An introduction to the Google C++ Testing Framework

By

Haramohan Sahu

Why use the Google C++ Testing Framework?

- Google C++ Testing Framework helps you write better C++ tests.
- No matter whether you work on Linux, Windows, or a Mac, if you write C++ code, Google Test can help you.
- Google C++ Testing Framework isolates the tests by running each of them on a different object. When a test fails, Google C++ Testing Framework allows you to run it in isolation for quick debugging.
- Google C++ Testing Framework groups related tests into test cases that can share data and subroutines.
- Google C++ Testing Framework doesn't stop at the first test failure. Instead, it only stops the current test and continues with the next. You can also set up tests that report non-fatal failures after which the current test continues. Thus, you can detect and fix multiple bugs in a single run-edit-compile cycle.

Setting up a New Test Project

- To write a test program using Google Test, you need to compile Google Test into a library and link your test with it.
- Googler Test build files for some popular build systems: msvc/ for Visual Studio, xcode/ for Mac Xcode, make/ for GNU make, codegear/ for Borland C++ Builder, and the autotools script (deprecated) and CMakeLists.txt for CMake (recommended) in the Google Test root directory.
- Once you are able to compile the Google Test library, you should create a project or build target for your test program. Make sure you have GTEST_ROOT/include in the header search path so that the compiler can find "gtest/gtest.h" when compiling your test.
- Set up your test project to link with the Google Test library (for example, in Visual Studio, this is done by adding a dependency on gtest.vcproj).
- If you still have questions, take a look at how Google Test's own tests are built and use them as examples.

Example of gtest Setup for 3 platform

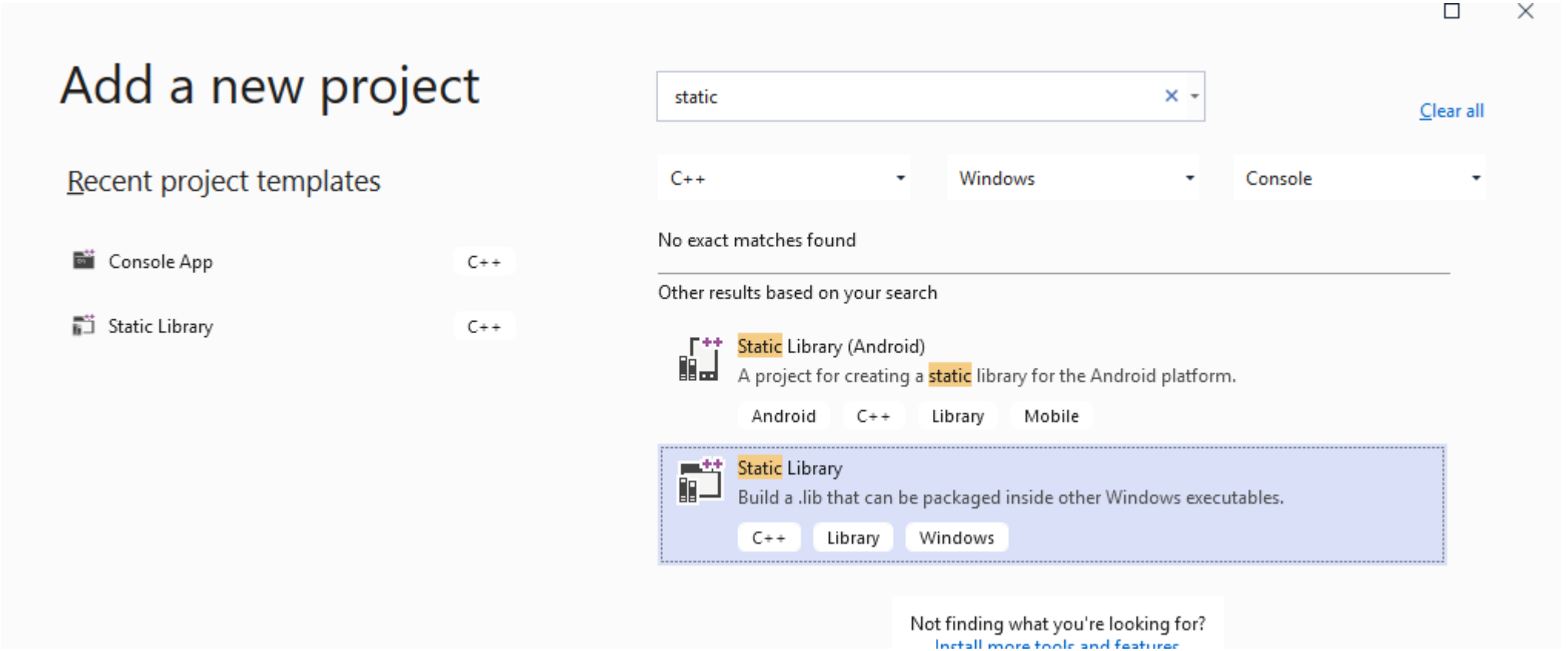
1. Using Visual Studio
2. Using Clion Editor
3. Using Linux

Using Visual Studio

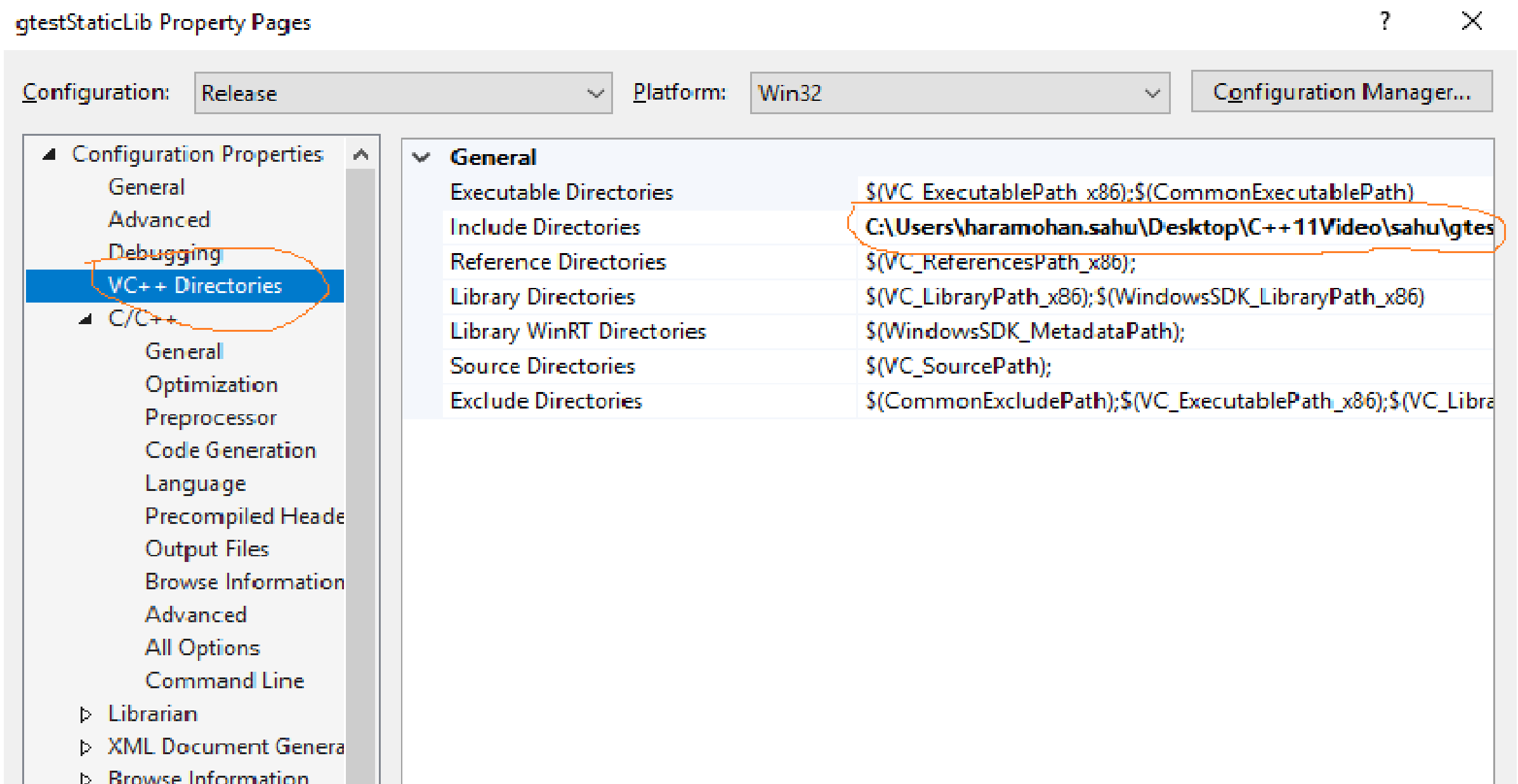
1. Create a project called googletest
2. Go to property pages & set the path of googletest source(which you have downloaded fro internet) **VC++ Directories under Configuration Properties**
3. Example : C:\gtest-1.7.0 and C:\gtest-1.7.0\include
4. Then add source files gtest_main.cc and gtest_all.cc in project source files
5. Then build all.
6. Now your google test static library is created.

Example of gtest Setup for 3 platform cont

- Now create your application , for example MyStack, just make it a static library or normal app whichever you want.

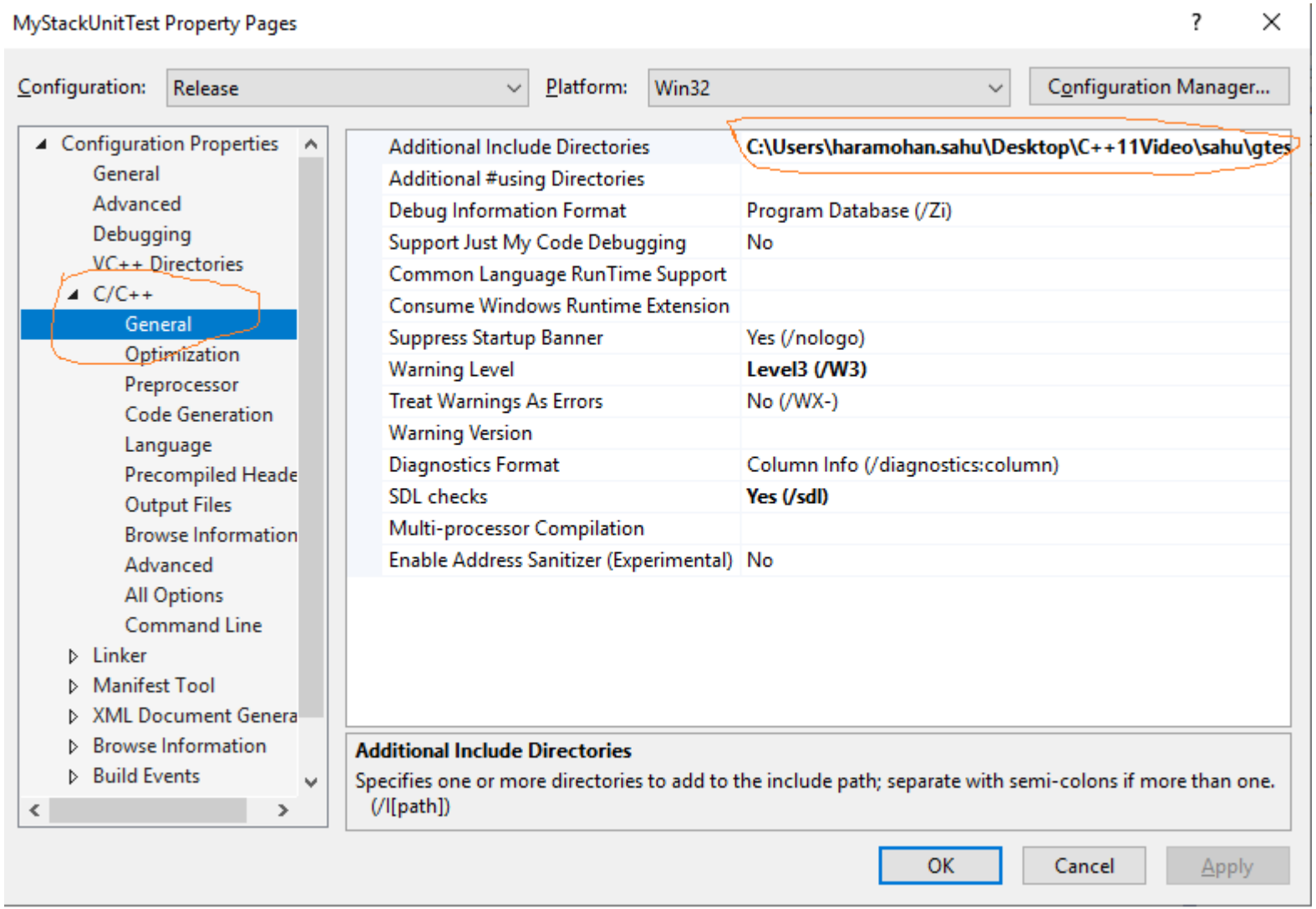


Screen show of how to set

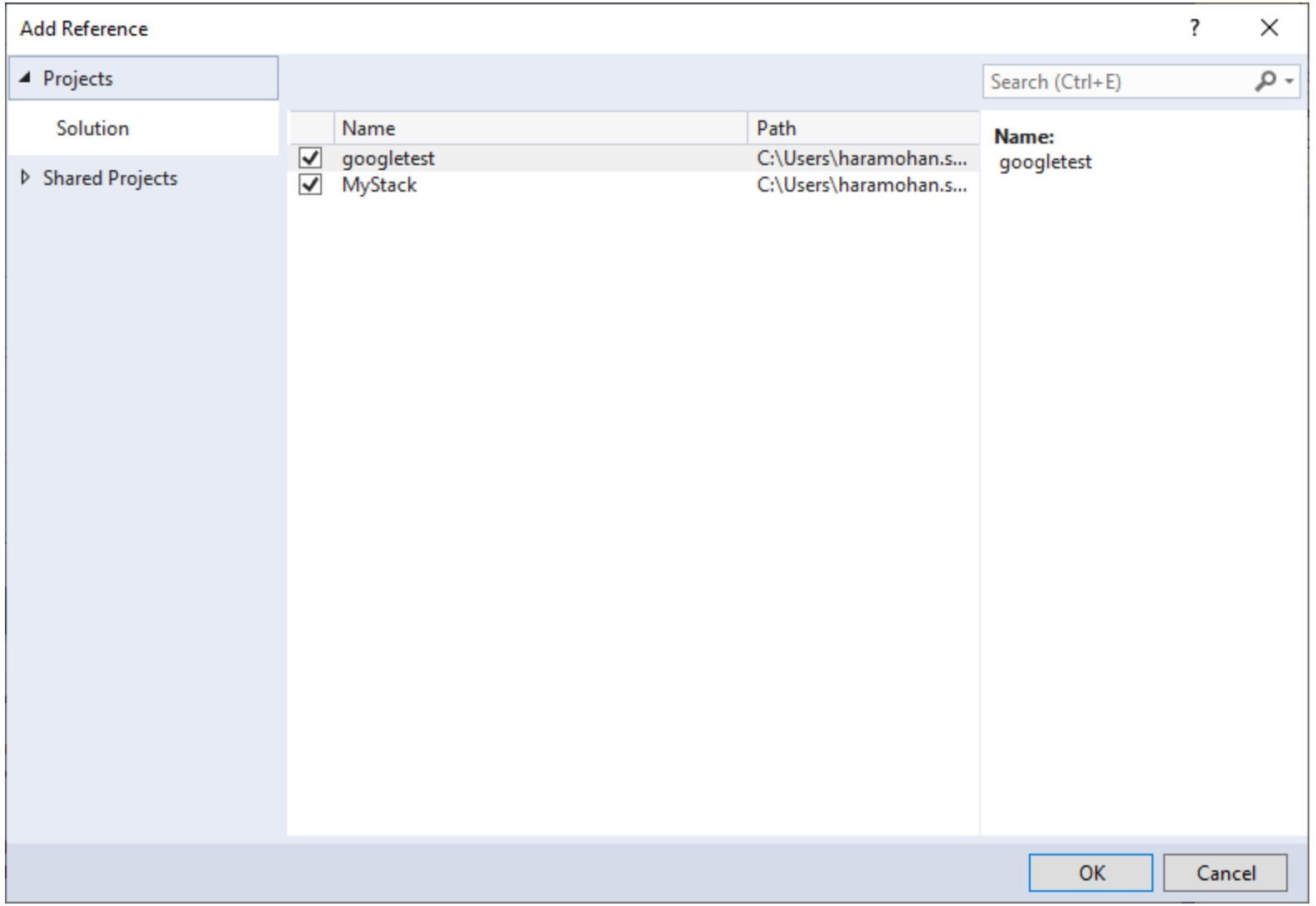


Create your UnitTest Project

- Create new application project with no precompiled headers
- Add include path to gtest directory and gtest directory/include



- After setting the gtest source path, we have to add reference that is googletest lib & your app lib like below:



Add teste cases to your Unit Test

```
#include<gtest/gtest.h>
#include "MyStack.h"

Stack myStackObj(6);

TEST(Stack_1, getStackSize_default)
{
	Stack myStackObj_1;
	EXPECT_EQ(0, myStackObj_1.getStackSize());
}
TEST(Stack_2, getCapacity_2)
{
	EXPECT_EQ(6, myStackObj.getCapacity());
}
```

Basic Assertions

Basic Assertions These assertions do basic true/false condition testing.		
Fatal assertion	**Nonfatal assertion**	**Verifies**
ASSERT_TRUE(condition);	**EXPECT_TRUE(condition);**	**condition is true**
ASSERT_FALSE(condition);	**EXPECT_FALSE(condition);**	**condition is false**

Binary Comparison

In the event of a failure, Google Test prints both *val1* and *val2*.

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_EQ(*val1*,*val2*);	EXPECT_EQ(*val1*,*val2*);	*val1* == *val2*
ASSERT_NE(*val1*,*val2*);	EXPECT_NE(*val1*,*val2*);	*val1* != *val2*
ASSERT_LT(*val1*,*val2*);	EXPECT_LT(*val1*,*val2*);	*val1* < *val2*
ASSERT_LE(*val1*,*val2*);	EXPECT_LE(*val1*,*val2*);	*val1* <= *val2*
ASSERT_GT(*val1*,*val2*);	EXPECT_GT(*val1*,*val2*);	*val1* > *val2*
ASSERT_GE(*val1*,*val2*);	EXPECT_GE(*val1*,*val2*);	*val1* >= *val2*

String Comparison

The assertions in this group compare two C strings. If you want to compare two string objects, use EXPECT_EQ, EXPECT_NE, and etc instead.

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_STREQ(*str1*,*str2*);	EXPECT_STREQ(*str1*,_str_2);	the two C strings have the same content
ASSERT_STRNE(*str1*,*str2*);	EXPECT_STRNE(*str1*,*str2*);	the two C strings have different content
ASSERT_STRCASEEQ(*str1*,*str2*);	EXPECT_STRCASEEQ(*str1*,*str2*);	the two C strings have the same content, ignoring case
ASSERT_STRCASENE(*str1*,*str2*);	EXPECT_STRCASENE(*str1*,*str2*);	the two C strings have different content, ignoring case

Sample Test

Use the TEST() macro to define and name a test function, These are ordinary C++ functions that don't return a value. In this function, along with any valid C++ statements you want to include, use the various Google Test assertions to check values.

The test's result is determined by the assertions; if any assertion in the test fails (either fatally or non-fatally), or if the test crashes, the entire test fails. Otherwise, it succeeds.

```
TEST(test_case_name, test_name) {
 ... test body ...
}
```

Test Fixtures: Using the Same Data Configuration for Multiple Tests

- If you want to write two or more tests that operate on similar data, you can use a test fixture.
- It reuse the same configuration of objects for several different tests.
- To create a fixture, just:
- Derive a class from ::testing::Test .
- Start its body with protected: or public: Inside the class, declare any objects you plan to use.
- If necessary, write a default constructor or SetUp() function to prepare the objects for each test.
- If necessary, write a destructor or TearDown() function to release any resources you allocated in SetUp() .

When using a fixture, use TEST_F() instead of TEST() as it allows you to access objects and subroutines in the test fixture:

Example of Fixture

Also, you must first define a test fixture class before using it in a TEST_F(),

or you'll get the compiler error “virtual outside class declaration”

- Create a fresh test fixture at runtime
- Immediately initialize it via SetUp() ,
- Run the test
- Clean up by calling TearDown()
- Delete the test fixture.

Note that different tests in the same test case have different test fixture objects, and Google Test always deletes a test fixture

before it creates the next one. Google Test does not reuse the same test fixture for multiple tests.

Any changes one test makes to the fixture do not affect other tests.

As an example, let's write tests for a FIFO queue class named Queue, which has the following interface:

```
template <typename E> // E is the element type.
class Queue {
 public:
  Queue();
  void Enqueue(const E& element);
  E* Dequeue(); // Returns NULL if the queue is empty.
  size_t size() const;
  ...
};
```

First, define a fixture class. By convention, you should give it the name FooTest where Foo is the class being tested.

```
class QueueTest : public ::testing::Test {
 protected:
  virtual void SetUp() {
    q1_.Enqueue(1);
    q2_.Enqueue(2);
    q2_.Enqueue(3);
  }
  // virtual void TearDown() {}
  Queue<int> q0_;
  Queue<int> q1_;
  Queue<int> q2_;
};
```

```
TEST_F(QueueTest, IsEmptyInitially) {
  EXPECT_EQ(0, q0_.size());
}

TEST_F(QueueTest, DequeueWorks) {
  int* n = q0_.Dequeue();
  EXPECT_EQ(NULL, n);

  n = q1_.Dequeue();
  ASSERT_TRUE(n != NULL);
  EXPECT_EQ(1, *n);
  EXPECT_EQ(0, q1_.size());
  delete n;

  n = q2_.Dequeue();
  ASSERT_TRUE(n != NULL);
  EXPECT_EQ(2, *n);
  EXPECT_EQ(1, q2_.size());
  delete n;
}
```

Visual Studio setup points

GooleTest using Visual Studio

1) Download Google test & unzip to a folder.
2) Compile it to static librarry.
    2.1) Create a project in visual Studio 2019 for static library without precompiled header
    2.2)
    C:\Users\haramohan.sahu\Desktop\C++11Video\sahu\gtestLib\MyStack;
    C:\gtest-1.7.0\include;
    C:\gtest-1.7.0;
3) Now Add source files from Gtest src directory (gtest_main.cc and gtest_all.cc)
4) now build it to a static library
5) Create your application where its functionality need to be tested with Unit test cases.
    5.1) add a new project
    5.2) create a library or or .exe app (Example
6) Now create the Google Unit Test
    6.1) Add new project
    6.2) set the gtest source directory & gtest /include directory & your application path
    6.3) Now add reference to first two project.
    6.4) Create the test cases where you want to do the unit testing .

Set Up for CLION

- Create a project folder
- Inside the project Folder create folder called lib
- Inside the newly cteated lib folder, place the downloaded google-master folder
- Inside the CMakeList.Txt (This file will be available in clion project)

we have to add below lines.

- add_subdirectory(lib/googletest-master)
- include_directories(lib/googletest-master/googletest/include)
- include_directories(lib/googletest-master/googlemock/include)
- set(SOURCE_FILES main.cpp Test/yourProduction files.h Test/testsoucefile where you want to test the unit of your application.cpp)
- add_executable(SettingUpGoogleTest ${SOURCE_FILES})
- target_link_libraries(SettingUpGoogleTest gtest gtest_main)

Example of CMakeList.Txt

example given below:

==========

```
cmake_minimum_required(VERSION 3.16)
project(SettingUpGoogleTest)

set(CMAKE_CXX_STANDARD 17)
add_subdirectory(lib/googletest-master)
include_directories(lib/googletest-master/googletest/include)
include_directories(lib/googletest-master/googlemock/include)
set(SOURCE_FILES main.cpp Test/Employee.h Test/testEmpFunctionality.cpp)
add_executable(SettingUpGoogleTest ${SOURCE_FILES})
target_link_libraries(SettingUpGoogleTest gtest gtest_main)
```

- Change your main.cpp to below lines:

```
#include <iostream>
#include <gtest/gtest.h>
#include <gmock/gmock.h>
int main(int argc,char ** argv) {
  testing::InitGoogleTest(&argc,argv);
  RUN_ALL_TESTS();
  return 0;
}
```

- Create a folder called Test inside your clion project for actual unit test cases.

```
namespace {
   class TestClassEmployee : public testing::Test{
   public:
      TestEmployee objE;
   public:
      TestClassEmployee() {
      }
   };
}
```

```
TEST_F(TestClassEmployee, empAgeGreaterThen18){
   objE.setAge(20) ;
   ASSERT_EQ(20,objE.getAge());
 }
TEST_F(TestClassEmployee, empAgeValidate){
   objE.setAge(12) ;
   ASSERT_FALSE(objE.validateEmpAge());

}
```

Some more example (Mysql db connection)

```
#include <gtest/gtest.h>
#include <MySqlConnection.h>
#pragma warning(disable : 4996)

TEST(sqlCon, getDriver)
{
   std::string user = "root";
   std::string pass = "password";
   std::string dataBase = "mysql";
   std::string url = "tcp://127.0.0.1";

   MySqlConnection mysqlConnectionObj(url, user, pass);

   EXPECT_EQ(true, mysqlConnectionObj.createDBIFNotExist());
   ASSERT_TRUE(true, mysqlConnectionObj.createDBIFNotExist());

}
TEST(sqlCon_1, getDriver_1)
{
   std::string user = "root";
   std::string pass = "password";
   std::string dataBase = "mysql";
   std::string url = "tcp://127.0.0.1";

   MySqlConnection mysqlConnectionObj(url, user, pass);

   EXPECT_NE(false, mysqlConnectionObj.createDBIFNotExist());

}
TEST(sqlCon_2, getDriver_2)
{
   std::string user = "root";
   std::string pass = "password";
   std::string dataBase = "mysql";
   std::string url = "tcp://127.0.0.1";

   MySqlConnection mysqlConnectionObj(url, user, pass);

   EXPECT_EQ(true, mysqlConnectionObj.createDBIFNotExist());
}
```

```
    //bool createDBIFNotExist();

public:
    bool createDBIFNotExist();
    bool initDB();
    sql::ResultSet* executeQuery(std::string sqlQuery);
    sql::Statement* getStatementObj();
    void directQueryExecute(std::string sqlQuery);
    MySqlConnection(std::string url, std::string user, std::string p);

    void setDatabase(std::string db);
    ~MySqlConnection();

};

bool MySqlConnection::createDBIFNotExist()
{
    std::string sqlRequest = "CREATE DATABASE IF NOT EXISTS student";
    directQueryExecute(sqlRequest);
    directQueryExecute("use student");
    return true;
}

bool MySqlConnection::initDB()
{
    try {
        //!  Create a connection
        con = driver->connect(myUrl, userName, pass);
        if (con) {
            //! Connect to the MySQL test database
            con->setSchema(dataBase);
            stmt = con->createStatement();
        }

    }
    catch (sql::SQLException& e) {
        std::cout << "# ERR: SQLException in " << __FILE__;
        std::cout << "(" << __FUNCTION__ << ") on line " << __LINE__ << std::endl;
        std::cout << "# ERR: " << e.what();
        std::cout << " (MySQL error code: " << e.getErrorCode();
        std::cout << ", SQLState: " << e.getSQLState() << " )" << std::endl;
        return false;
    }
    catch (...)
    {
        std::cout << "exception in DB " << std::endl;
        return false;
    }
    return true;
}
```

```
#include "MySqlConnection.h"

#define DEFAULT_URI "tcp://127.0.0.1"
#define EXAMPLE_USER "root"
#define EXAMPLE_PASS "password"
#define EXAMPLE_DB "mysql"
using namespace std;
/*
  Usage example for Driver, Connection, (simple) Statement, ResultSet
*/
int main(int argc, const char** argv)
{
    const char* url = (argc > 1 ? argv[1] : DEFAULT_URI);
    const string user(argc >= 3 ? argv[2] : EXAMPLE_USER);
    const string pass(argc >= 4 ? argv[3] : EXAMPLE_PASS);
    const string database(argc >= 5 ? argv[4] : EXAMPLE_DB);

    sql::ResultSet* res;
    try {
        MySqlConnection mysqlConnectionObj(url, user, pass);
        string sqlRequest = "SELECT 'Hello World!' AS _message";
        res = mysqlConnectionObj.executeQuery(sqlRequest);


        while (res->next()) {
            //cout << "\t... MySQL replies: ";
            /* Access column data by alias or column name */
            cout << res->getString("_message") << endl;
           // cout << "\t... MySQL says it again: ";
            /* Access column fata by numeric offset, 1 is the first column */
            cout << res->getString(1) << endl;
        }

    }
    catch (sql::SQLException& e) {
        cout << "# ERR: SQLException in " << __FILE__;
        cout << "(" << __FUNCTION__ << ") on line " << __LINE__ << endl;
        cout << "# ERR: " << e.what();
        cout << " (MySQL error code: " << e.getErrorCode();
        cout << ", SQLState: " << e.getSQLState() << " )" << endl;
    }

    return EXIT_SUCCESS;
}
```

THNAK YOU