

C++ Korea 4<sup>th</sup> Seminar

C++ 프로젝트 ~처음 만난 세계~

# Visual C++에서 유닛 테스트 하기

Com2us

최흥배



**1.유닛 테스트?**

**2.Visual C++에서 유닛 테스트 하기**

**3.사례 – 계산기, 서버 개발에서 유닛 테스트**

**4. 마무리**



유닛 테스트 왜 해요?

# "TDD는 죽었다" - Rails를 만든 DHH의 글

---

2014-04-25

- Test-first fundamentalism(테스트 우선 근본주의)는 마치 금욕을 하라고만 하는 성교육과 같다.
  - 자기 혐오에 빠져있는 사람을 위한 비현실적이고 실효성없는 도덕교육 같은 것이다.

## 그래서 우리는 어디로 가야 하나?

---

- 첫 단계로, 문제가 있다는 것은 인정하자.
  - 두번째 단계는, unit 부터 system 사이의 테스트 스펙트럼의 균형을 잡는 것이다.
  - 현재의 광신적인 TDD 운동은 unit 테스트를 포커스하는 경향이 있다.

출처: <https://sangwook.github.io/2014/04/25/tdd-is-dead-long-live-testing.html>



# TDD 잘알못을 위한 돌직구 세미나 참석 후기

세미나 · 2018.06.22 08:32

OKKY 최단시간 마감 세미나!

TDD 잘알못을 위한 돌직구 세미나에 다녀왔습니다.

- [세미나 링크](#)







실내운전연습장

**개발 잘 하기, 시간 절약....**

# 개발 일정 계산 실수

## 프로그램 개발 시 생각 이상으로 디버깅에서 많은 시간 소요

경험이 부족하면 개발 일정을 계산 할 때 디버깅 시간을 계산하지 못해서 일정 계산에 실수 발생

그렇다고 디버깅 시간을 계산하려고 하니....

코드에 버그가 있을지? 없을지?

있다면 얼마나 있을지?

알기 힘들다....





코드에 버그가 없으면 디버깅 시간도 없어지겠죠...



그래서 유닛 테스트...

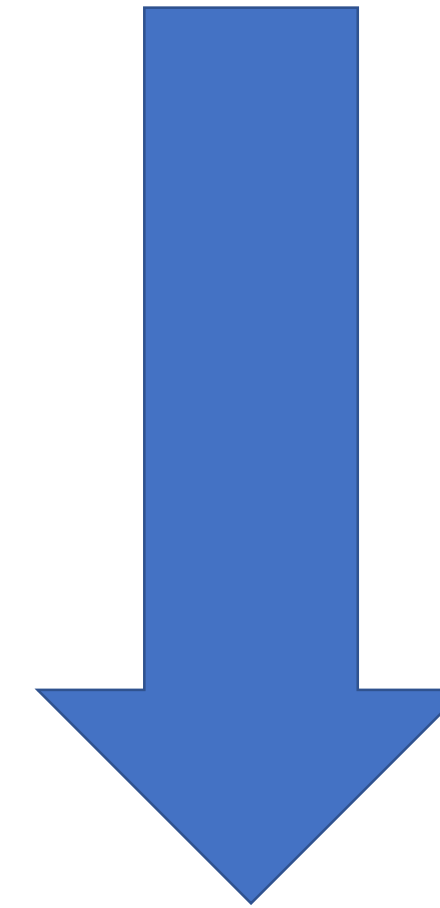
# 디버깅과 시간

수정 비용이 커진다

프로그래머 자신이 하는 테스트

QA에서 잡은 버그

유저가 보고한 버그



# 신념 만으로는...

버그는 만들지 않겠다 라는 신념 만으로는 잡을 수 없음

체계화된 시스템

기술이 필요





# 유닛 테스트

버그를 사전에 잡을 수 있는 방법 중의 하나

프로그래머 단계에서 꽤 많은 수의 버그를 잡을 수 있음



# 그러나....

유닛 테스트는 공짜가 아님  
개발하기에도 바쁜데....

개발 코드에 수정이 발생하면 그 이상으로 유닛 테스트 코드에도  
수정 발생  
중복 코드는 악!!!

유닛 테스트는 해당 기능의 배움보다 경험이 꽤 중요



# 또, 유닛 테스트 만으로는 부족

유닛 테스트를 잘 했다고 해서 버그가 0은 결코 아니다.

유닛 테스트는 전체 보다는 부분 부분을 테스트 하는 것  
부분 부분이 정확하다면 전체도 정확할 **확률이 높음**

**AutoTest와 QA 테스트도 필요**

# 성공 보다는 실패

보통 기능의 성공에 대한 부분은 유닛 테스트 없이 테스트가 가능.

**그러나 실패에 대한 테스트는 하기 어렵고, 잘 하지 않음.**

정말 중요 보통 시연, QA 테스트에서는 성공에 대한 것만 테스트  
하지 실패 상황에 대한 테스트를 하기 어려움

유닛 테스트는 실패에 대한 테스트를 다양하게 자주 할 수 있다.

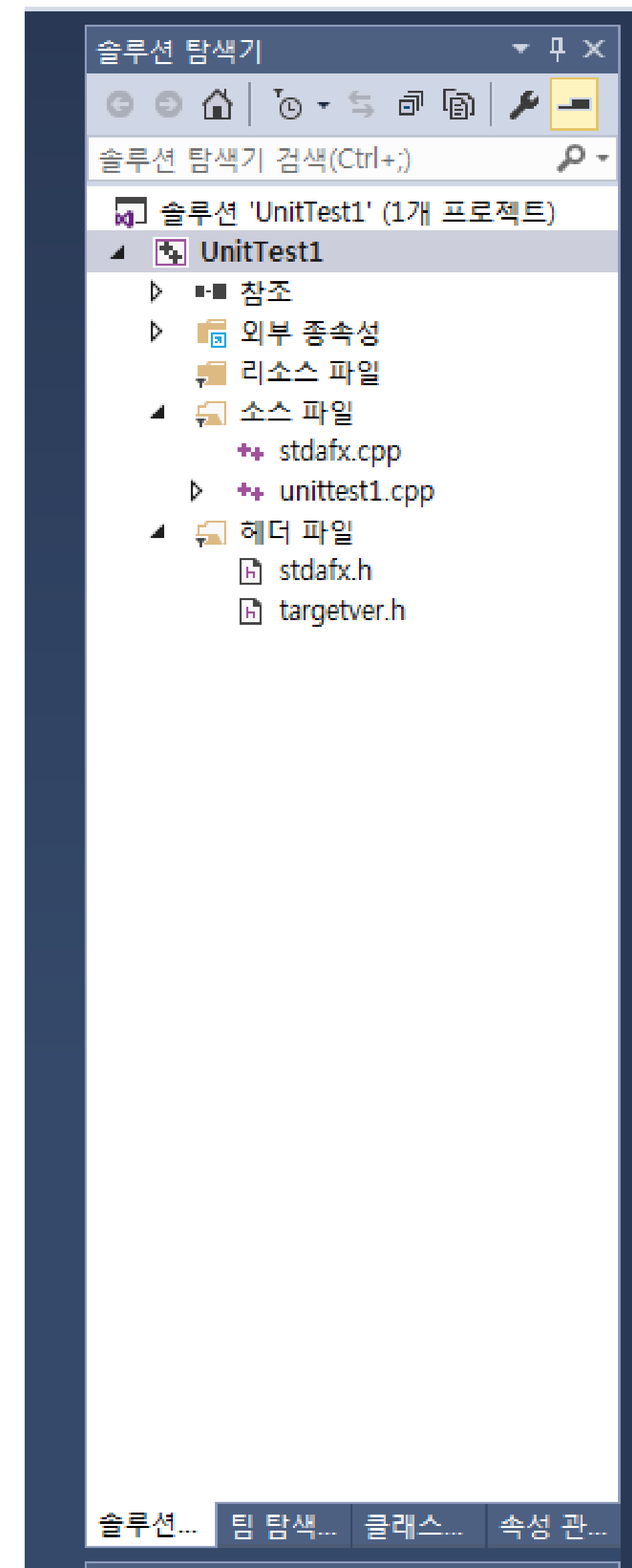
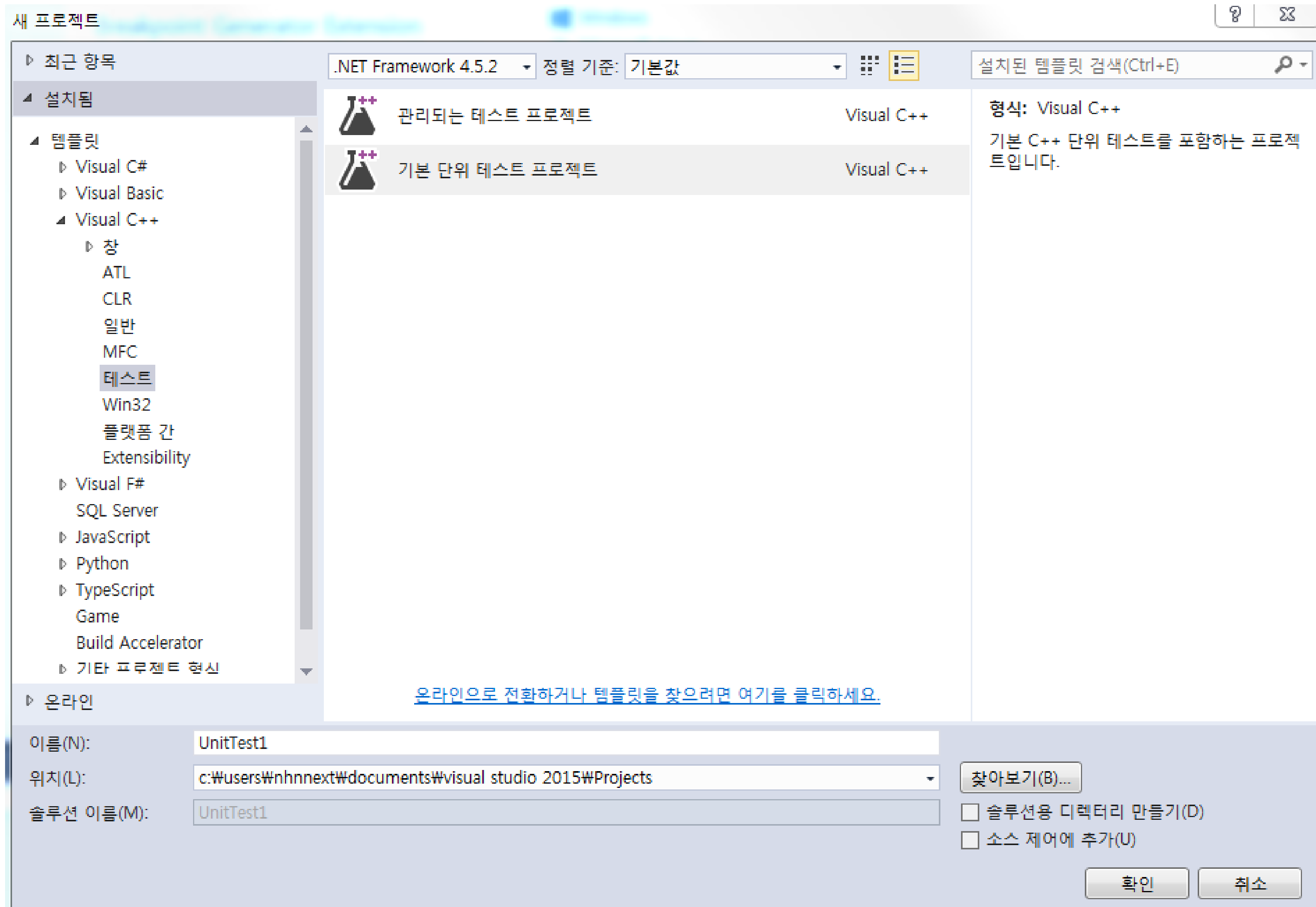
# 좋은 함수를 만들 수 있다

크기가 큰 함수, 복수 기능이 있는 함수는 유닛 테스트 어려움.

**리팩토링 공부**



# Visual Studio의 (순정)C++ 유닛 테스트



## New Project



Recent

.NET Framework 4.6.1

Sort by: Default



Search (Ctrl+E)



Installed

Visual C++

Windows Desktop

General

MFC

Windows Universal

ATL

Test

Azure Data Lake

Stream Analytics

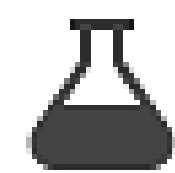
Other Languages

Other Project Types



Native Unit Test Project

Visual C++



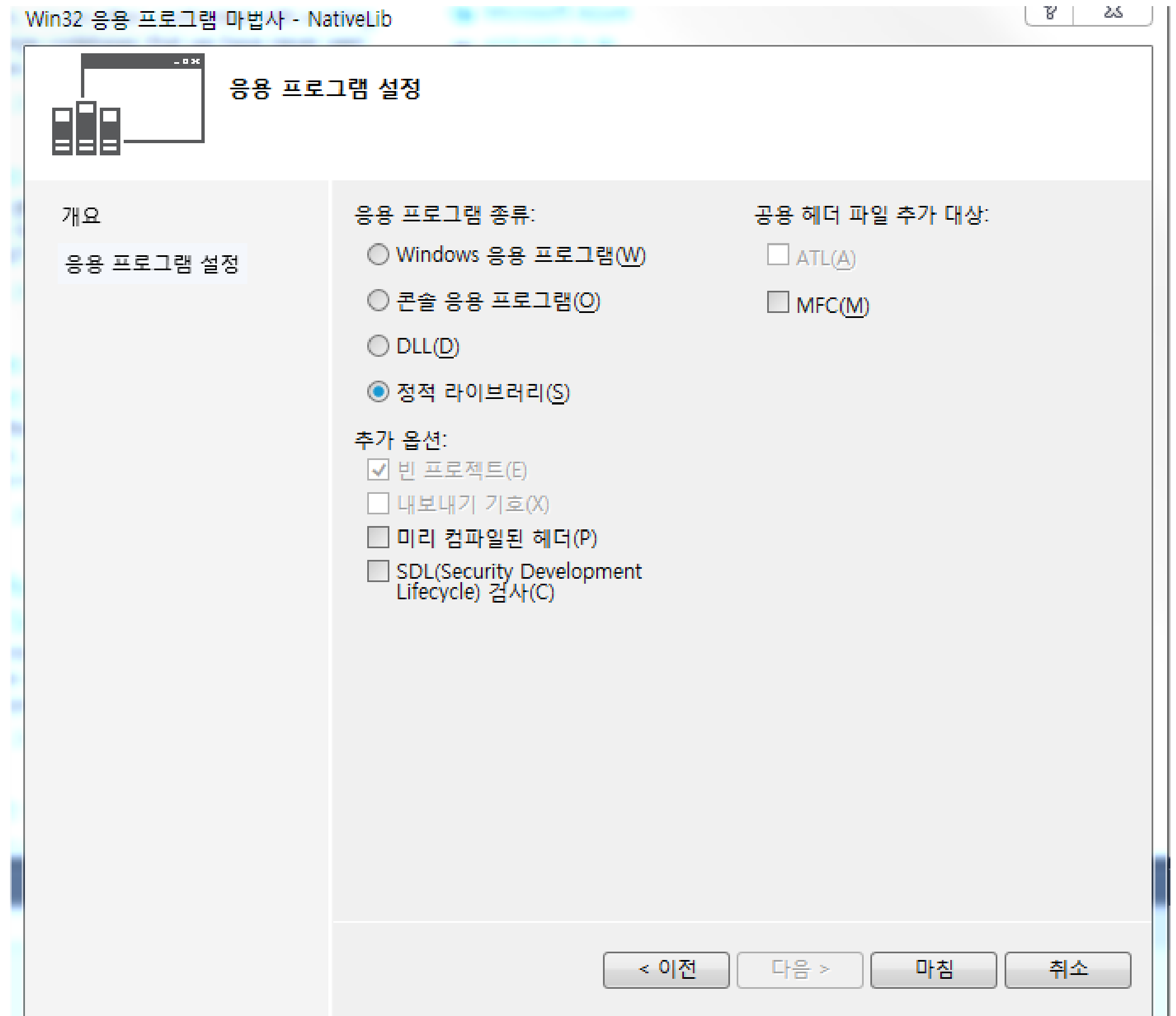
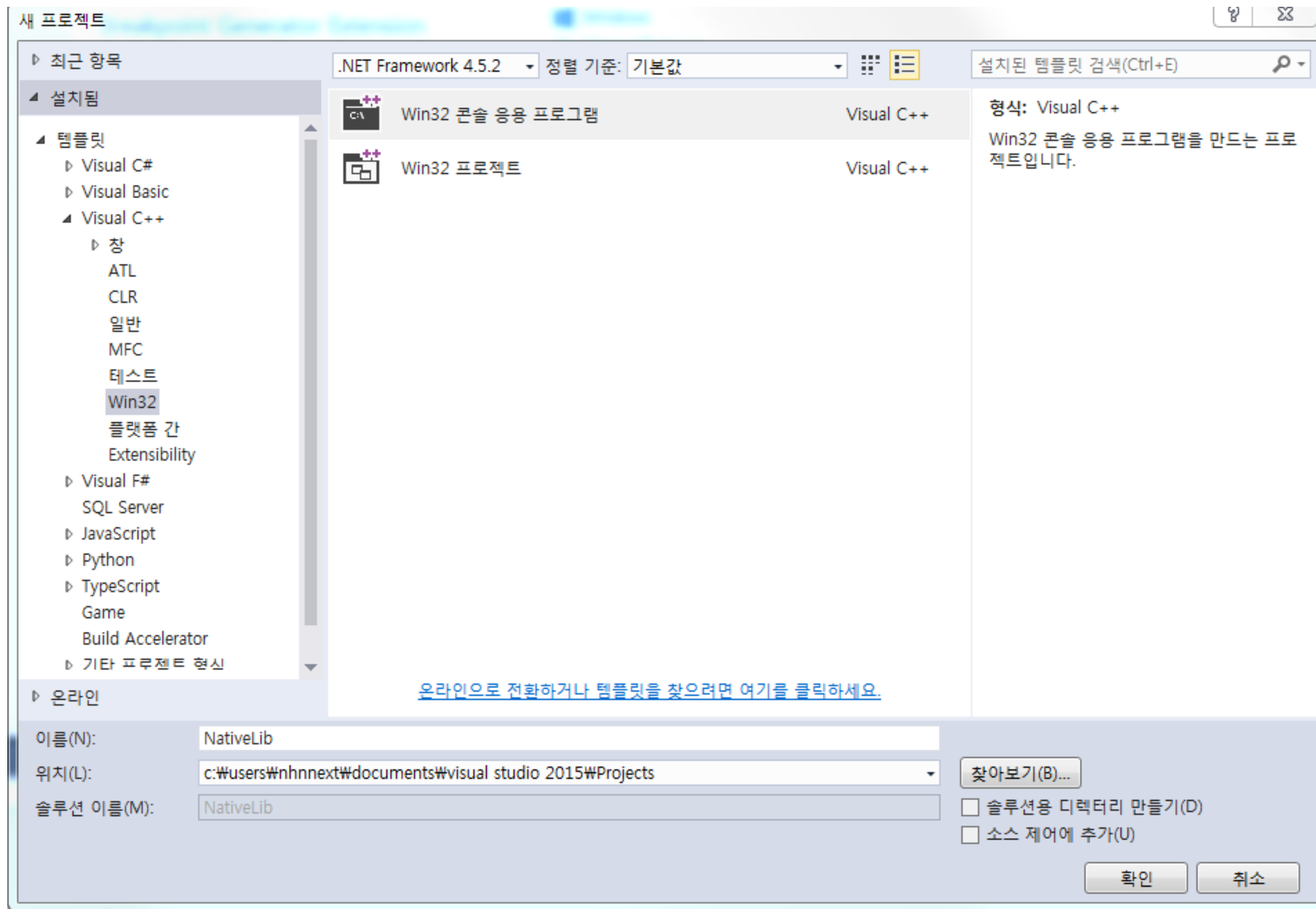
Google Test

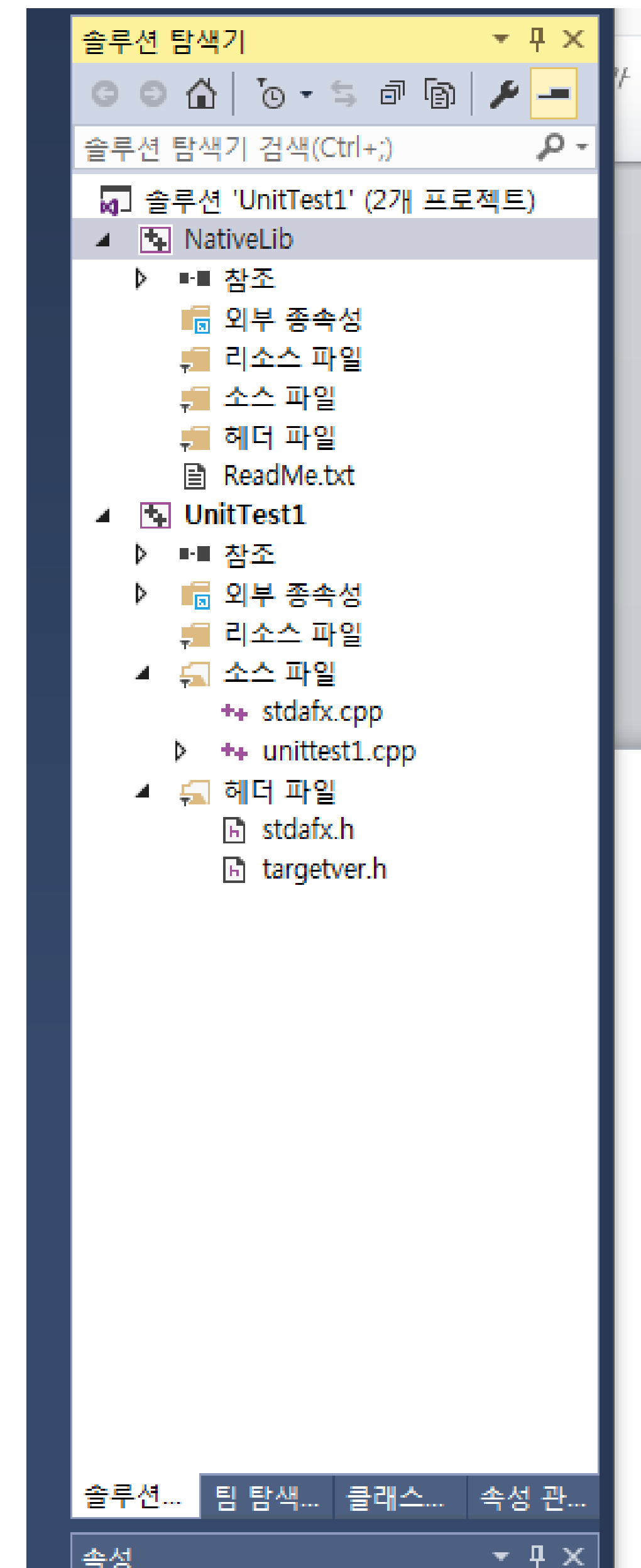
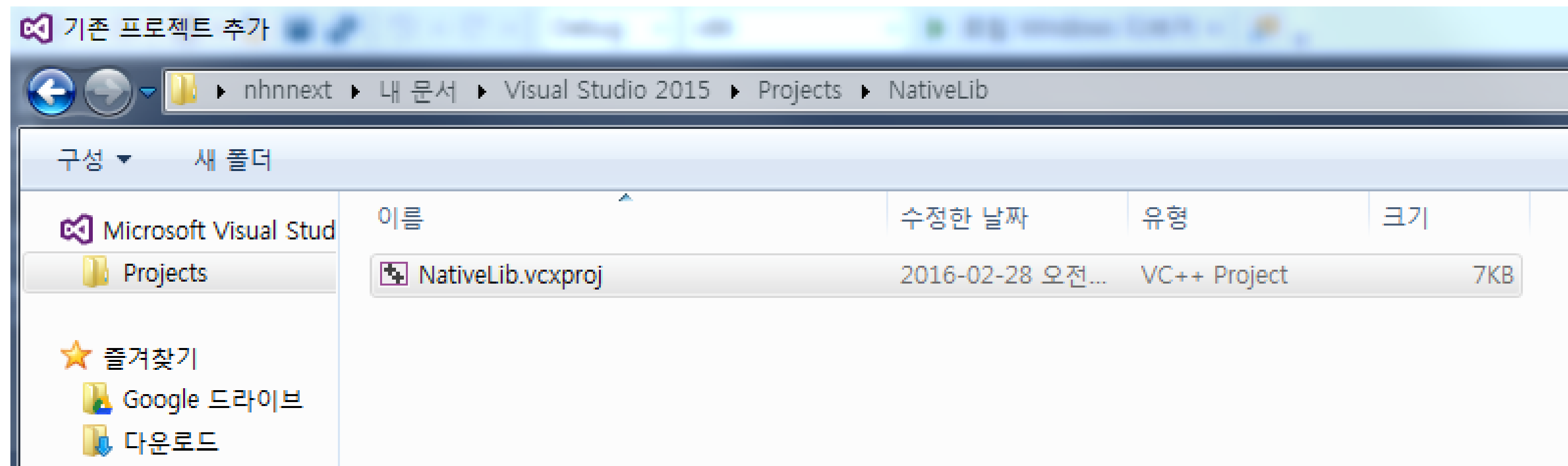
Visual C++

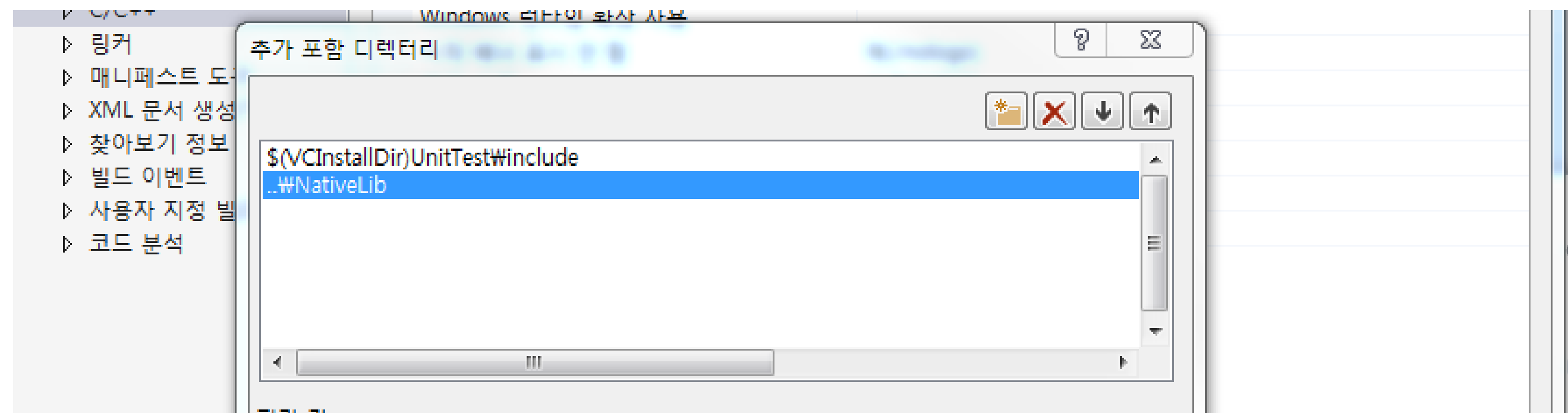
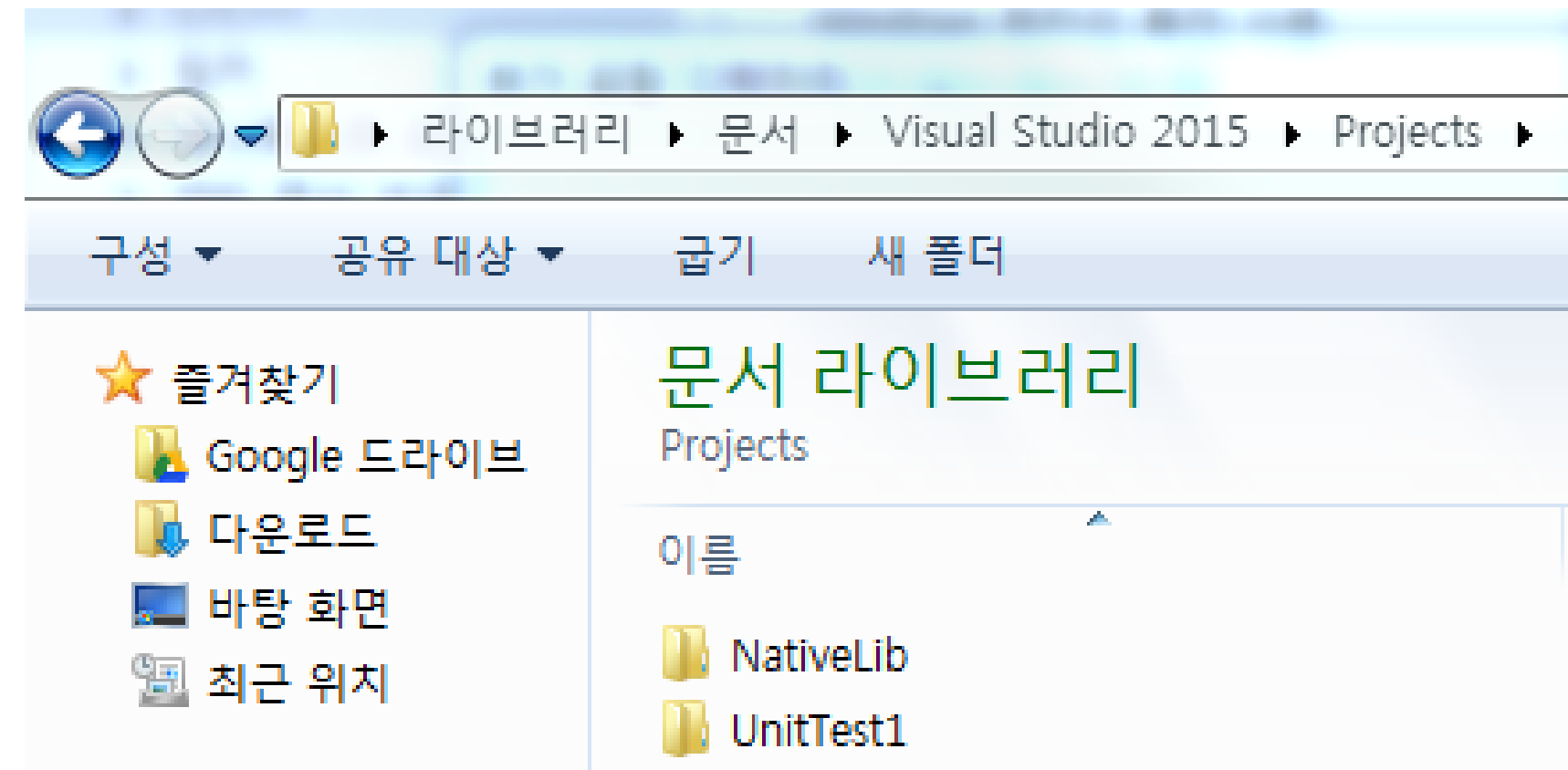
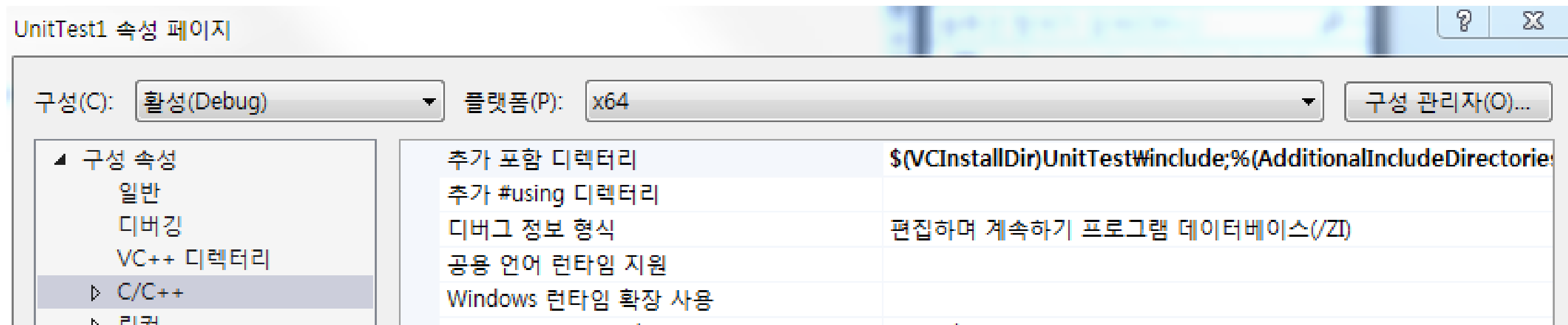
**Type:** Visual C++

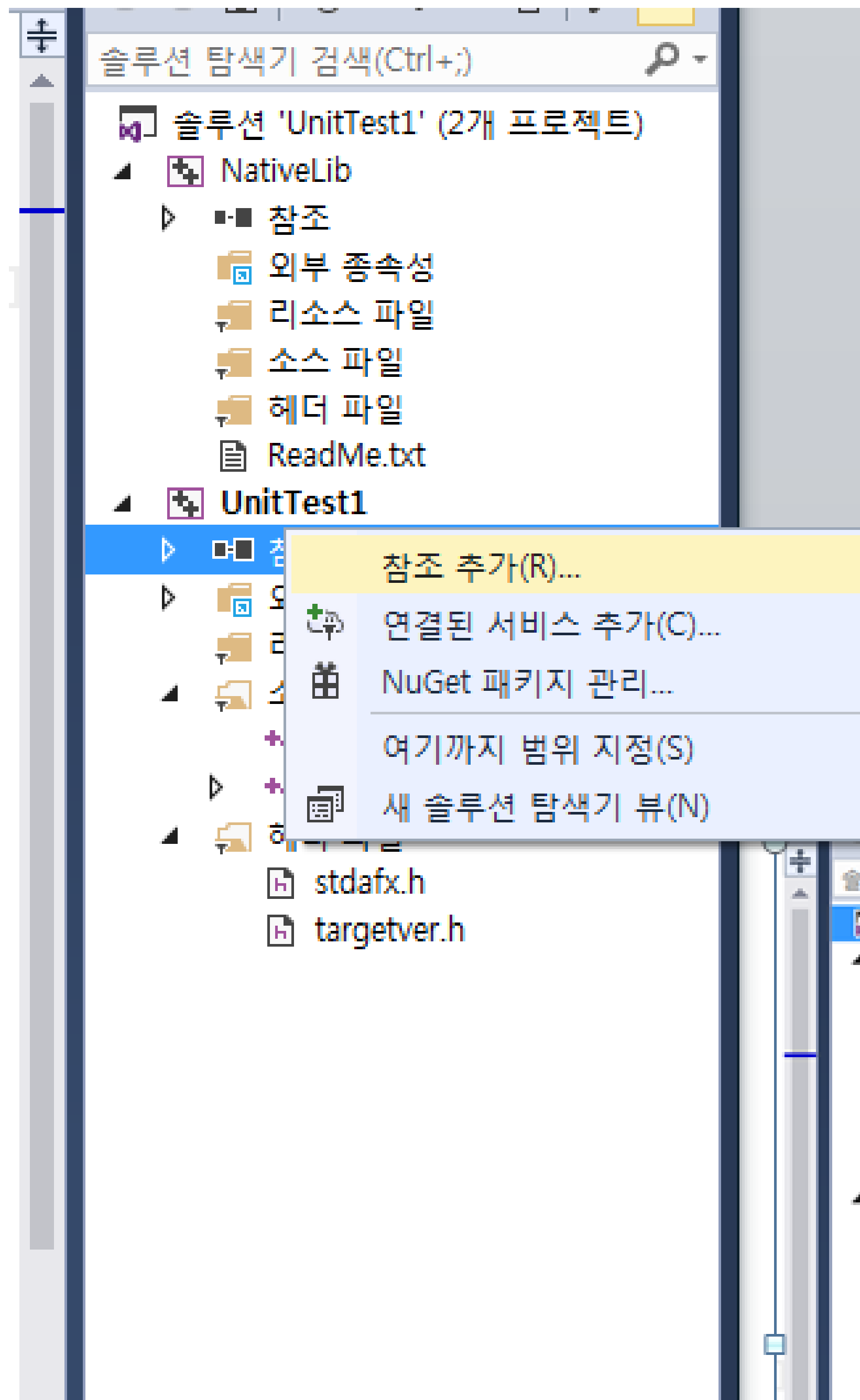
A project that contains native C++ unit tests







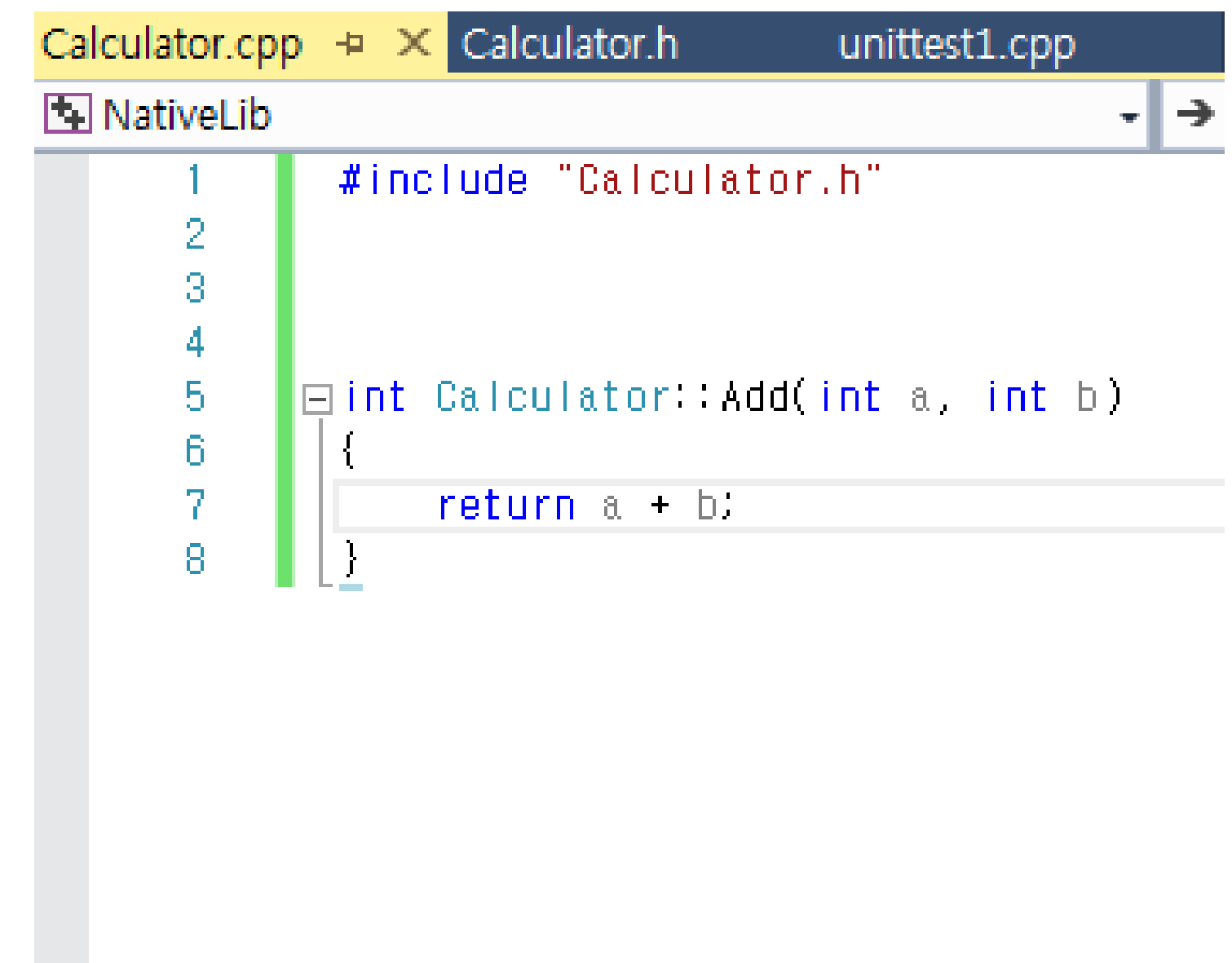
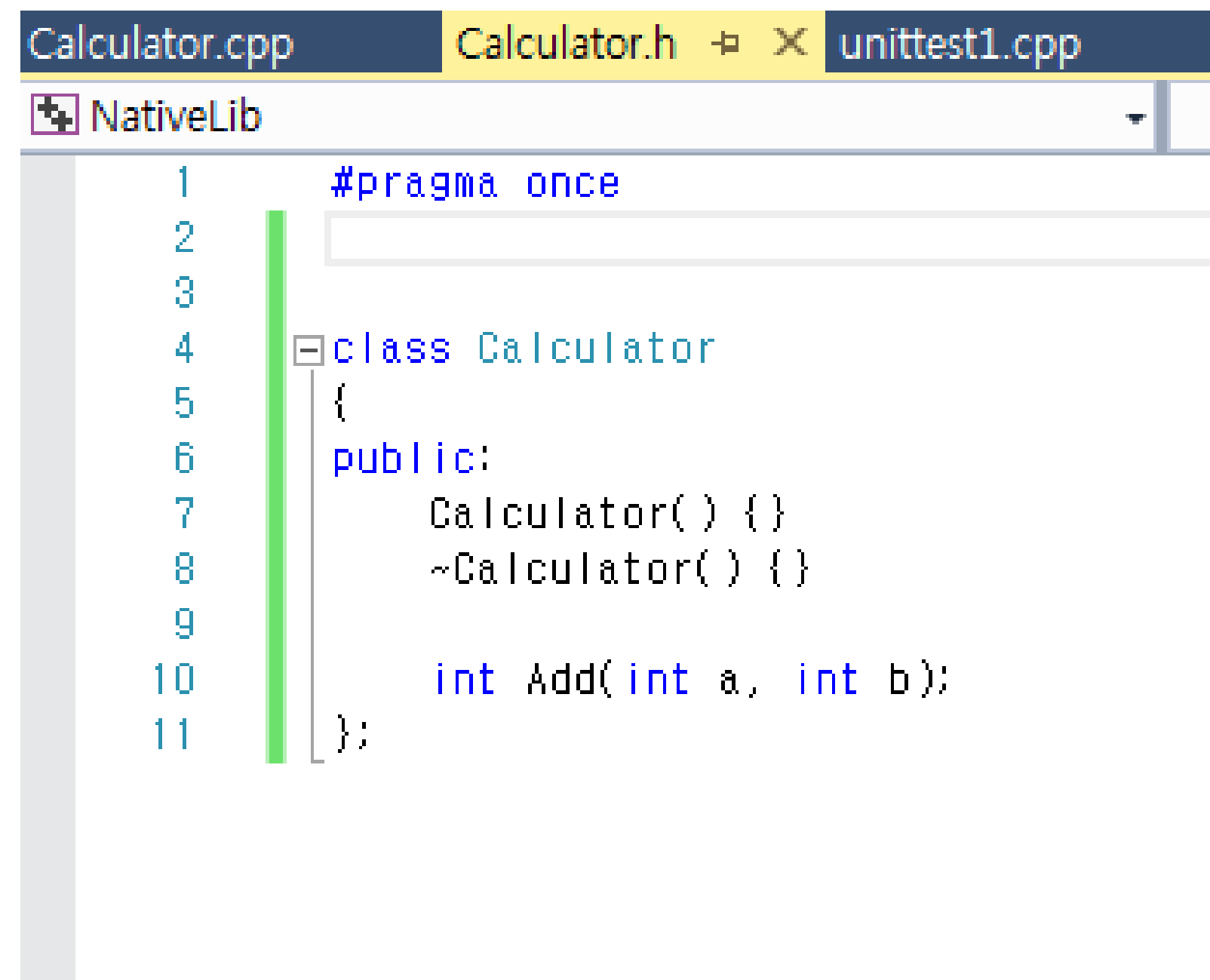
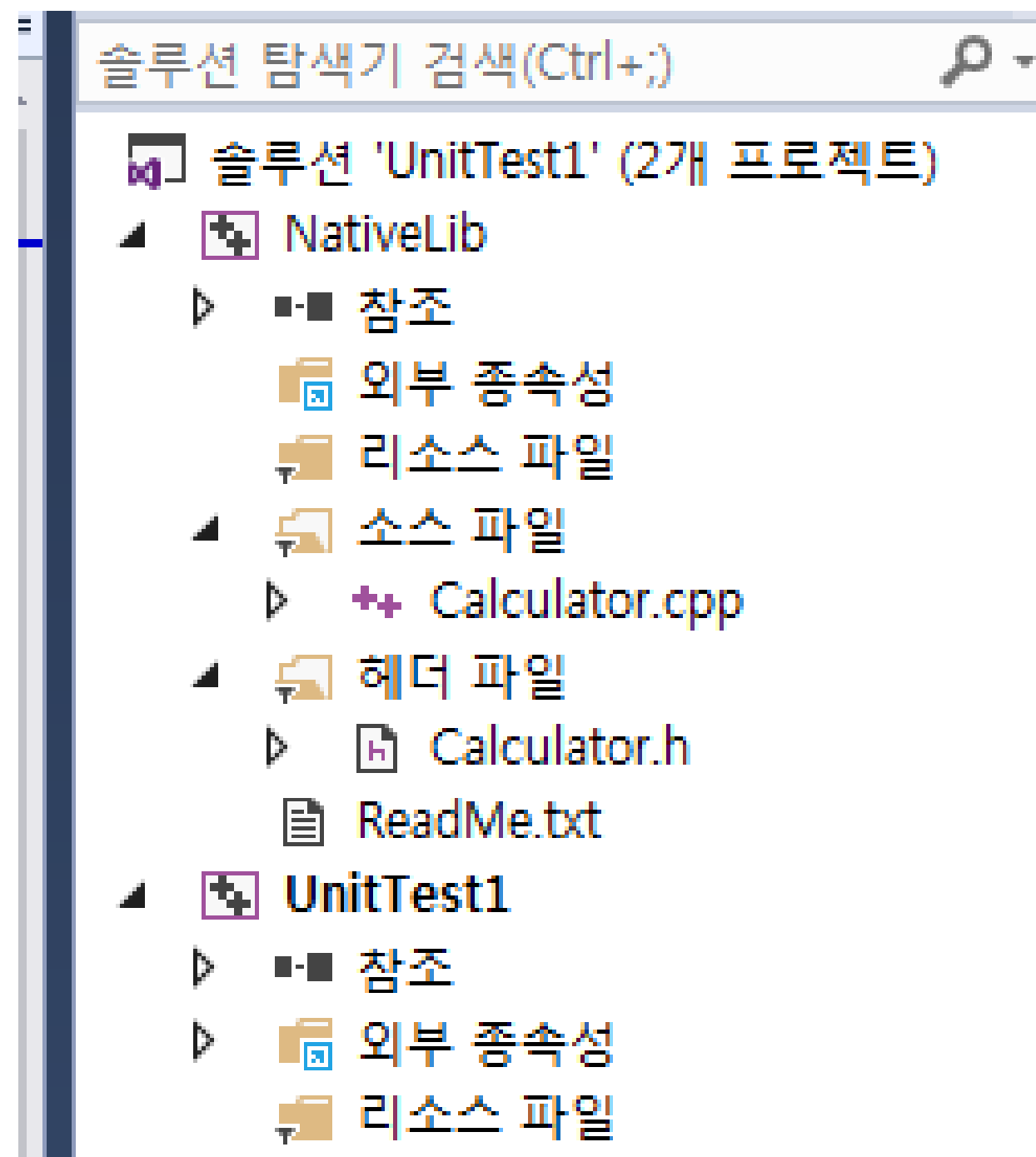




#### 참조 추가

프로젝트			프로젝트 검색(Ctrl)
솔루션			
공유 프로젝트			
			이름:
			NativeLib
			이름:
			NativeLib





Calculator.cppCalculator.hunittest1.cpp

UnitTest1

12345678910111213141516171819202122

```
#include "stdafx.h"
#include "CppUnitTest.h"

#include "..\NativeLib\Calculator.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1
{
    TEST_CLASS(UnitTest1)
    {
    public:
        TEST_METHOD(TestMethod1)
        {
            Calculator cal;
            Assert::AreEqual(3, cal.Add(1, 2));
        }
    };
}
```

솔루션 탐색기

솔루션 탐색기 검색(Ctrl+;)

솔루션 'UnitTest1' (2개 프로젝트)

NativeLib

참조

외부 종속성

리소스 파일

소스 파일

Calculator.cpp

헤더 파일

Calculator.h

ReadMe.txt

UnitTest1

참조

외부 종속성

리소스 파일

소스 파일

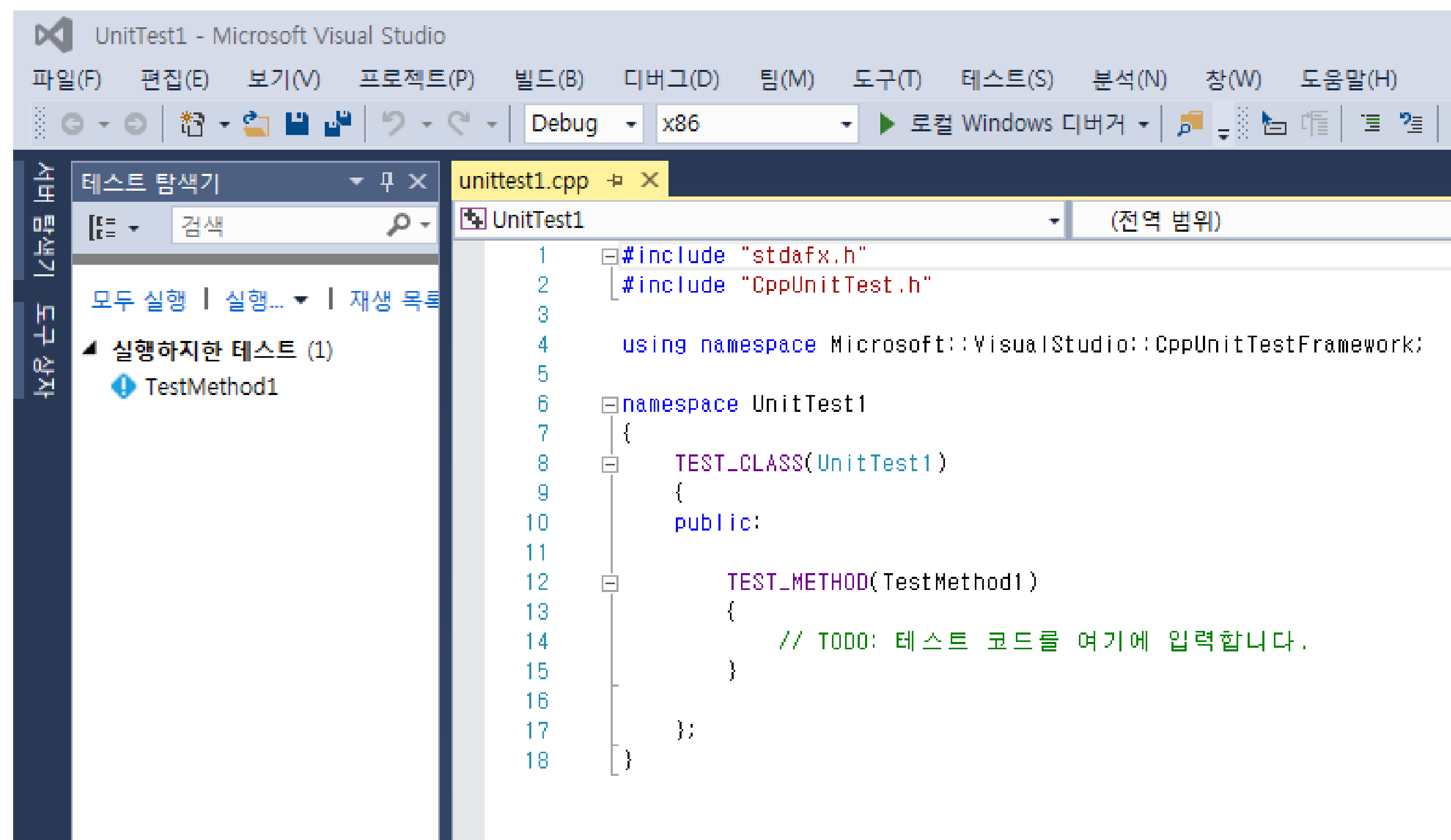
stdafx.cpp

unittest1.cpp

헤더 파일

stdafx.h

targetver.h



테스트 탐색기

검색

모두 실행 | 실행... | 재생 목록

### 성공한 테스트 (1)

✓ TestMethod1 4밀리초

### TestMethod1

소스: unittest1.cpp 줄 15

✓ 테스트 성공 - TestMethod1

경과 시간: 4밀리초

Calculator.cpp

Calculator.h

unittest1.cpp

UnitTest1

UnitTest1::UnitTe

```

1  #include "stdafx.h"
2  #include "CppUnitTest.h"
3
4  #include "..\NativeLib\Calculator.h"
5
6
7  using namespace Microsoft::VisualStudio::CppUnitTes
8
9  namespace UnitTest1
10 {
11     TEST_CLASS(UnitTest1)
12     {
13     public:
14
15         TEST_METHOD(TestMethod1)
16         {
17             Calculator cal;
18             Assert::AreEqual(3, cal.Add(1, 2));
19         }
20     };
21 }
22

```



서버 탐색기 | 도구 상자

테스트 탐색기

검색

모두 실행 | 실행... | 재생 목록

실패한 테스트 (1)  
✖ TestMethod1 137밀리초

TestMethod1

소스: unittest1.cpp 줄 15

✖ 테스트 실패 - TestMethod1  
메시지: Assert가 실패했습니다. 필요한 값:<3> 실제 값:<4>.  
경과 시간: 137밀리초

Stack Trace:  
UnitTest1::TestMethod1()

Calculator.cpp | Calculator.h | unittest1.cpp

UnitTest1

```
1  #include "stdafx.h"
2  #include "CppUnitTest.h"
3
4  #include "..\NativeLib\Calculator.h"
5
6
7  using namespace Microsoft::VisualStudio::CppUnitTe
8
9  namespace UnitTest1
10 {
11     TEST_CLASS(UnitTest1)
12     {
13     public:
14
15         TEST_METHOD(TestMethod1)
16         {
17             Calculator cal;
18             Assert::AreEqual(3, cal.Add(2, 2));
19         }
20     };
21 }
22
```

서버 탐색기 | 도구 상자

테스트 탐색기

검색

모두 실행 | 실행... | 재생 목록

실패한 테스트 (1)

선택한 테스트 실행(S)  
선택한 테스트 디버그(D)  
그룹화 방법(G)  
재생 목록에 추가(T)  
복사(Y) Ctrl+C  
모두 선택(A) Ctrl+A  
테스트 열기(O) F12

### TestMethod1

소스: unittest1.cpp 줄 15

❌ 테스트 실패 - TestMethod1

메시지: Assert가 실패했습니다. 필요한 값:<3> 실제 값:<4>.

경과 시간: 137밀리초

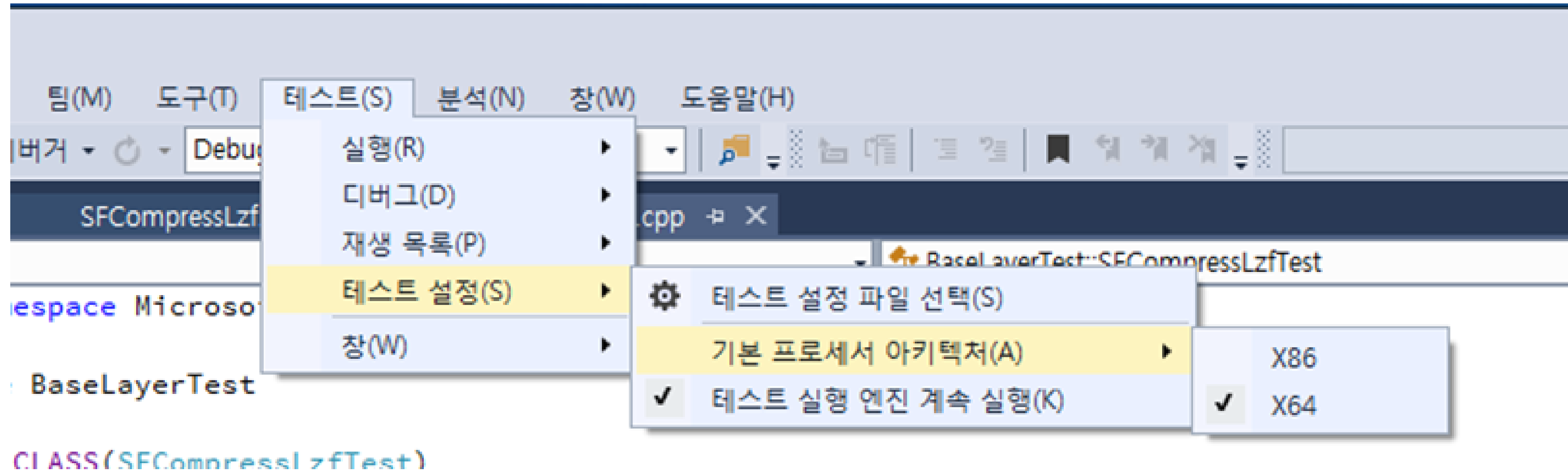
Stack Trace:  
UnitTest1::TestMethod1()

Calculator.cpp | Calculator.h | unittest1.cpp | UnitTest1

UnitTest1

```
1 #include "stdafx.h"
2 #include "CppUnitTest.h"
3
4 #include "..\NativeLib\Calculator.h"
5
6 using namespace Microsoft::VisualStudio::CppUnitTe
7
8 namespace UnitTest1
9 {
10     TEST_CLASS(UnitTest1)
11     {
12     public:
13
14         TEST_METHOD(TestMethod1)
15         {
16             Calculator cal;
17             Assert::AreEqual(3, cal.Add(2, 2));
18         }
19     };
20 }
21
22 }
```

# 64비트 테스트 하기



# VC++ 유닛 테스트 기능 소개

**TEST\_CLASS(class\_name){...};**

TEST\_METHOD를 모은 것이 TEST\_CLASS이다.

테스트 대상 별로 TEST\_CLASS를 정의하는 것이 기본이지만, 상황 혹은 기호 시(예를 들면 정상 계열과 이상 계열 등)세분화해도 상관 없다. 그 때는 단일 소스에 복수의 TEST\_CLASS를 적어 나가도 좋고, 추가/새로운 항목/C++ 유닛 테스트 클래스로 또 다른 소스에 나눠도 좋음.

**TEST\_METHOD(method\_name){...}**

테스트 각 항목에 해당한다. 테스트 대상을 동작시키고 그 결과를 검증한다.

**TEST\_MODULE\_INITIALIZE(method\_name){...}**

**TEST\_MODULE\_CLEANUP(method\_name){...}**

**모든 테스트의 실행 직전/직후에 각각 한번만 호출한다.** 이름 그대로 테스트 모듈의 전 준비/후 처리를 위한 것. 환경 변수/글로벌 변수를 설정하거나 DLL을 LoadLibrary/FreeLibrary 하거나 COM을 초기화/해방하는 등.



**TEST\_CLASS\_INITIALIZE(method\_name){...}**

**TEST\_CLASS\_CLEANUP(method\_name){...}**

TEST\_CLASS 내 일련의 TEST\_METHOD 실행 직전/직후에 한번만 하는 전 준비/뒤처리를 정의한다. 예를 들면 데이터베이스와 Socket과의 연결/절단 이라든가.

**TEST\_METHOD\_INITIALIZE(method\_name){...}**

**TEST\_METHOD\_CLEANUP(method\_name){...}**

각 TEST\_METHOD의 실행 직전/직후에 호출된다.  
테스트 별로 준비/뒤처리를 하는 것이 목적이다.

```
#include "stdafx.h"
#include "CppUnitTest.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace dummy {

    TEST_MODULE_INITIALIZE(test_module_initialize) { Logger::WriteMessage(__FUNCTION__); }
    TEST_MODULE_CLEANUP(test_module_cleanup)      { Logger::WriteMessage(__FUNCTION__); }

    TEST_CLASS(DummyTest) {
    public:
        TEST_CLASS_INITIALIZE(test_class_initialize) { Logger::WriteMessage(__FUNCTION__); }
        TEST_CLASS_CLEANUP(test_class_cleanup)      { Logger::WriteMessage(__FUNCTION__); }

        TEST_METHOD_INITIALIZE(test_method_initialize) { Logger::WriteMessage(__FUNCTION__); }
        TEST_METHOD_CLEANUP(test_method_cleanup)      { Logger::WriteMessage(__FUNCTION__); }

        TEST_METHOD(TestMethod1) { Logger::WriteMessage(__FUNCTION__); }
        TEST_METHOD(TestMethod2) { Logger::WriteMessage(__FUNCTION__); }
        TEST_METHOD(TestMethod3) { Logger::WriteMessage(__FUNCTION__); }
    };
}
```

出力

出力元の表示(S): テスト

----- テストの実行が開始されました -----  
dummy::test\_module\_initialise  
dummy::UnitTest1::test\_class\_initialise  
dummy::UnitTest1::test\_method\_initialise  
dummy::UnitTest1::TestMethod3  
dummy::UnitTest1::test\_method\_cleanup  
dummy::UnitTest1::test\_method\_initialise  
dummy::UnitTest1::TestMethod2  
dummy::UnitTest1::test\_method\_cleanup  
dummy::UnitTest1::test\_method\_initialise  
dummy::UnitTest1::TestMethod1  
dummy::UnitTest1::test\_method\_cleanup  
dummy::UnitTest1::test\_class\_cleanup  
dummy::test module cleanup

エラー一覧

出力

コードカバレッジの結果

검증 함수	설명
AreEqual<T>(const T& expected, const T& actual)	expected == actual
AreEqual(double expected, double actual, double tolerance)	expected-actual  <=  tolerance
AreEqual(float expected, float actual, float tolerance)	
AreEqual(const char* expected, const char* actual, bool ignoreCase =false)	문자열로서 등가
AreEqual(const wchar_t* expected, const wchar_t* actual, bool ignoreCase =false)	
AreSame<T>(const T& expected, const T& actual)	&expected == &actual
AreNotEqual<T>(const T& notExpected, const T& actual)	!(notExpected == actual)
AreNotEqual(double notExpected, double actual, double tolerance)	notExpected-actual  >  tolerance
AreNotEqual(float notExpected, float actual, float tolerance)	

검증 함수	설명
AreNotEqual(const char* notExpected, const char* actual, bool ignoreCase=false)	문자열로서 등가가 아니다
AreNotEqual(const wchar_t* notExpected, const wchar_t* actual, bool ignoreCase=false)	
AreNotSame<T>(const T& notExpected, const T& actual)	!(&notExpected == &actual)
IsNull<T>(const T* actual)	actual == nullptr
IsNotNull<T>(const T* actual)	!(actual == nullptr)
IsTrue(bool condition)	condition == true
IsFalse(bool condition)	condition == false
Fail()	실패
AreEqual<T>(T^ expected, T^ actual)	expected->Equals(actual)
AreEqual(string^ expected, string^ actual, bool ignoreCase=false)	문자열로서 등가
AreSame<T>(T% expected, T% actual)	%expected == %actual



검증 함수	설명
AreNotEqual<T>(T^ notExpected, T^ actual)	!expected->Equals(actual)
AreNotEqual(string^ notExpected, string^ actual, bool ignoreCase =false)	문자열로서 등가가 아니다
AreNotSame<T>(T% notExpected, T% actual)	!(%notExpected == %actual)
IsNull<T>(T^ actual)	actual == nullptr
IsNotNull<T>(T^ actual)	!(actual == nullptr)
ExpectedException<Exception, Functor>(Functor functor)	functor() <sup>0</sup>   Exception을 throw 한다
ExpectedException<Exception, ReturnType>(ReturnType (*func)())	func() <sup>0</sup>   Exception을 throw 한다

# Visual C++ Team Blog

C++ tutorials, C and C++ news, and information about the C++ IDE Visual Studio from the Microsoft C++ team.

## C++ Unit Testing in Visual Studio

April 19, 2017 by [Augustin Popa](#) // [18 Comments](#)



Share 59

53

0

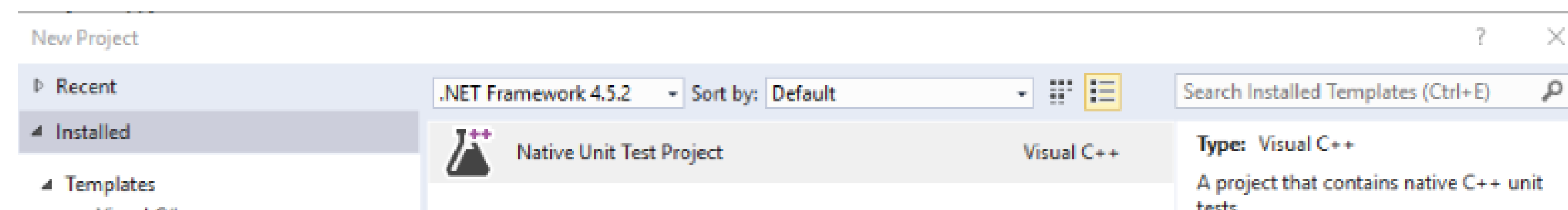
Testing is an increasingly important part of a software development workflow. In many cases, it is insufficient to test a program simply by running it and trying it out – as the scope of the project gets more involved, it becomes increasingly necessary to be able to test individual components of the code on a structured basis. If you're a C++ developer and are interested in unit testing, you'll want to be aware of Visual Studio's unit testing tools. This post goes through just that, and is part of a series aimed at new users to Visual Studio. This blog post goes over the following concepts:

1. [Setting Up Unit Testing](#)
2. [The Microsoft Native C++ Unit Test Framework](#)
3. [Using the Test Explorer to Run Tests in the IDE](#)
4. [Determining Unit Test Code Coverage](#)

## Setting Up Unit Testing

The easiest and most organized way to set up unit tests is to create a separate project in Visual Studio for your tests. You can create as many test projects as you want in a solution and connect them to any number of other Visual Studio projects in that solution that contain the code you want to test. Assuming you already have some code that you want to test, simply follow these steps to get yourself set up:

1. Right-click your solution and choose *Add > New > Project*. Click the *Visual C++* category, and choose the *Test* sub-category. Select *Native Unit Test Project*, give the project a descriptive name, and then click *OK*.



<https://blogs.msdn.microsoft.com/vcblog/2017/04/19/cpp-testing-in-visual-studio/>

① 텍스트 위로 마우스를 움직여서 팝업 창에 영문 텍스트를 표시할 수 있습니다.

사용

×

Docs / Visual Studio / 테스트

피드백 편집 공유 어두움 영어로 읽기

제목으로 필터링

> 테스트 탐색기를 사용하여  
단위 테스트 실행

명령줄에서 테스트 실행

단위 테스트를 64비트  
프로세스로 실행

.runsettings 파일을 사  
용하여 단위 테스트 구  
성

> 관리 코드의 단위 테스  
트 작성

▼ C/C++ Code용 단위 테  
스트 작성

C++에 대한 Microsoft  
단위 테스트 프레임워  
크 사용

Google C++ 테스트  
프레임워크 사용

Boost.Test 사용

CTest 사용

C/C++ DLL의 단위 테  
스트 작성

연습: C++ DLL에 대한

# Visual Studio에서 C/C++에 대한 단위 테스트 작성

2017. 11. 04. • 읽는 데 8분 • 참가자

## 이 문서의 내용

[기본 테스트 워크플로](#)

[참고 항목](#)

다른 언어처럼 **테스트 탐색기** 창을 사용하여 C++ 단위 테스트를 작성하여 실행할 수 있습니다. 테스트 탐색기에 대한 자세한 내용은 [테스트 탐색기를 사용하여 단위 테스트 실행](#)을 참조하세요.

### 참고

Live Unit Testing, Coded UI Tests 및 IntelliTest 등의 일부 기능은 C++에서 지원되지 않습니다.

Visual Studio에는 다음 C++ 테스트 기능이 포함되어 있으며 추가 다운로드가 필요하지 않습니다.

- C++에 대한 Microsoft 단위 테스트 프레임워크
- Google Test
- Boost.Test
- CTest

이 문서의 영문 버전은 <https://docs.microsoft.com/ko-kr/visualstudio/test/writing-unit-tests-for-c-cpp>

# 다른 C++ 유닛테스트 프레임워크

## GTest

<https://github.com/google/googletest>

## Boost.Test

[http://www.boost.org/doc/libs/1\\_60\\_0/libs/test/doc/html/index.html](http://www.boost.org/doc/libs/1_60_0/libs/test/doc/html/index.html)

## Catch

<https://github.com/philsquared/Catch>

### ▼ C/C++ Code용 단위 테스트 작성

C++에 대한 Microsoft 단위 테스트 프레임워크 사용

Google C++ 테스트 프레임워크 사용

Boost.Test 사용

CTest 사용

C/C++ DLL의 단위 테스트 작성

### 참고 항목

다른 언어처럼 테스트에 대한 자세한 내용은

### ① 참고

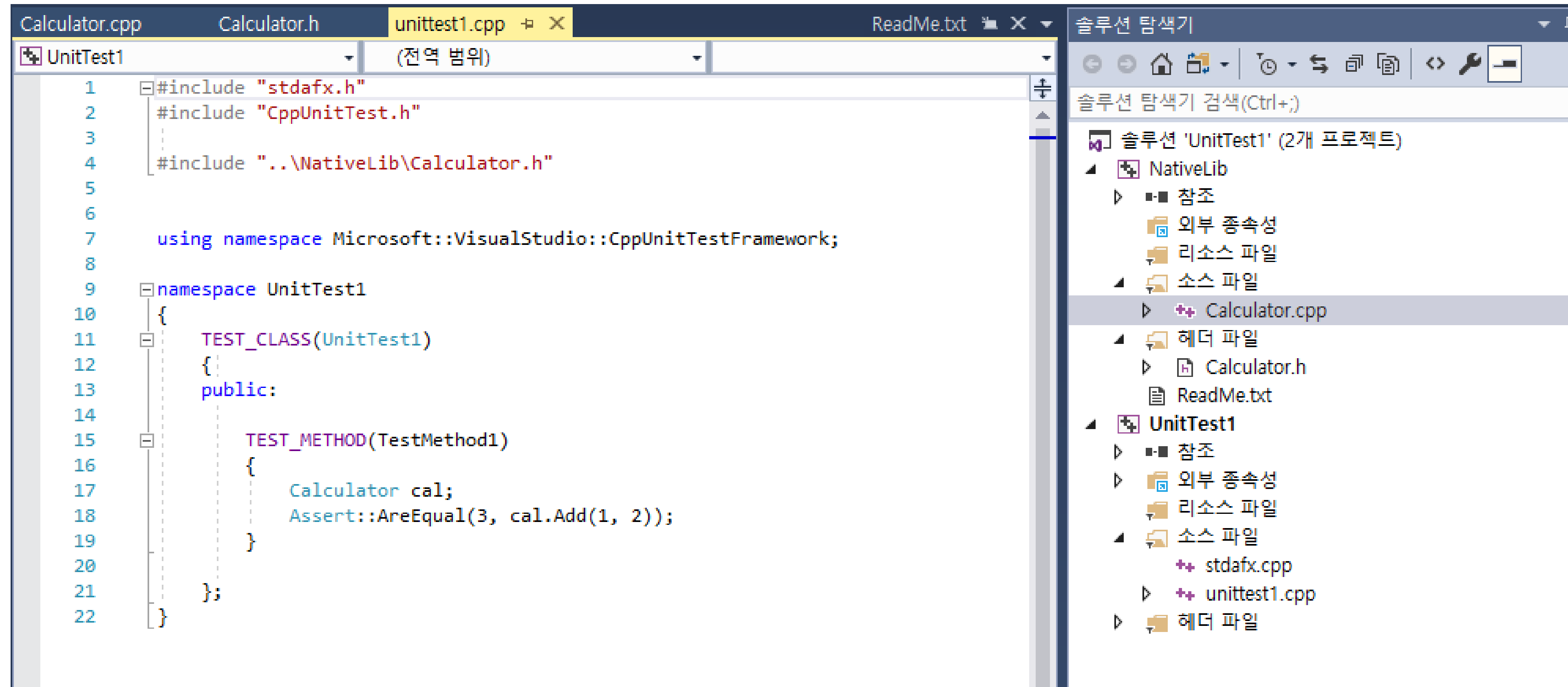
Live Unit Testing

Visual Studio에는

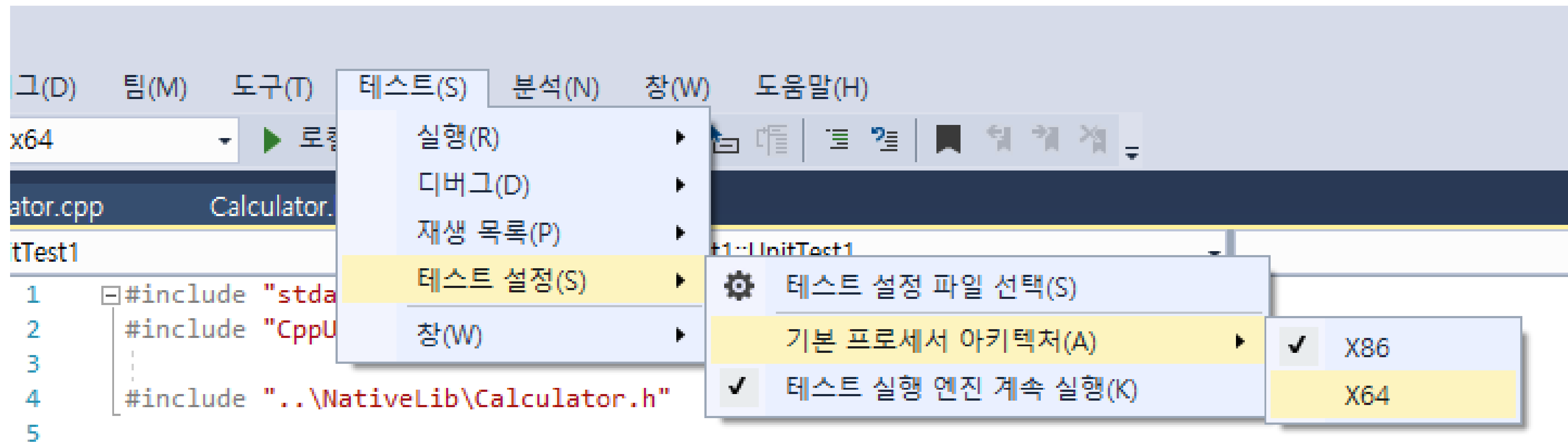
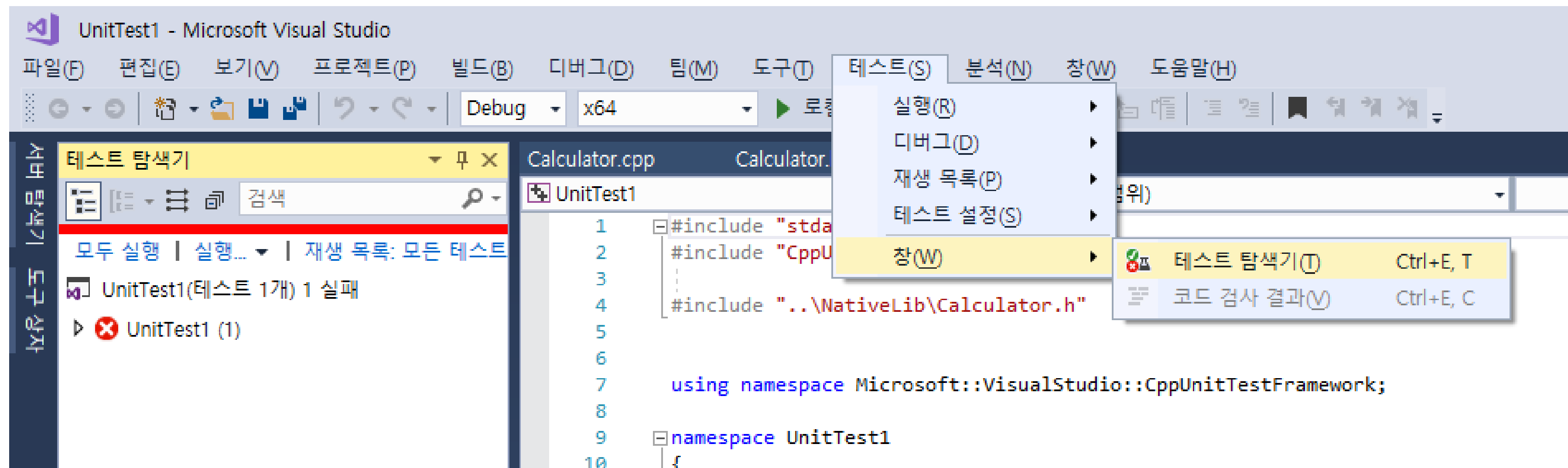
Visual Studio에는 다음 C++ 테스트 기능이 포함되어 있으며 추가 다운로드가 필요하지 않습니다.

- C++에 대한 Microsoft 단위 테스트 프레임워크
- Google Test
- Boost.Test
- CTest

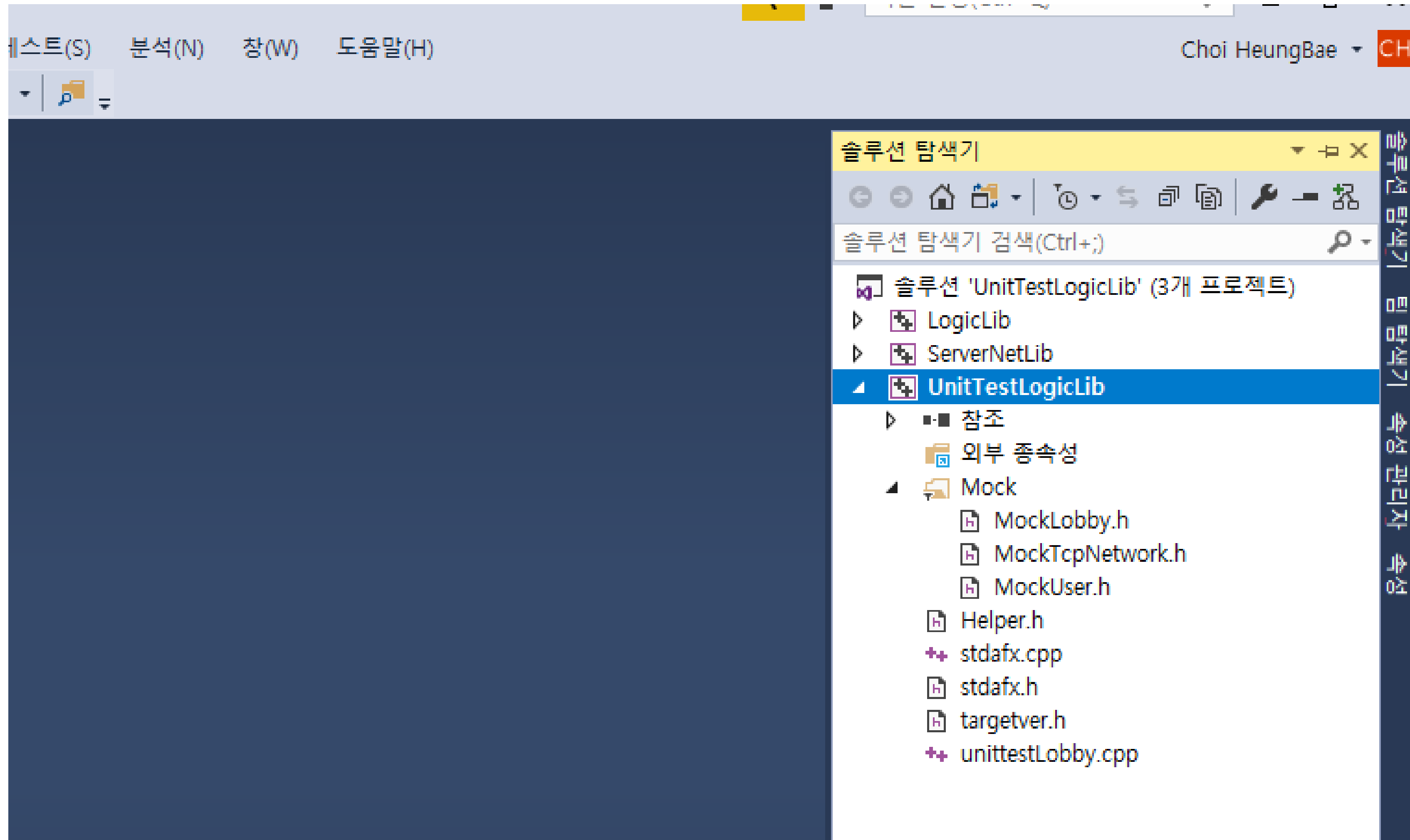
# Demo – 자주 보는(?) 계산기







# Demo – 채팅 서버



EXE/DLL

Static Library

Unit Tests

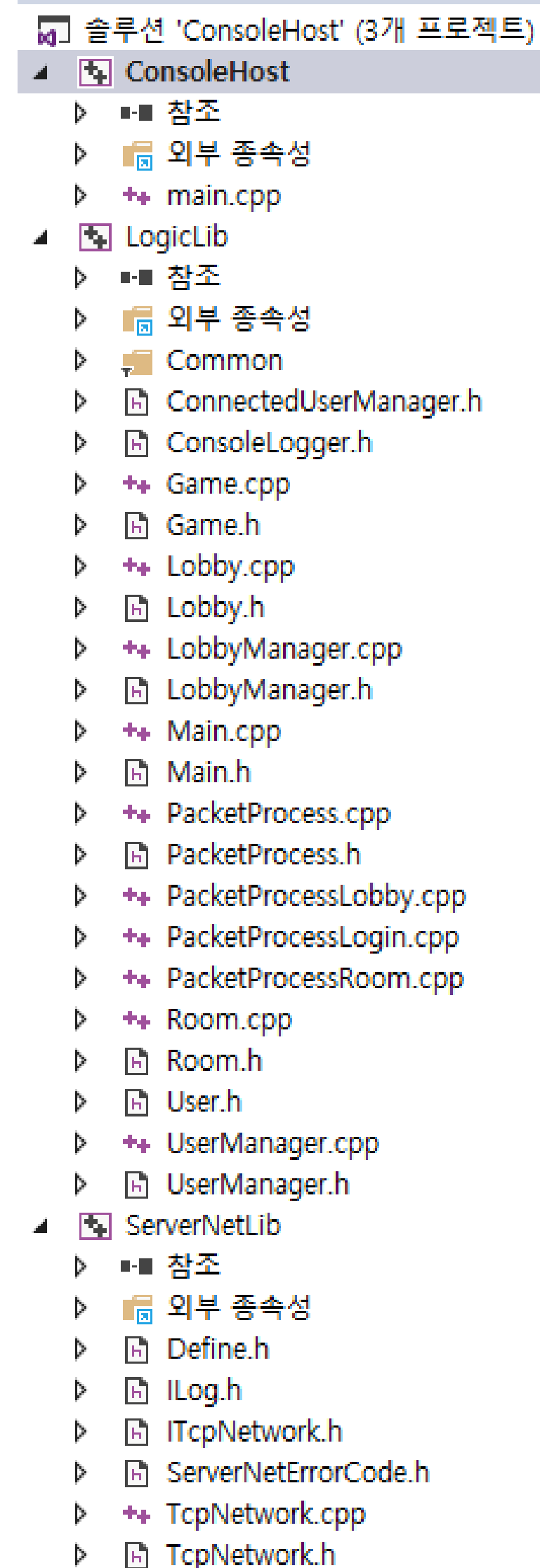


클라이언트/서버 공통 라이브러리 프로젝트

네트워크 라이브러리 프로젝트

게임 로직 라이브러리 프로젝트

플랫폼 의존 실행 파일 프로젝트



# 유닛테스트를 할 때는 해당 기능만 테스트 해야 한다

```
class class MembserManager
{
    .....
    void AddMember(Player* pNewPlayer)
    {
        // 조건 조사
        // 컨테이너에 추가
        // 네트워크로 다른 멤버들에게 통보
    }
};
```

테스트에서 제외

# 인터페이스를 사용한 클래스 정의, mock이 필요

```
class INetwork
{
};

class Network : public INetwork
{
};
```

```
class MembserManager
{
public:
    MembserManager(INetwork* pNetwork);
    {
        m_pNetwork = pNetwork;
    }

private:
    INetwork* m_pNetwork;
};
```

```
class MockNetwork : public INetwork
{
public:
    virtual Send(char* pData) override
    {
        ....
        m_SendDatas.push_back(packet);
    }

    std::vector<Packet> m_SendDatas;
};
```

```
auto pMockNetwork = new MockNetwork();
auto MemberMgr =
    MembserManager(pMockNetwork);
```



전역 변수  
`static` 변수

사용하지 않는 것이...

# 채팅 서버 유닛 테스트

jacking75 / nhn\_next\_gameserver\_lab\_2017\_chatServer

Unwatch

1

Star

16

Fork

4

Code

Issues 0

Pull requests 0

Projects 0

Wiki

Insights

Settings

NHN Next의 2017년 게임서버 과정 수업에 사용할 채팅 서버. select 기반

Edit

cpp

network

select

winsock

vs2017

windows

nhnnext

Manage topics

9 commits

1 branch

0 releases

1 contributor

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download

jacking75

No commit message

Latest commit 70e26e5 on 30 Jan

Bin	No commit message	6 months ago
ChatClient	No commit message	6 months ago
ChatServer	No commit message	6 months ago
Common	No commit message	a year ago
Chat Server-Client.pptx	No commit message	6 months ago
README.md	Update README.md	a year ago

# 마무리...



# 유닛 테스트 효용성

유닛 테스트가 절대적인 것은 아니다.

유닛 테스트가 없는 것보다는 훨씬 좋다.

유닛 테스트를 해보지 않고 비판하지 말자.

유닛 테스트를 하지 않으면 대안을 제시하고 실천하자.

유닛 테스트를 하면 코드 리팩토링도 얻을 수 있다

# JavaScript, Ruby, Python

## C#, Java

## C++

C++은 유닛 테스트가 다른 언어에 비해 어려움.  
C#의 경우 유닛 테스트에서 private도 쉽게 접근 가능

좋은 함수가 아니면  
유닛 테스트도 힘들다

리팩토링

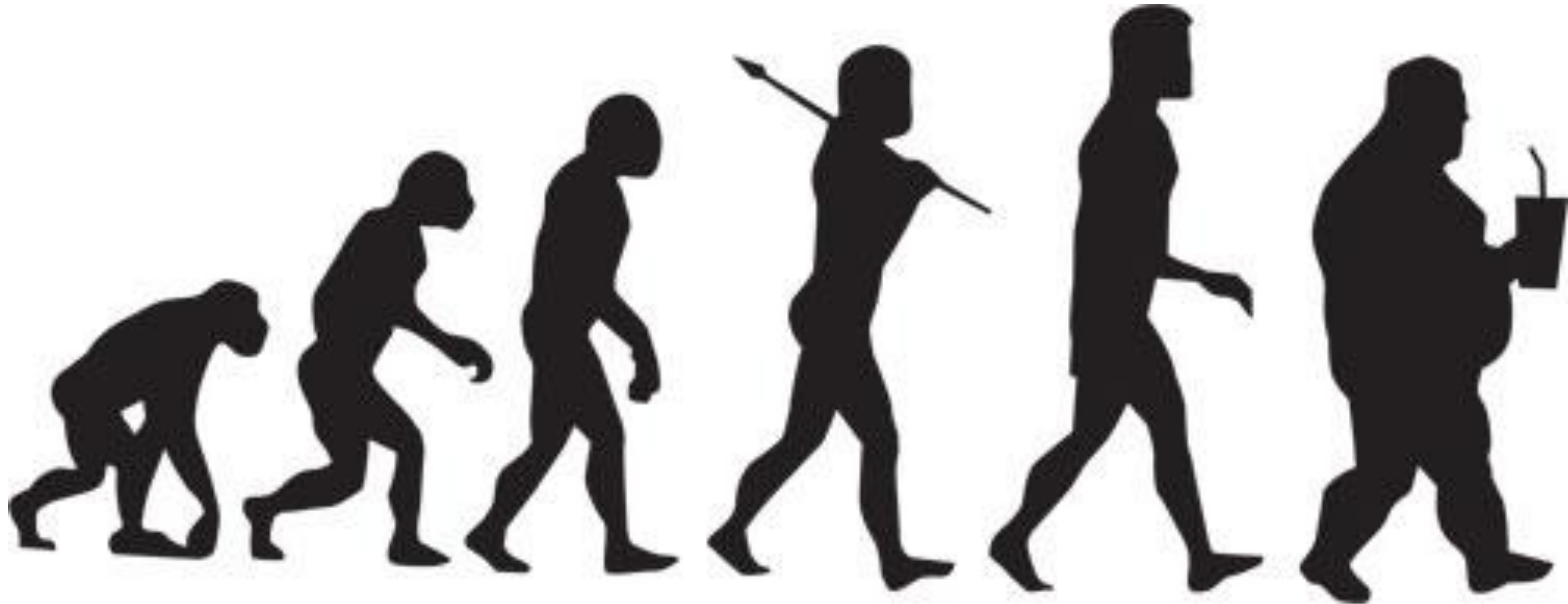
# 빨리 끝내야 하는 일

- 1) 개발하고 바로 시연 테스트 하기
- 2) 유닛 테스트 만든 후 시연 테스트 하기

아마 1번의 유혹에 많이 빠질 듯(사실 저도... ;;;;)



유닛 테스트 좋은지는 알겠는지 습관을 들이기 힘들다  
나와 주위의 도움이 필요



게임 서버는 대부분 분산 서버 구조,  
테스트 중 버그가 발생하면 서버 수정하고,  
배포하고, 다시 시작.  
너무 많은 시간을 소비하고 피곤함

유닛 테스트를 먼저해서  
시연 테스트를 빨리 끝내야 한다

**꽤 진행된 프로젝트,  
누군가의 반대(특히 리더급)**

**너무 힘들다 ㅠㅠ**

# 추천 하고 싶은 문서

생산적인 개발을 위한 지속적인 테스트

<http://www.slideshare.net/birdkr/ss-2183466>

**감사합니다.**