Google Test Framework

Index

What is GoogleTest?

- googletest is a testing framework developed by the Testing Technology team with Google's specific requirements and constraints in mind.
- It is a library for writing C++ tests.

Why is it used?

- Googletest helps us to write better C++ tests.
- **Independent and Repeatable:** Googletest isolates the tests by running each of them on a different object.
- **Portable and Reusable**: Googletest works on different OS (Linux, Windows, or a Mac), with different compilers

Installing Gtest on linux

1. Get googletest framework
   wget https://github.com/google/googletest/archive/release-1.8.0.tar.gz
2. Unpack and build google test
   - tar xf release-1.8.0.tar.gz
   - cd googletest-release-1.8.0
   - cmake -DBUILD_SHARED_LIBS=ON .
   - Make
3. "Install" the headers and libs on your system.
   - sudo cp -a googletest/include/gtest /usr/include
   - sudo cp -a googlemock/gtest/libgtest_main.so googlemock/gtest/libgtest.so /usr/lib/
4. Update the cache of the linker
   - sudo ldconfig -v | grep gtest

Types of results of the Test

1. Success: Success is when the test case passed successfully.
2. Non-Fatal Failure: Non-fatal failures are those when the test case fails but the execution of the test does not stop.
3. Fatal Failure: Fatal failures are those when the test case fails and the execution of the test stops.

Steps involved in the test

1. Arrange: In this step we arrange all the necessary requirements for the test.
2. Act: In this step the actual operation is performed.
3. Assert: In this step we verify the results and generate final test results.

Writing a sample unit test

a. Testing a function

Source Code

```cpp
#include<iostream>
#include<gtest/gtest.h>

using namespace std;

// Function to calculate factorial on n
int factorial(int n){
    if (n==1 || n==0){
        return 1;
    }else{
        return n*factorial(n-1);
    }
}

TEST(TestFactorial, Test_1){
    // Arrange
    int n=1;
    // Act
    int result=factorial(n);
    // Assert
    ASSERT_EQ(result,1);
}
```

Output

```
[sabuj@localhost gtest-examples]$ g++ test6.cpp -l gtest -l gtest_main -pthread
[sabuj@localhost gtest-examples]$ ./a.out
Running main() from gtest_main.cc
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from TestFactorial
[ RUN      ] TestFactorial.Test_1
[       OK ] TestFactorial.Test_1 (0 ms)
[----------] 1 test from TestFactorial (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (1 ms total)
[  PASSED  ] 1 test.
[sabuj@localhost gtest-examples]$
```

b. **Testing a Class**

Source Code

```
#include<iostream>
#include<gtest/gtest.h>

using namespace std;

class MyClass{
    private:
        string name;
    public:
        MyClass(string _name){
            this->name=_name;
        }
        string GetName(){
            return this->name;
        }
};

TEST(TestClass, TestGetter1){
    // Arrange
    MyClass object=MyClass("Obama");
    // Act
    string name=object.GetName();
    // Assert
    ASSERT_EQ(name,"Obama");
}
TEST(TestClass, TestGetter2){
    // Arrange
    MyClass object=MyClass("Barak");
    // Act
    string name=object.GetName();
    // Assert
    ASSERT_EQ(name,"Barak");
}
```

Output

```
[sabuj@localhost gtest-examples]$ g++ test4.cpp -l gtest -l gtest_main -pthread
[sabuj@localhost gtest-examples]$ ./a.out
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from TestClass
[ RUN      ] TestClass.TestGetter1
[       OK ] TestClass.TestGetter1 (0 ms)
[ RUN      ] TestClass.TestGetter2
[       OK ] TestClass.TestGetter2 (0 ms)
[----------] 2 tests from TestClass (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (0 ms total)
[  PASSED  ] 2 tests.
[sabuj@localhost gtest-examples]$
```