Assignment #3(Unit Test)

Project Team

T6 Team

Date

2016-10-29

Team Information

201211387 하헌규

201310672 한석호

1. 단위 테스트란?

단위 테스트는 가장 기본적인 실행 가능한 코드에 대한 테스트를 작성하고, 이를 실행하는 것을 말한다. "가장 기본적인 실행 가능한 코드"는 C언어에서는 함수로 이해할 수 있다. 간단하게 말하면, 함수 혹은 메소드가 잘 동작하는지 확인 해보는 작업이라고 말할 수 있다. 프로그램이 커지면 커질수록 모듈의 수는 많아지고, 한눈에 파악하기가 힘들다. 따라서 더 작은 단위로 나눠서, 복잡도를 낮춰 테스트 하는 것이, 잘못된 작업을 수행하는 코드가 어떤 것인지 파악하기 쉽고, 안정성도 더 높일 수 있는 방법이다. 이러한 이유 때문에 단위 테스트가 필요하다. 하지만, 보통 테스트 코드를 작성해야 하는 번거로움 때문에 이를 비효율적이라고 생각하게 되는데, 큰 프로젝트에 대해 문제점을 더욱 쉽게 파악할 수 있고 기본적인 품질을 확보할 수 있다는 점에서 필수적으로 수행되어야 하는 작업이다.

2. 단위 테스트의 방법

단위 테스트의 절차는 생각보다 간단하다. 먼저 자신이 테스트 하고자 하는, 프로젝트 내부의 가장 기본적인 실행 코드, 즉 함수를 작성한 후, 테스트에 사용하는 Framework에서 필요로 하는 테스트 코드를 작성하고, 그에 맞춰서 테스트를 실행하면 된다.

3. 어떤 Framework를 사용할 것인가?

1) Microsoft Unit Testing Framework

Microsoft Unit Testing Framework는 Microsoft의 Visual Studio에 내장되어 있는 Unit Test Framework로, Visual Studio 2012 IDE 버전부터 사용이 가능하다. Microsoft Unit Testing Framework는 매뉴얼이나 Example code가 가장 잘 설명되어 있어 Unit Test라는 작업을 처음해보는 상황에서도 최소한의 작동 능력을 보장할 수 있다고 생각했기 때문에 이번 프로젝트의 가장 주된 Unit Test Framework로 사용할 계획이다.

1-1) 간단한 실습

일단 처음으로 visual Studio로 Native Test Project 프로젝트를 만든다.

```
CalculatorTest                          CalculatorTest::UnitTest1
#include "stdafx.h"
#include "CppUnitTest.h"
#include "C:\Users\하헌규\Documents\Visual Studio 2015\Projects\Calculator\Calculator.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace CalculatorTest
{
    TEST_CLASS(UnitTest1)
    {
    public:

        TEST_METHOD(TestMethod1)
        {
            Assert::AreEqual(1, 1);
            // TODO: Your test code here
        }
        TEST_METHOD(BasicTest) {
            CCalculator cal;
            Assert::AreEqual(2.1, cal.Calculate(1,'+', 1),0.2,L"Basic Test failed",LINE_INFO());
        }


    };
}
```

TEST_METHOD는 하나의 테스트를 정의하는 것을 의미하고, 이는 TEST_CLASS로써 그룹화 된다.

그 다음으로 Win32 프로젝트를 만든다. (DLL 및 내보내기 기호를 선택)

```
// Calculator.cpp : Defines the exported functions for the DLL application.
//

#include "stdafx.h"
#include "Calculator.h"


// This is an example of an exported variable
CALCULATOR_API int nCalculator=0;

// This is an example of an exported function.
CALCULATOR_API int fnCalculator(void)
{
    return 42;
}

// This is the constructor of a class that has been exported.
// see Calculator.h for the class definition
CCalculator::CCalculator()
{
    return;
}
double CCalculator::Calculate(double i, char c, double j)
{
    switch (c)
    {
    case '+':
        return i + j;
    case '-':
        return i - j;
    case '*':
        return i*j;
    case '/':
        return i / j;
    default:
        break;
    }

}
```

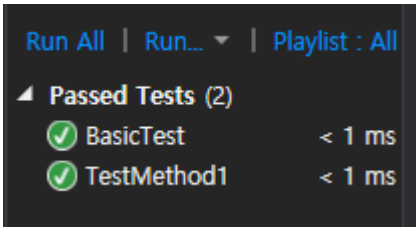
Switch 문을 이용한 아주 기초적인 계산기 함수이다.

```
TEST_METHOD(BasicTest) {
                        CCalculator cal;
                        Assert::AreEqual(2.1, cal.Calculate(1,'+',
1),0.2,L"Basic Test failed",LINE_INFO());
                    }
```

첫 번째 이미지에 나오는 이 테스트 함수는, 테스트의 대상이 되는 함수가 원하는 결과를 의도대로 반환하는지 테스트 해보는 것이다. 사용되는 인자는 아래와 같다.

(Expected Number, Actual Number, Tolerance, Message, Line Number)

이제 테스트를 실행하면, 아래와 같이 테스트가 완료되고 이에 대한 결과값이 Test Explorer에 출력된다.



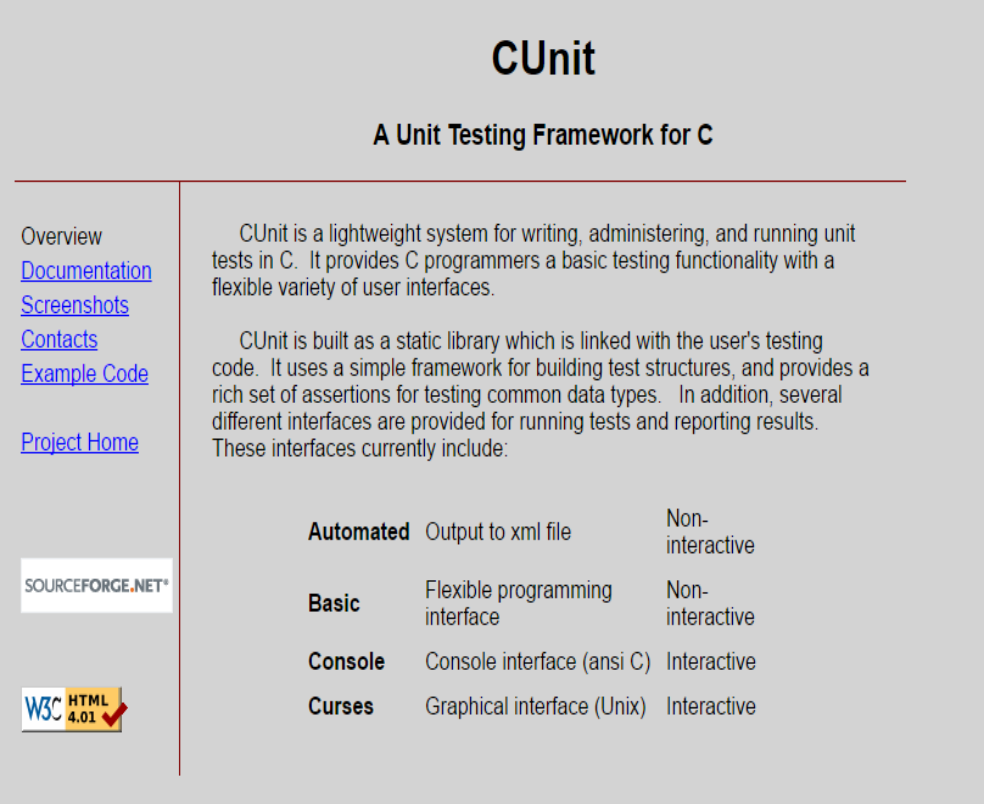
2) Check

Check는 C 언어를 위한 Unit Test Framework이다. Check라는 이름은 Autotools를 이용한 빌드 환경에서 테스트에 관련된 make target의 이름이 check인 것에서 유래했다. Check의 가장 큰 특징은 각 테스트 케이스를 실행할 때 별도의 프로세스를 생성하여 실행한다는 것이다. 이는 segmentation fault가 발생하거나, 내부에서 exit()를 호출하여 비정상 종료되는 경우에도 종료 상태를 감지하고 결과를 보고받을 수 있게 한다.

```
check_test.c:
#include <check.h>

START_TEST(test_check)
{
  fail_unless(1 + 1 == 2, "What's wrong? 1 + 1 = %d", 1 + 1);
}
END_TEST

START_TEST(test_check_new_api)
{
  ck_assert(1);
  ck_assert_int_eq(1 + 1, 2);
  _ck_assert_int(1, <, 2);
  ck_assert_str_eq("test_check_new_api", __FUNCTION__);
}
END_TEST

Suite *
my_suite(void)
{
  Suite *s = suite_create("My Test Suite");

  TCase *tc = tcase_create("My Test Case");
  tcase_add_test(tc, test_check);
  tcase_add_test(tc, test_check_new_api);

  suite_add_tcase(s, tc);
  return s;
}

int
main (void)
{
  Suite *s = my_suite();
  SRunner *sr = srunner_create(s);
  srunner_run_all(sr, CK_ENV);
  return 0;
}
```

예제로 쓰여진 테스트코드를 검색하여 찾아보았다.

3) C Unit Test



C Unit Test 역시 C언어를 위한 Unit Test Framework이다. 범용적인 이름의 특성 때문인지 구체적인 활용법을 찾는 것은 쉽지 않고, 제공하는 README 파일이나 홈페이지의 Example Code를 통해 그 구성을 이해할 필요가 있다.

4. 추가 사항 및 출처

Check와 C Unit Test의 경우 다양하게 제공되는 코드를 통해 테스트할 것을 요구하는데, 이를 모두 이해하고 실행시키는 것이 어려워 조사하고 코드를 찾아보는 것으로 마무리했습니다. 실제 Coffee Machine Project에서 활용하기에도 무리가 있다고 생각합니다. 이에 따라 저희 팀은 Microsoft에서 제공하는 Unit Testing Framework를 사용할 생각입니다.

Unit Test : http://pseg.or.kr/pseg/unitccplus/325

http://blog.naver.com/knix008/220332606212

Check : https://libcheck.github.io/check/index.html

http://egloos.zum.com/studyfoss/v/5346826

CUnit : http://cunit.sourceforge.net/index.html