Unit Test
for C Programming
(CUnit)

201411283 유병찬

Date

2017-11-02

1. Unit Test

-Unit Test 란?

Unit Test는 컴퓨터 프로그래밍에서 소스 코드의 특정 모듈이 의도된 대로 작동하는지 검증하는 절차다. 즉, 모든 함수와 메소드에 대한 테스트 케이스를 작성하는 절차이다. Unit Test를 함으로써 어느 부분이 잘못되었는지를 재빨리 확인할 수 있어서 프로그램의 안정성을 높일 수 있다. 또한 문제점을 금방 확인할 수 있기에 변경하기가 쉽다. 또한 Unit 자체의 불확실성을 제거 해주므로 상향식 텍스트 방식에서 유용하다.

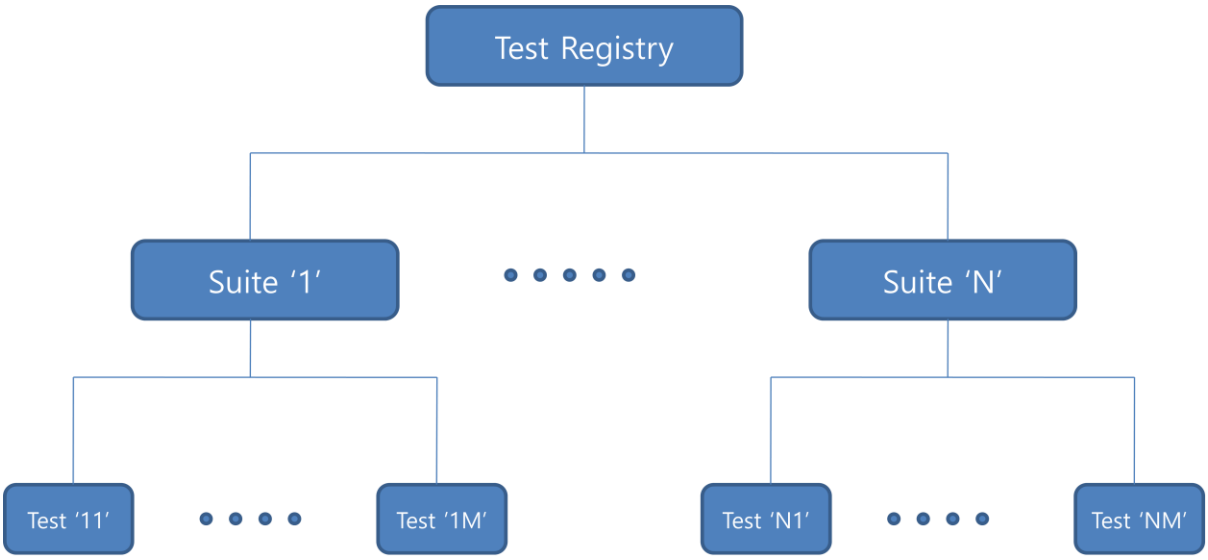
-수행방법

단위 테스트 작성 시 가장 중요하게 인식할 점은 테스트 단위가 복수의 테스트 시나리오들을 가질 수 있다는 것이다. 그리고 모든 테스트 시나리오들은 독립적인 테스트 코드로 작성되어져야 한다. 또한 불필요한 검증 구문을 작성하지 않아야 한다. 단위 내의 모든 것에 대해 검증 구문을 작성하지 말고 테스트하려고 하는 하나의 시나리오에 집중해서 테스트하는 것이 좋다. 이렇지 않으면 하나의 이유로 여러 테스트 케이스가 실패 할 수 있어서 문제점을 찾기 쉽지 않다. 또한 각 테스트는 독립적이어야 한다. 그리고 시스템 설정 파일에 관한 Unit Test를 작성하지 않는 것이 좋다. 그리고 Unit Test 케이스의 이름은 명확하고 일관되게 테스트의 의미를 반영해야 한다.

2. CUnit

-CUnit은 C로 작성된 소스에 대해서 Unit Test를 지원해주는 라이브러리이다.

CUnit의 테스트 구조는 Registry 있고 Registry 안에 Suite가 있고 그 내부에 Test들이 존재하는 구조이다.



-일반적인 사용법

1) 테스트 함수를 작성한다.

-필요에 따라서 Initialize, Cleanup함수도 작성한다.

2) Registry에 등록한다.

-**CU_initialize_registry**()를 사용한다.

3) Registry에 Suite를 등록한다.

-**CU_add_Suite**()를 사용한다.

4) Suite에 Test에 Test를 등록한다.

-**CU_add_Test**()를 사용한다.

5) Running모드 함수를 사용한다.

-**CU_console_run_tests**()등을 이용한다.

6) Registry를 초기화한다.

-**CU_Cleanup_registrty**()를 사용한다.

-Registry API

1)**Cu_ErrorCode CU_initalize_registry**(void)

다른 CUnit 함수들을 사용 전에 반드시 사용되어야 한다.

Return Value)

CUE_SUCCESS initialization was successful.

CUE_NOMEMORY memory allocation failed.

Ex)

```
if(CUE_SUCCESS != CU_initialize_registry())
        return CU_get_error();
```

2) **void CU_cleanup_registry**(void)

테스트가 완료되면 메모리를 반환하기 위하여 사용한다.

Ex) **CU_cleanup_registrty();**

-Suite API

CU_pSuite CU_add_suite(const char* strName, CU_InitializeFunc pInit, CU_CleanupFunc pClean)

Suite를 생성하고, 생성된 Suite는 자동으로 Registry에 등록된다. pInit과 pClean은 0를 return하면 Success로 처리되고, 0이 아니면 Fail로 처리되는 파라미터가 없는 함수를 사용한다. Optional하며 불필요시 NULL을 준다.

Return Value)

CUE_SUCCESS suite creation was successful.

CUE_NOREGISTRY the registry has not been initialized.

CUE_NO_SUITENAME strName was NULL.
CUE_DUP_SUITE the suites' name was not unique.
CUE_NOMEMORY memory allocation failed.

Ex)

```
pSuite = CU_add_suite("Suite_1", init_suite1, clean_suite1);
if(NULL == pSuite){
        CU_cleanup_registry();
        Return CU_get_error();
}
```

-Test 등록 API

CU_pTest CU_add_test(CU_pSuite pSuite, const char* strName. CU_TestFunc pTestFunc)

Suite에 Test를 등록할 때 사용한다.

Return Value)

CUE_SUCCESS test creation was successful.
CUE_NOSUITE the specified suite was NULL or invalid.
CUE_NO_TESTNAME strName was NULL.
CUE_DUP_TEST the test's name was not unique.
CUE_NOMEMORY memory allocation failed.

Ex)

```
If((NULL == CU_add_test(pSuite, "test of fprintf()", testFPRINTF)) )
{
        CU_cleanup_registry();
        Return CU_ger_error();
}
```

-Test API

다음은 CUnit에서 제공하는 Test용 API이다.

#include<CUnit/CUnit.h>

API	**Result**
CU_ASSERT(int expression) **CU_ASSERT_FATAL**(int expression) **CU_TEST**(int expression) **CU_TEST_FATAL**(int expression)	Assert that expression is TRUE (non-zero)

CU_ASSERT_TRUE(value) **CU_ASSERT_TRUE_FATAL**(value)	Assert that value is TRUE (non-zero)
CU_ASSERT_FALSE(value) **CU_ASSERT_FALSE_FATAL**(value)	Assert that value is FALSE (zero)
CU_ASSERT_EQUAL(actual, expected) **CU_ASSERT_EQUAL_FATAL**(actual, expected)	Assert that actual = = expected
CU_ASSERT_NOT_EQUAL(actual, expected)) **CU_ASSERT_NOT_EQUAL_FATAL**(actual, expected)	Assert that actual != expected
CU_ASSERT_PTR_EQUAL(actual, expected) **CU_ASSERT_PTR_EQUAL_FATAL**(actual, expected)	Assert that pointers actual = = expected
CU_ASSERT_PTR_NOT_EQUAL(actual, expected) **CU_ASSERT_PTR_NOT_EQUAL_FATAL**(actual, expected)	Assert that pointers actual != expected
CU_ASSERT_PTR_NULL(value) **CU_ASSERT_PTR_NULL_FATAL**(value)	Assert that pointer value == NULL
CU_ASSERT_PTR_NOT_NULL(value) **CU_ASSERT_PTR_NOT_NULL_FATAL**(value)	Assert that pointer value != NULL
CU_ASSERT_STRING_EQUAL(actual, expected) **CU_ASSERT_STRING_EQUAL_FATAL**(actual, expected)	Assert that strings actual and expected are equivalent
CU_ASSERT_STRING_NOT_EQUAL(actual, expected) **CU_ASSERT_STRING_NOT_EQUAL_FATAL**(actual, expected)	Assert that strings actual and expected differ
CU_ASSERT_NSTRING_EQUAL(actual, expected, count) **CU_ASSERT_NSTRING_EQUAL_FATAL**(actual, expected, count)	Assert that 1st count chars of actual and expected are the same
CU_ASSERT_NSTRING_NOT_EQUAL(actual, expected, count) **CU_ASSERT_NSTRING_NOT_EQUAL_FATAL**(actual, expected, count)	Assert that 1st count chars of actual and expected differ
CU_ASSERT_DOUBLE_EQUAL(actual, expected, granularity) **CU_ASSERT_DOUBLE_EQUAL_FATAL**(actual, expected,	Assert that |actual - expected| <= |granularity| Math library must be linked in for this

granularity)	assertion.
CU_ASSERT_DOUBLE_NOT_EQUAL(actual, expected, granularity) **CU_ASSERT_DOUBLE_NOT_EQUAL_FATAL**(actual, expected, granularity)	Assert that |actual - expected| > |granularity| Math library must be linked in for this assertion.
CU_PASS(message)	Register a passing assertion with the specified message. No logical test is performed.
CU_FAIL(message) **CU_FAIL_FATAL**(message)	Register a failed assertion with the specified message. No logical test is performed.

-Running Test API

Running Test Mode는 다음과 같이 네가지 Mode가 있다.

terface	Platform	Description
Automated	all	non-interactive with output to xml files
Basic	all	non-interactive with optional output to stdout
Console	all	interactive console mode under user control
Curses	Linux/Unix	interactive curses mode under user control

1) Automated

void CU_automated_run_tests(void)

테스트 결과가 XML 파일로 출력된다.

2) Basic

CU_ErrorCode CU_basic_run_tests(void)

테스트 결과가 stdout 으로 출력된다.

3) Console

void CU_console_run_tests(void)

테스트를 command ui 형태로 할 수 있다.

4) Curses

void CU_curses_run_tests(void)

테스트를 curses ui 형태로 할 수 있다.