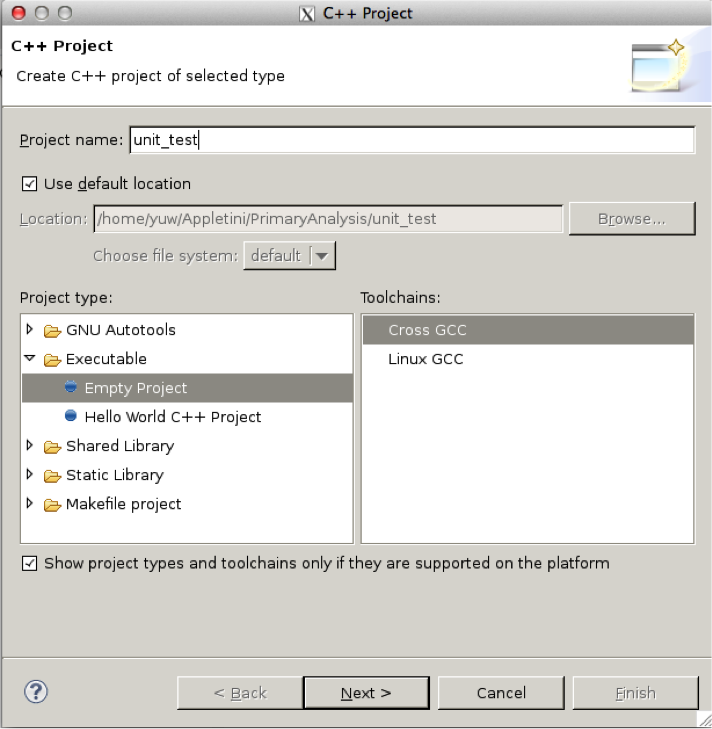


### **Step 3: Setting up working environment for googletest in Eclipse**

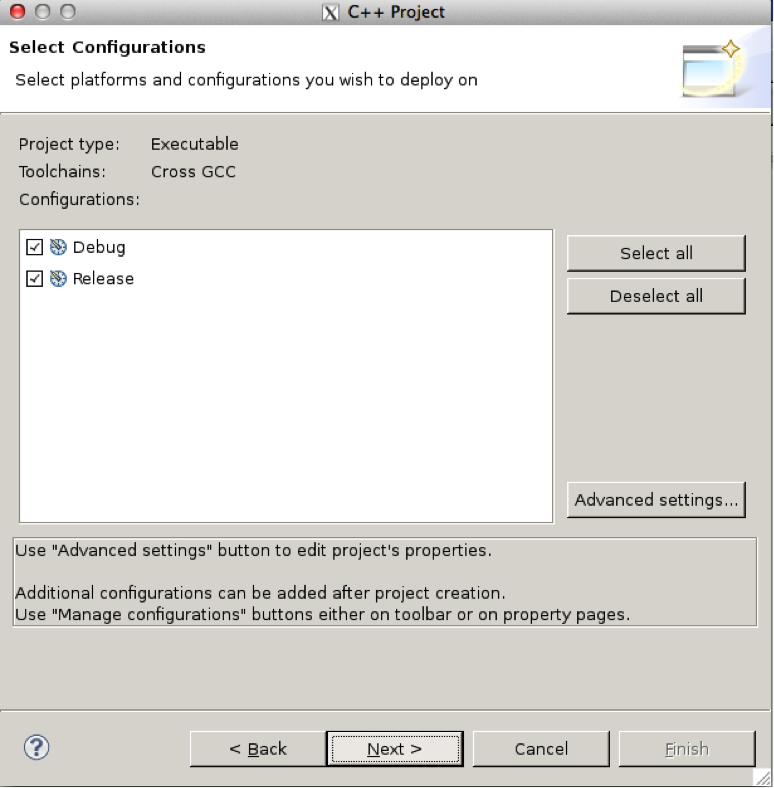
#### Step 3.1 Create unit test project

Open eclipse, go to File->New->C++ Project

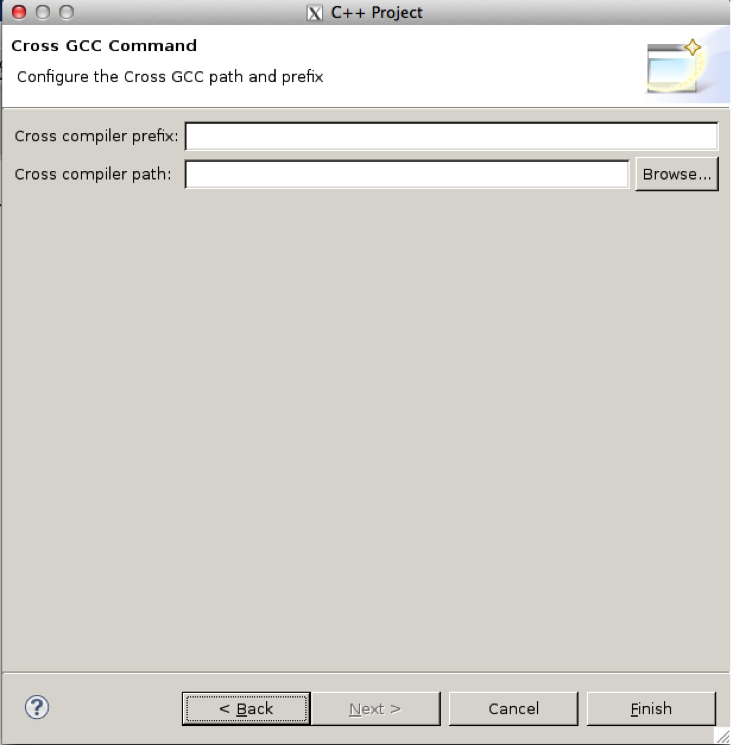
Type the name of your project in “Project name:” such as “unit\_test”,



and then click next, select “Debug” and “Release” in Select Configurations



And then click “next”



And click Finish, unit\_test will appear in the left tab of “Project Explorer”.

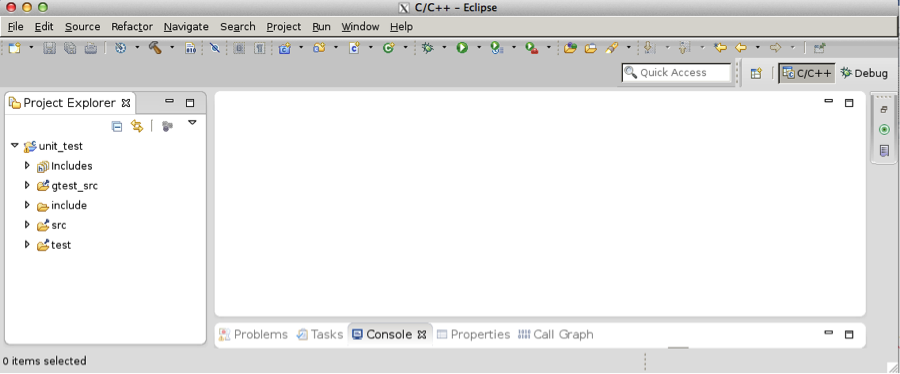
**Step 3.2 adding googletest library, source and test folder.**

**1.**unt\_test-> New -> Folder, create source code (code to be tested) folder “src”.

**2.**unt\_test-> New -> Folder, create header file folder “include”.

**3.**unt\_test-> New -> Folder, create folder “gtest\_src” to store the gtest library.

**4.**unt\_test-> New -> Folder, create folder “test” to store the test code.



#### Step 3.3 adding googletest library, source and test code.

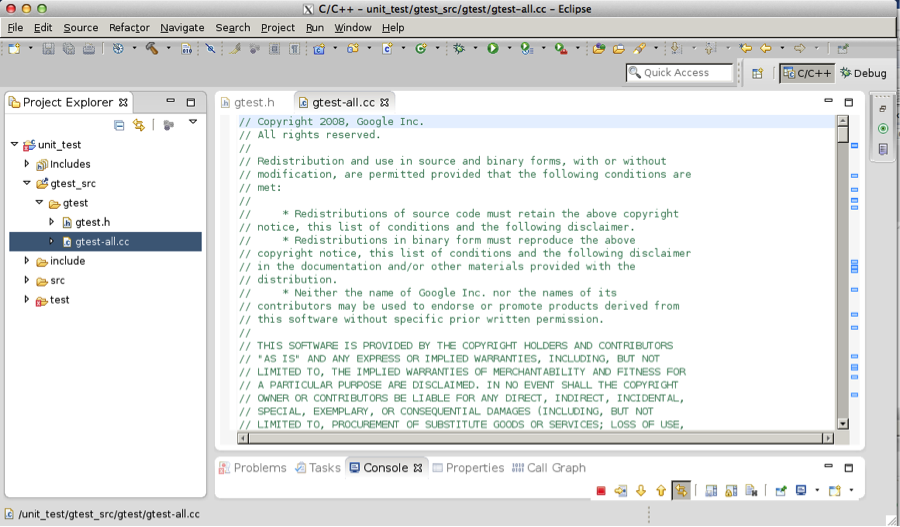
##### 1. adding Googletest library to folder gtest\_src.

The next thing we want to do is add google test library to the gtest\_src folder by running the following command. This step basically copy gtest.h and gtest\_all.cc to the folder of gtest\_src/gtest.



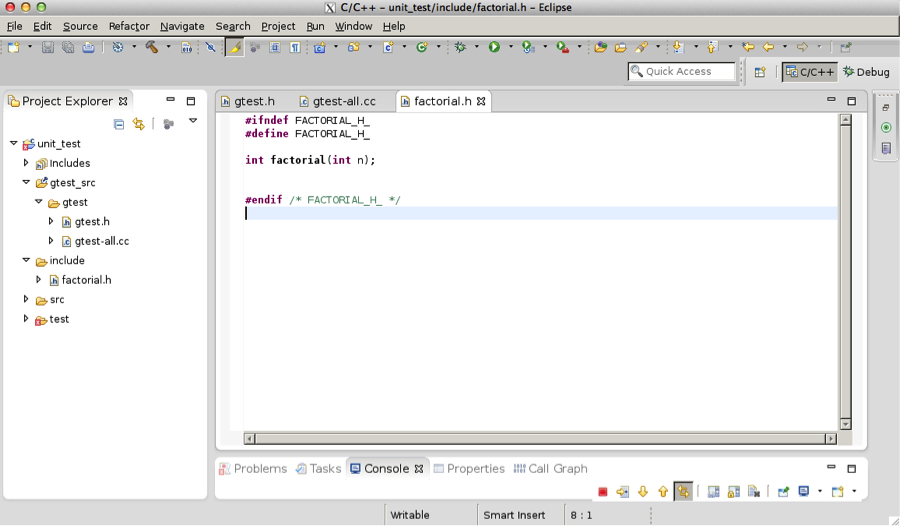
You will need to run the script with python 2.7.

After running the above command in the terminal, the snapshot looks like this after refresh.



##### 2. adding source code.

Folder include->New-> Header File, and create header file “factorial.h”. The code is shown in the Figure.



Hide   Copy Code

*//factorial.h*

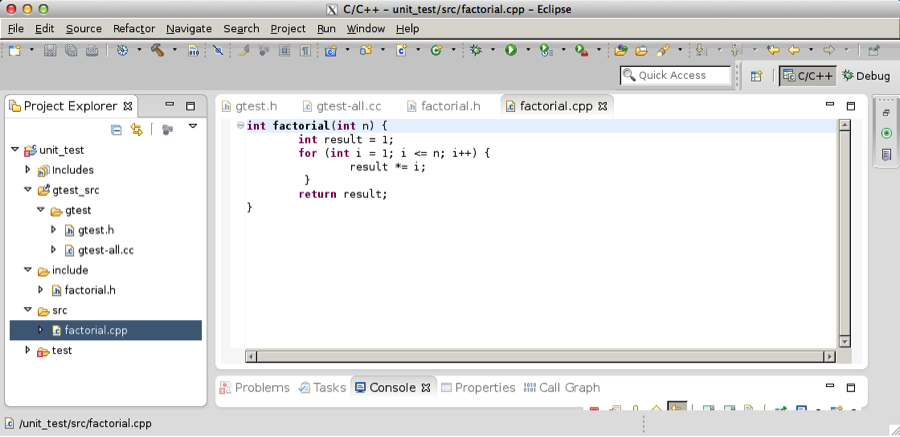
#ifndef FACTORIAL\_H\_

#define FACTORIAL\_H\_

int factorial(int n);

#endif /\* FACTORIAL\_H\_ \*/

Next, folder src->New-> Source File, enter the source code file name, such as “factorial.cpp”.



Hide   Copy Code

*//factorial.cpp*

int factorial(int n) {

int result = 1;

for (int i = 1; i <= n; i++) {

result \*= i;

}

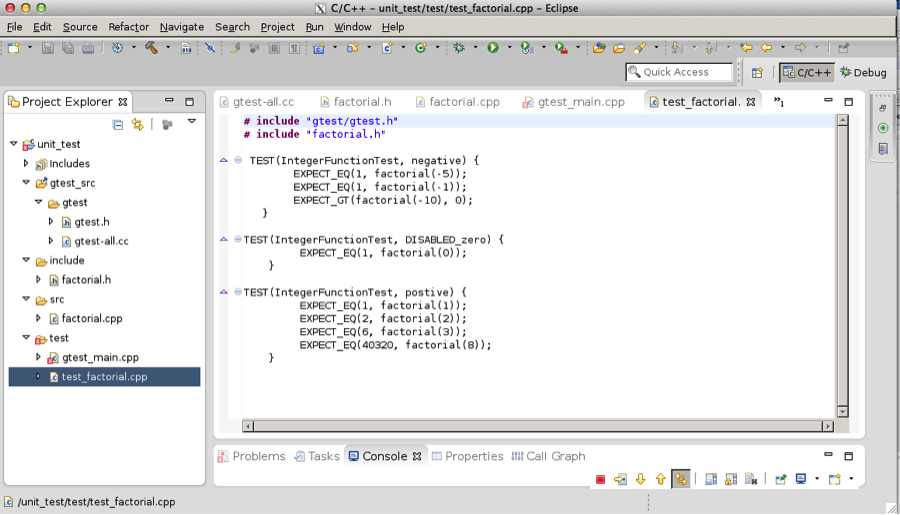
return result;

}

##### 3. Adding test code.

Folder test\_code->New-> Source File, enter the test code file name,

Test code: “test\_factorial.cpp”



Hide   Copy Code

*//test\_factorial.cpp*

# include "gtest/gtest.h"

# include "factorial.h"

TEST(IntegerFunctionTest, negative) {

EXPECT\_EQ(1, factorial(-5));

EXPECT\_EQ(1, factorial(-1));

EXPECT\_GT(factorial(-10), 0);

}

TEST(IntegerFunctionTest, DISABLED\_zero) {

EXPECT\_EQ(1, factorial(0));

}

TEST(IntegerFunctionTest, postive) {

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

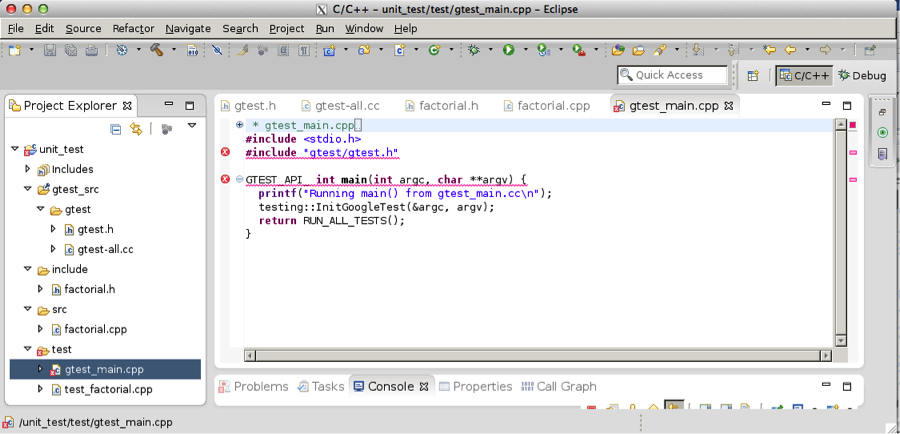
EXPECT\_EQ(2, factorial(2));

EXPECT\_EQ(6, factorial(3));

EXPECT\_EQ(40320, factorial(8));

}

And “gtest\_main.cpp”



Hide   Copy Code

*// gtest\_main.cpp*

#include <stdio.h>

#include "gtest/gtest.h"

GTEST\_API\_ int main(int argc, char \*\*argv) {

printf("Running main() from gtest\_main.cc\n");

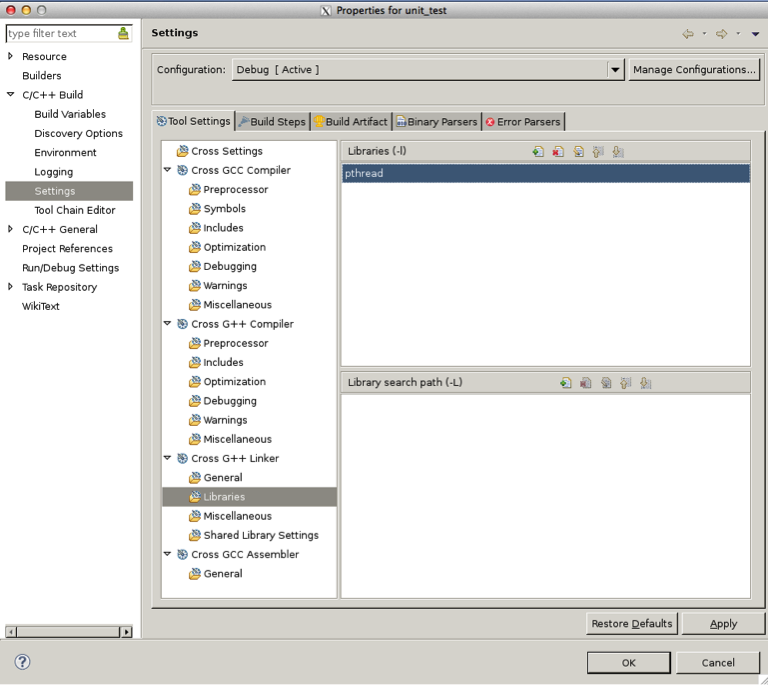
testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

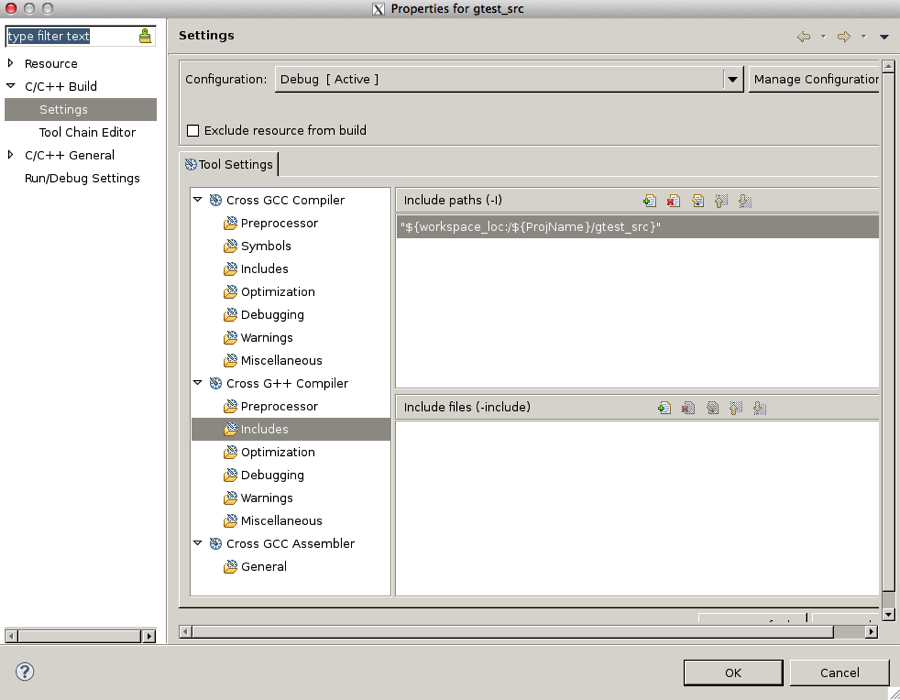
**Step 4: Setup the running environment**

1. Project unit\_test->properties -> C++ linker -> Libraries -> Libraries(-I) -> enter “pthread” and click “ok”

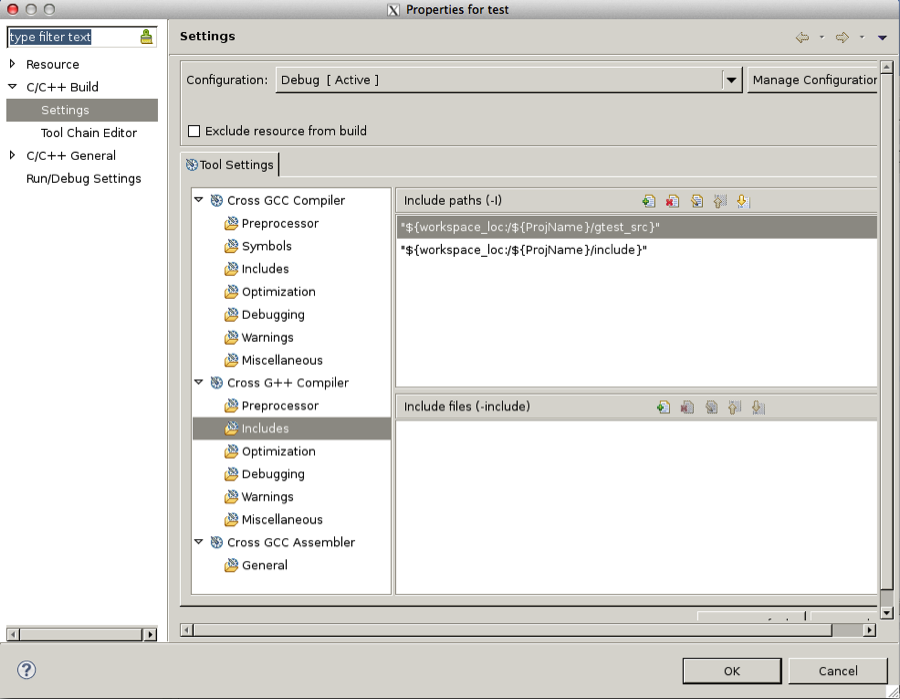


2. folder test-> Properties -> C/C++ Build -> Setting -> Cross G++ Compiler -> includes -> Include paths (-I)

And then click “add..” -> “workspace..”, add folder “include” and “gtest\_src” to the paths.



3. folder gtest\_src-> Properties -> C/C++ Build -> Setting -> Cross G++ Compiler -> includes -> Include paths (-I). And then click “add..” -> “workspace..”, add folder “include” to the path.



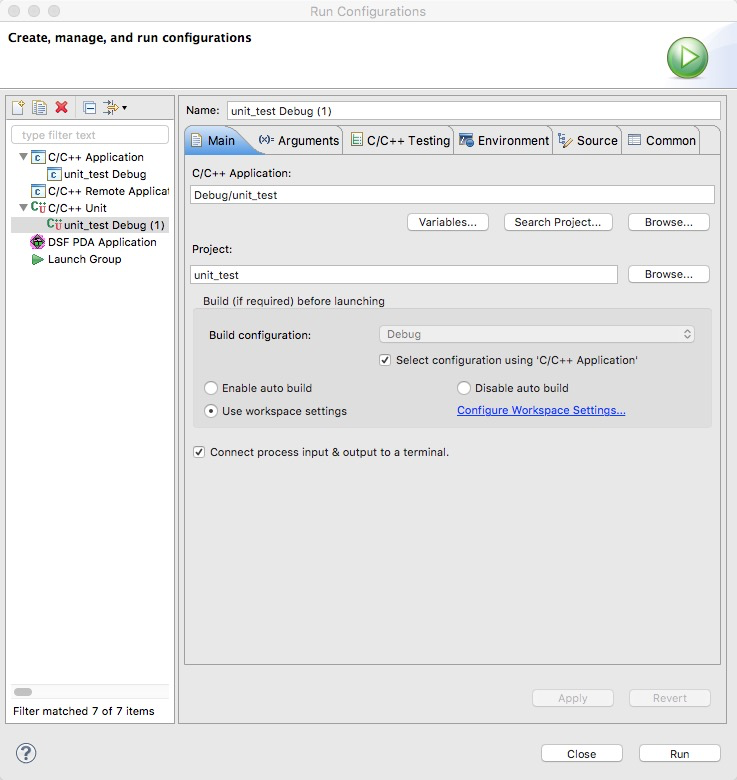
After this, we have set up the running environment, and we can start unit test.

### **Step 5: running Google Test.**

1. In eclipse, go to Project -> Build All or ctrl+B to build the project.

2. go to Run-> Run or ctrl+F11 to run the executable file.

If Run does not work, you may need to go to Run -> Run Configurations, and set as below:



Alternative, we can go to the terminal, go to the Debug folder and run the executable file unit\_test. '

The output is shown below.

