Unit testing using Google Test

A testing framework for testing C++ codes

The structure for all unit tests

Arrange

Act

Assert

Try this and compile

```
1   #include "pch.h"
2   #include <iostream>
3   using namespace std;
4
5   TEST(, ) {
6
7       cout << "TEST Called>>>>" << endl;
8   }
9
10
```

Format of the output

```
Microsoft Visual Studio Debug Console
Running main() from e:\a\_work\2194\s\thirdparty\googletest\goo
t\src\gtest_main.cc
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from
[ RUN      ] .
TEST Called>>>>
[       OK ] . (1 ms)
[----------] 1 test from  (2 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (8 ms total)
[  PASSED  ] 1 test.
```

Try this code

```
1   #include "pch.h"
2   #include <iostream>
3   using namespace std;
4
5   TEST( TESTNAME , SubTestName ) {
6
7       ASSERT_TRUE(100 == 100);
8
9       //cout << "TEST Called>>>>" << endl;
10  }
11
12  TEST(TESTNAME, SubTestName_1) {
13
14      cout << "TEST Called>>>>" << endl;
15  }
16
17
```

Try to compile and see the outcome

Please comment on your observation

Update the code

ASSERT_TRUE(100==101)

Now compile the same code and watch the output

Google Test Assertions

- Success
- Non Fatal Failures EXPECT_
- Fatal Failures ASSERT_

Example

```
2   #include <iostream>
3   using namespace std;
4
5   TEST( TESTNAME , SubTestName ) {
6
7       // non fatal
8       EXPECT_TRUE(100 == 101);
9       cout << "AFTER THE TEST" << endl;
10
11      //cout << "TEST Called>>>>" << endl;
12  }
13
14  TEST(TESTNAME, SubTestName_1) {
15
16      cout << "TEST Called>>>>" << endl;
17  }
18
```

The output

```
Running main() from e:\a\_work\2194\s\thirdparty\googlete
t\src\gtest_main.cc
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from TESTNAME
[ RUN      ] TESTNAME.SubTestName
C:\Users\Daksh\source\repos\GTestVideo\GTestVideo\test.cp
 Value of: 100 == 101
  Actual: false
Expected: true
AFTER THE TEST
[  FAILED  ] TESTNAME.SubTestName (1 ms)
[ RUN      ] TESTNAME.SubTestName_1
TEST Called>>>>
[       OK ] TESTNAME.SubTestName_1 (0 ms)
[----------] 2 tests from TESTNAME (4 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (8 ms total)
[  PASSED  ] 1 test.
[  FAILED  ] 1 test, listed below:
```

Now try this

```
#include <iostream>
using namespace std;

TEST( TESTNAME , SubTestName ) {

    // non fatal
    ASSERT_TRUE(100 == 101);
    cout << "AFTER THE TEST" << endl;
```

```
t\src\gtest_main.cc
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from TESTNAME
[ RUN      ] TESTNAME.SubTestName
C:\Users\Daksh\source\repos\GTestVideo\GTestVideo\test.c
 Value of: 100 == 101
   Actual: false
Expected: true
[  FAILED  ] TESTNAME.SubTestName (1 ms)
[ RUN      ] TESTNAME.SubTestName_1
TEST called>>>>
[       OK ] TESTNAME.SubTestName_1 (0 ms)
```

Some important points to remember

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`

More important points to remember

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_STREQ(str1,str2);	EXPECT_STREQ(str1,str2);	the two C strings have the same content
ASSERT_STRNE(str1,str2);	EXPECT_STRNE(str1,str2);	the two C strings have different contents
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 contents, ignoring case

Now try this

```
#include "pch.h"
#include <iostream>
using namespace std;

TEST( TESTNAME , SubTestName ) {

    // non fatal
    EXPECT_EQ(100, 101);
    cout << "AFTER THE TEST" << endl;

    //cout << "TEST Called>>>>" << endl;
}

TEST(TESTNAME, SubTestName_1) {

    cout << "TEST Called>>>>" << endl;
}
```

Now try this

```
class Check {
    int val;
public:
    Check() : val(0) {}
    void setValue(int newVal) { this->val = newVal; }
    int getVal() { return this->val; }
};

TEST( TESTNAME , SubTestName ) {
    // Arrange

    //

}
```

Write out the unit test for the class

```
TEST( TESTNAME , SubTestName ) {
    // Arrange

    Check* c1 = new Check();
    //Act
    c1->setValue(100);

    //Assert
    ASSERT_EQ(c1->getVal(), 100);

}
```

Write another test

```
TEST(TESTNAME, SubTestName_1) {

    // Arrange

    Check* c1 = new Check();
    //Act
    c1->setValue(500);

    //Assert
    ASSERT_EQ(c1->getVal(), 500);
}
```

Now Compile