1. Unit Test란 무엇인가?

단위 테스트(unit test)는 프로그램의 기본 단위인 모듈을 테스트하여 모듈 테스트(module test)라고도 한다. 구현 단계에서 각 모듈의 개발을 완료한 후 개발자가 명세서의 내용대로 정확히 구현되었는지를 테스트한다. 즉 개별 모듈이 제대로 구현되어 정해진 기능을 정확히 수행하는지를 테스트한다. 화이트박스 테스트와 블랙박스 테스트를 모두 사용할 수 있지만 모듈 내부의 구조를 구체적으로 들여다볼 수 있는 화이트박스 테스트 같은 구조적 테스트를 주로 시행한다.

단위 테스트가 개발된 모듈만 테스트하기 때문에 쉬울 것 같지만, 시스템은 수많은 모듈이 서로 정보를 주고받으며 연결되어 있다. 즉 테스트할 모듈을 호출하는 모듈도 있고, 테스트할 모듈이 호출하는 모듈도 있다. 따라서 한 모듈을 테스트하려면 그 모듈과 직접 관련된 상위 모듈과 하위 모듈까지 모두 존재해야 정확히 테스트할 수 있다.

그러나 하나의 모듈을 테스트할 때 상위나 하위 모듈이 개발이 안 된 경우도 있다. 이때 상위나 하위 모듈이 개발될 때까지 기다릴 수는 없으므로 가상의 상위나 하위 모듈을 만들어 사용해야 한다. 상위 모듈의 역할을 하는 가상의 모듈을 테스트 드라이버(test driver)라 하고 그 역할은 테스트할 모듈을 호출하는 것이다. 즉 필요한 데이터를 인자를 통하여 넘겨주고, 테스트가 완료된 후 그 결과 값을 받는 역할을 해준다.

반대로 하위 모듈의 역할을 하는 모듈을 테스트 스텁(stub)이라고 한다. 스텁 모듈은 테스트할 모듈이 호출할 때 인자를 통해 받은 값을 가지고 수행한 후 그 결과를 테스트할 모듈에 넘겨주는 역할을 한다. 따라서 드라이버와 스텁 모듈은 테스트할 때 필요한 기능만 제공할 있도록 단순히 구현한다.

[네이버 지식백과] 단위 테스트 (쉽게 배우는 소프트웨어 공학, 2015. 11. 30., 한빛아카데미(주))

2.Unit Test의 장점

-문제점 발견

유닛 테스트의 목적은 프로그램의 각 부분을 고립 시켜서 각각의 부분이 정확하게 동작하는지 확인하는 것이다. 즉, 프로그램을 작은 단위로 쪼개서 각 단위가 정확하게 동작하는지 검사하고 이를 통해 문제 발생 시 정확하게 어느 부분이 잘못되었는지를 재빨리 확인할 수 있게 해준다. 따라서 프로그램의 안정성이 높아진다. 유닛 테스트는 일견 개발 시간을 증가 시키는 것처럼 보이지만 개발 기간 중 대부분을 차지하는 디버깅 시간을 단축시킴으로써 여유로운 프로그래밍을 가능케 한다.

-변경이 쉽다

프로그래머는 언제라도 유닛 테스트를 믿고 리팩토링을 할 수 있다. 리팩토링 후에도 해당 모듈이 의도대로 작동하고 있음을 유닛 테스트를 통해서 확신할 수 있다. 이를 회귀 테스트(Regression testing)라 한다. 어떻게 코드를 고치더라도 문제점을 금방 파악할 수 있고 수정된 코드가 정확하게 동작하는지 쉽게 알 수 있게 되므로 프로그래머들은 더욱 더 의욕적으로 코드를 변경할 수 있게 된다. 좋은 유닛 테스트 디자인은 그 유닛이 사용되는 모든 경로를 커버할 수 있는 테스트 케이스를 만들어 준다.
지속적인 유닛 테스트 환경을 구축하면 어떠한 변화가 있더라도 코드와 그 실행이 의도대로 인지를 확인하고 검증 할 수 있게 된다. 확립된 개발 방법과 유닛 테스트의 범위에 따라서 프로그램의 정확성이 좌우된다.

-통합이 간단하다

유닛 테스트는 유닛 자체의 불확실성을 제거해주므로 상향식(bottom-up) 테스트 방식에서 유용하다. 먼저 프로그램의 각 부분을 검증하고 그 부분들은 합쳐서 다시 검증하는 통합 테스트에서 더욱 더 빛을 발한다.

3. 수행 방법

단위 테스트는 일반적으로 자동화되어 있지만 수동으로 수행 할 수도 있습니다. IEEE는 다른 하나를 선호하지 않습니다. 단위 테스트의 목적은 단위를 분리하고 그 단위의 정확성을 검증하는 것입니다. 단위 테스트에 대한 수동 접근 방식은 단계별 지침 문서를 사용할 수 있습니다. 그러나 자동화는 이를 달성하는 데 효율적이며 이 기사에 나열된 많은 이점을 가능하게 합니다. 반대로 신중하게 계획하지 않으면 부주의 한 수동 단위 테스트 케이스가 많은 소프트웨어 구성 요소를 포함하는 통합 테스트 케이스로 실행될 수 있으므로 단위 테스트를 위해 설정된 목표의 전부는 아니더라도 대부분의 성취를 방지 할 수 있습니다.

자동화 된 접근법을 사용하면서 고립의 효과를 완전히 실현하기 위해 테스트 대상 단위 또는 코드 본문이 자연 환경 외부의 프레임 워크 내에서 실행됩니다. 즉, 원래 생성 된 제품 또는 호출 컨텍스트 외부에서 실행됩니다. 이러한 분리 된 방식으로 테스트하면 테스트 중인 코드와 제품의 다른 단위 또는 데이터 공간 사이의 불필요한 종속성이 드러납니다. 이러한 종속성은 제거 될 수 있습니다.

자동화 프레임 워크를 사용하여 개발자는 정확성을 확인하기 위해 테스트에 올바른 것으로 알려진 oracle 또는 결과를 코딩합니다. 테스트 케이스 실행 중에 프레임 워크는 모든 기준을 충족하지 않는 테스트를 기록합니다. 많은 프레임 워크는 이러한 실패한 테스트 케이스를 자동으로 표시하고 요약하여보고합니다. 실패의 심각도에 따라 프레임 워크는 후속 테스트를 중단시킬 수 있습니다.

결과적으로 단위 테스트는 전통적으로 프로그래머가 분리되고 일관된 코드 본문을 만드는 동기입니다. 이 연습은 소프트웨어 개발에서 건강한 습관을 조장합니다. 디자인 패턴, 단위 테스트 및 리팩토링은 종종 함께 작동하여 최상의 솔루션이 나타날 수 있습니다.

4. C Language Unit Testing Tool

<Min Unit>

1) Introduction

단위 테스트 프레임 워크는 객체 지향 프로그래밍 세계에서 널리 사용됩니다. JUnit (Java 용), SUnit (Smalltalk 용) 및 CppUnit (C ++ 용)과 같은 프레임 워크는 풍부한 기능 세트를 제공합니다. 그러나 이 풍부한 기능 세트는 C로 작성된 임베디드 시스템과 같이 보다 제한된 환경에서 단위 테스트를 수행하려는 사람에게 위협적 일 수 있습니다. 단 단위 테스트의 중요한 점은 프레임 워크가 아니라 테스트입니다. MinUnit은 C로 작성된 매우 간단한 단위 테스트 프레임 워크입니다. 메모리 할당을 사용하지 않으므로, ROMable 코드를 포함하여 거의 모든 상황에서 정상적으로 작동합니다.

2) Source Code

Macro에 여러 줄의 statement를 사용하여 작성하므로 do-while문을 사용하여 작성하는 것이 특징.

Source Code

```
/* file: minunit.h */
#define mu_assert(message, test) do { if (!(test)) return message; } while (0)
#define mu_run_test(test) do { char *message = test(); tests_run++; \
                               if (message) return message; } while (0)
extern int tests_run;
```

3) Test code

MinUnit 테스트 케이스는 테스트가 통과하면 0 (null)을 반환하는 함수이다. 테스트가 실패하면 함수는 실패한 테스트를 설명하는 문자열을 반환한다. mu_assert는 단순히 전달 된 표현식이 false 인 경우 문자열을 반환하는 매크로다. mu_runtest 매크로는 테스트 케이스를 호출하고 해당 테스트 케이스가 실패하면 리턴 한다. 어렵고 복잡하게 생각되는 testing을 간다한 헤더 파일을 작성하여 원하는 기댓값만 바꾸어 주여 실행하는 tesing이다.

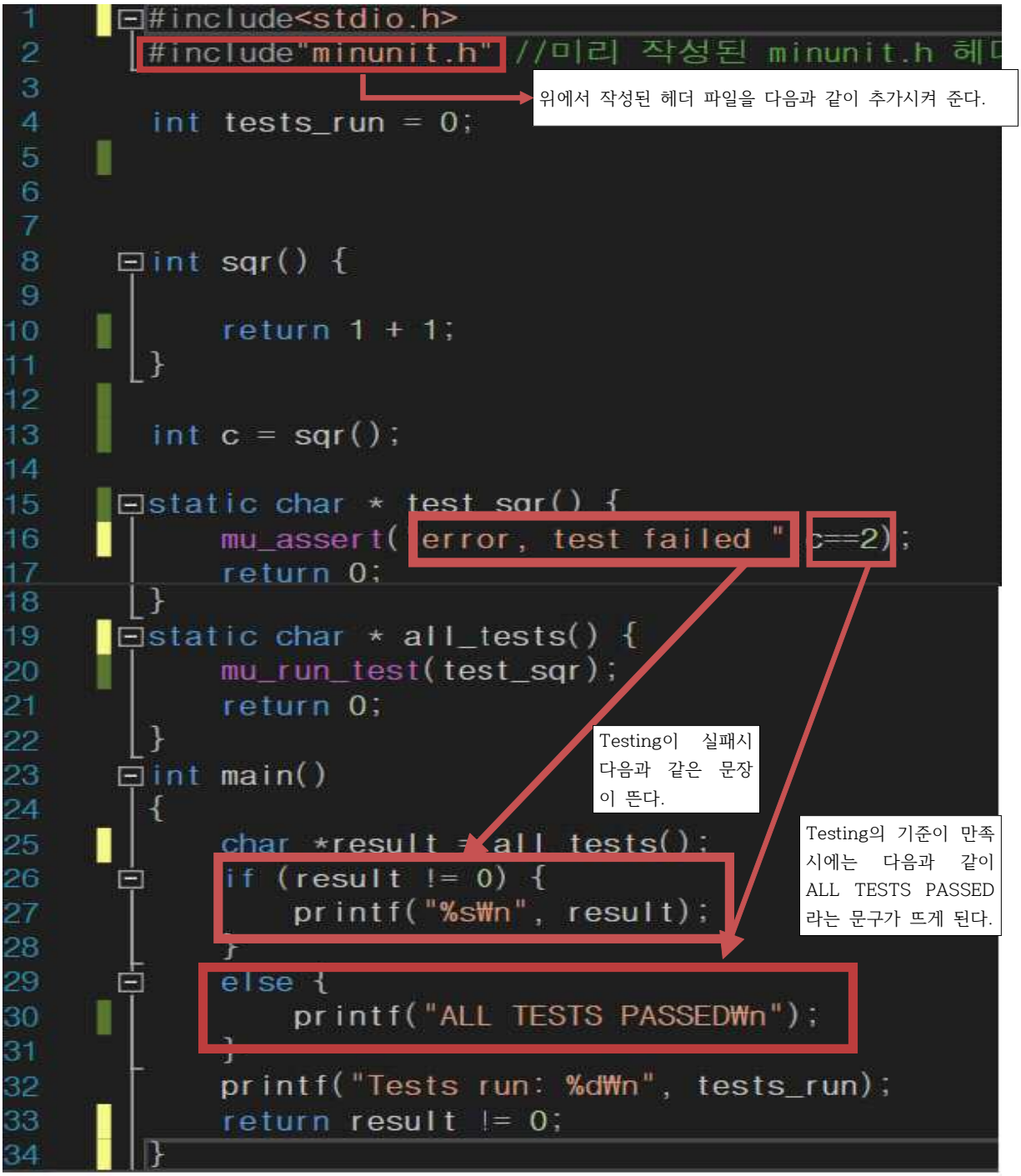
1+1의 결과를 return 하는 함수 sqr() 를 간단히 test 해보겠다.

다음과 같이 minunit.h 헤더 화일을 작성한다.

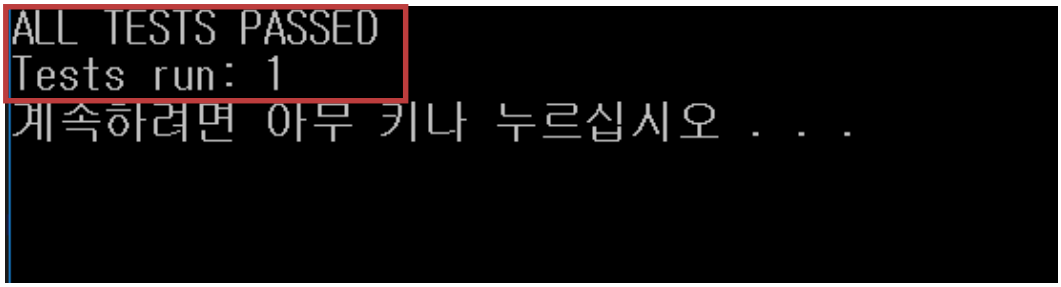
```
/*file: minunit.h*/
#define mu_assert(message,test) do {if(!(test))return message;} while(0)
#define mu_run_test(test) do{char *message = test(); tests_run++; if(message) return message;} while(0)

extern int tests_run;
```

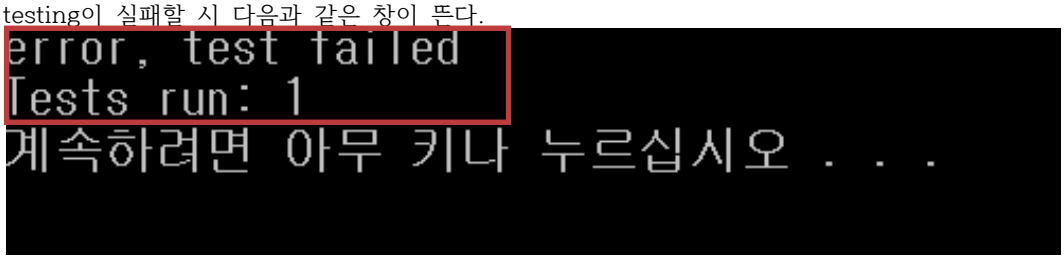
sqr()함수를 테스팅 하기 위하여 다음과 같이 코딩을 하였다.



만약 testing 이 통과되면 다음과 같은 창이 뜬다.



testing이 실패할 시 다음과 같은 창이 뜬다.



mu_assert()에 두 번째 인자 값에 testing에서 함수의 결과로 기대대는 값을 넣어주고 그 기댓값과 다를시 첫 번째 인자에 있는 문자열을 반환해준다. 위 식에서 Tests run은 mu_run_test가 실행될 때 마다 1씩 증가하므로, testing한 항목의 수이다.

4. References

http://www.jera.com/techinfo/jtns/jtn002.html

http://terms.naver.com/entry.nhn?docId=3533037&cid=58528&categoryId=58528

https://en.wikipedia.org/wiki/Unit_testing