***GTEST C++ UNIT TESTING FRAMEWORK GUIDELINES***

SOURCE: <https://code.google.com/p/googletest/>

**STEP 1: SETTING UP**

To write a test program using Google Test, you need to compile Google Test into a library and link your test with it.

**Windows Environment**:

* Microsoft Visual Studio 2010 is required (Gtest is not compatible with Visual Studio 2012 as of now).
* Download Gtest, unzip and install.
* Go to -> msvc\gtest.sln, and open with Visual Studio
* Go to the target's properties -> C/C++ -> Code Generation -> Runtime Library, and it needs to be set as "Multi-Threaded" for Release build and "Multi-Threaded Debug" for Debug build.
* Application should point to the directory where gtest header files are stored
* For test codes, create new project and do the following steps:
  + Right click on the solution, go to properties
  + Change Configuration to Debug.
  + Configuration Properties > C/C++ > General > Additional Include Directories: Add <Drive>:\gtest\include
  + Configuration Properties > C/C++ > Code Generation > Runtime Library: If your code links to a runtime DLL, choose Multi-threaded Debug DLL (/MDd). If not, choose Multi-threaded Debug (/MTd).
  + Configuration Properties > Linker > General > Additional Library Directories: Add <Drive>:\gtest\msvc\gtest\Debug or C:\gtest\msvc\gtest-md\Debug, depending on the location of gtestd.lib
  + Configuration Properties > Linker > Input > Additional Dependencies: Add gtestd.lib

**Linux Environment:**

* Install gtest by sudo apt-get install gtest
* Create CMake file to compile and execute.

**STEP 2: WRITING TEST CODES**

When using Google Test, start by writing assertions, which are statements that check whether a condition is true. An assertion's result can be success, nonfatal failure, or fatal failure. If a fatal failure occurs, it aborts the current function; otherwise the program continues normally. Tests use assertions to verify the tested code's behavior. If a test crashes or has a failed assertion, then it fails; otherwise it succeeds.

A test case contains one or many tests. Group tests into test cases that reflect the structure of the tested code. When multiple tests in a test case need to share common objects and subroutines, put them into a test fixture class. A test program can contain multiple test cases.

**STEP 3: FINDING OUT MACROS TO USE**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Fatal Expression | Nonfatal Expression | Verifies |
| Basic Assertion | ASSERT\_TRUE(CONDITION) | EXPECT\_TRUE(CONSITION) | CONDITION IS TRUE |
| ASSERT\_FALSE(CONDITION) | EXPECT\_FALSE(CONSITION) | CONDITION IS FALSE |
| Binary Comparison  (Compare values, can be integer, float etc) | ASSERT\_EQ(EXPECTED,ACTUAL) | EXPECT\_EQ(EXPECTED,ACTUAL) | EXPECTED==ACTUAL |
| 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  (Compare String) | ASSERT\_STREQ(expected\_str, actual\_str) | EXPECT\_STREQ(expected\_str, actual\_str) | Two C strings have same content |
| ASSERT\_STRNE(str1, str2) | EXPECT\_STRNE(str1, str2) | Two C strings have different content |
| ASSERT\_STRCASEEQ(expected\_str, actual\_str) | EXPECT\_STRCASEEQ(expected\_str, actual\_str) | Two C strings have same content ignoring case |
| ASSERT\_STRCASENE(str1, str2) | EXPECT\_STRCASENE(str1, str2) | Two C strings have same content ignoring case |

**Note:** While using String Comparison methods, remember to convert C++ strings to C strings. All the string comparison macros take only C strings.

From my personal experience, it is always advisable to use ASSERT\_EQ (while learning Gtest). It will help find out the error and debug it quickly, neatly and then proceed forward. Once there is good understanding of test framework, it is efficient to use EXPECT\_EQ, since this shows multiple errors (if they exists) in a test file together.

For example, let’s consider a test file where test cases #2,#5, #13 are erroneous. The main difference between ASSERT\_EQ and EXPECT\_EQ is that the first one will show error in test case #2, and will not further process other test cases that come after #2. On the contrary, EXPECT\_EQ will provide the error information in test cases #2, #5,#13 and also will provide success information on other successful test cases.

Gtest macros can be found here: <https://code.google.com/p/googletest/wiki/Primer>